EDGE RATCHET AND SIMULATED ANNEALING TO IMPROVE RF SCORE OF THE SUPERTREE OF LIFE

A Thesis

by

REZA MANSHOURI

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Tiffani Williams
Co Chair of Committee,	Jennifer Welch
Committee Member,	Mariana Mateos
Head of Department,	Dilma Da Silva

December 2017

Major Subject: Computer Science

Copyright 2017 Reza Manshouri

ABSTRACT

Constructing the Supertree of Life can provide crucially valuable knowledge to address many critical contemporary challenges such as fighting diseases, improving global agriculture, and protecting ecosystems to name a few. However, building such a tree is among the most complicated and challenging scientific problems. In the case of biological data, the true species tree is not available. Hence, the accuracy of the supertree is usually evaluated based on its similarity to the given source input trees.

In this work, we aim at improving the accuracy of the supertree in terms of its cumulative Robinson Foulds (RF) distance to the source trees. This problem is NP-hard. Therefore, we have to resort to heuristic algorithms. We have two main contributions in this work. First, we propose a new technique, Edge Ratchet, which is used in a hill-climbing based algorithm to deal with local optimum problem. Second, we develop a Simulated Annealing algorithm to minimize total RF distance of the supertree to the source trees. Our results demonstrate that these two algorithms are able to improve the accuracy of the best existing supertree algorithms with regard to RF distance.

DEDICATION

To my parents, my sisters, and my brother for all their support over the years.

ACKNOWLEDGMENTS

First and foremost, I would like to thank my parents for their love and support over these years. I would also like to specially thank my sisters and my brother for all their love and encouragement. Thanks for always believing in me, and pushing me toward my dreams.

I would like to express my special thanks to my advisor, Dr. Tiffani Williams, for her support, patience, and guidance throughout my graduate studies. Thanks for providing me the opportunity to better know myself, and become a stronger and more confident person.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by a thesis committee consisting of Professor Tiffani Williams and Professor Jennifer Welch of the Department of Computer Science & Engineering and Professor Mariana Mateos of the Department of Wildlife and Fisheries Sciences.

All work for the thesis was completed by the student under the advisement of Professor Tiffani Williams of the Department of Computer Science & Engineering.

Funding Sources

This work was made possible in part by NSF grant under Grant Number DEB-1208337.

TABLE OF CONTENTS

Page

ABSTRACT ii

DEDICATION iii

ACKNOWLEDGMENTS iv

CONTRIBUTORS AND FUNDING SOURCES v

TABLE OF CONTENTS vi

LIST OF FIGURES viii

1. INTRODUCTION AND BACKGROUND INFORMATION 1

  1.1 Supertree of Life 1
  1.2 Our Research 6
    1.2.1 Edge Ratchet Robinson Foulds Supertree Algorithm 7
    1.2.2 Simulated Annealing Algorithm 9
  1.3 Background Information and Terminology 10
    1.3.1 Phylogenetic Tree 10
    1.3.2 Rooted vs Unrooted Trees 12
    1.3.3 Newick Format 13

2. LITERATURE REVIEW 15

  2.1 MR Based Supertrees 15
  2.2 Graph Based Supertrees 20
  2.3 Distance Based Supertrees 22

3. EDGE RATCHET ROBINSON FOULDS SUPERTREE ALGORITHM 26

  3.1 Robinson Foulds Distance 26
  3.2 Robinson Foulds Supertree Problem 27
  3.3 Designing SPR Hill-Climbing Algorithm 28
  3.4 Solving Local Search Problem by RFS Algorithm 30
  3.5 Edge Ratchet Technique to Deal With Local Optimum Problem 35
  3.6 ER-RFS Algorithm 38

3.7 Experimental Results 42
3.7.1 Datasets 42
3.7.2 Measurements and Other Supertree Algorithms 42
3.7.3 Calculating RF Score 43
3.7.4 Calculating Parsimony Score 43
3.7.5 ER Configuration 46
3.7.6 Terminology 47
3.7.7 Results 47

4. SIMULATED ANNEALING ALGORITHM 52

4.1 Motivation and Algorithm Description 52
4.2 Generating a Random Neighbor 53
4.3 Annealing Schedule, the Challenging Part 55
4.4 Our Strategy to Choose Annealing Schedule 56
4.5 Experimental Results, SA-RR vs SA-RB 59

5. SUMMARY AND CONCLUSIONS 64

5.1 Overall Comparison 64
5.2 Conclusion 67
5.3 Future Work 68

REFERENCES 72

APPENDIX A. ER TECHNIQUE EFFECTIVENESS 76

LIST OF FIGURES

FIGURE Page

1.1 This phylogenetic tree, created by David Hillis, Derreck Zwickil and Robin Gutell, University of Texas, depicts the evolutionary relationships of about 3,000 species throughout the Tree of Life. Less than 1 percent of known species are depicted [1]. 2

1.2 Sequence data on three different genes, Gen1, Gene2, and Gene3, on 8 species $\{a, b, c, d, e, f, g, h\}$.... 3

1.3 Top table is the supermatrix of the sequence data in Figure 1.2. Bottom tree is a (super)tree constructed using one of phylogenetic construction methods from supermatrix above.... 4

1.4 Supertree inference. We first build a phylogeny for each gene data in Figure 1.2, trees in (*a*), (*b*), and (*c*). Then, we use supertree inference to construct the supertree depicted in (*d*.) 5

1.5 A rooted phylogenetic tree on taxa set $\{a, b, c, d, e\}$ 11

1.6 Rooting an unrooted tree (on the left): The middle tree is the rooting using midpoint method where the distance between a and c is maximum possible pairwise distance, and the half-way lies in between the two internal nodes. The right tree shows rooting of the same tree using c as outgroup.... 12

1.7 Newick format visualization. Tree above representation in newick format is $(c : 0.6, (d : 0.4, (a : 0.1, b : 0.1) : 0.5) : 0.2) : 0.0;$.... 13

1.8 The Unrooted tree $(a, b, (c, d))$; is showed on the left. The same tree can be drawn in a way that gives the illusion of being rooted, the middle tree. But note that the "root" has three children, and hence the tree is unrooted (equivalent to the tree on the left). The tree on the right, however, corresponds to $((a, b), (c, d))$; which is actually a rooted tree. Even without that little line on the leftmost internal node, we can recognize that it is rooted tree since the "root" has only 2 children. 14

2.1 Baum-Ragan MR coding of phylogenetic trees.... 16

2.2 SuperFine first stage. SCM of two trees S_1 and S_2. In S_1' and S_2', the strict consensus of S_1 and S_2 restricted to their common taxon set is shown in bold. In S_1'' and S_2'', the branches that are involved in collapsing of a path in S_1' and S_2' are shown in bold, respectively. T is the SCM tree of S_1 and S_2. 17

2.3 SuperFine second stage. a) The deletion of the polytomy u from the tree T' partitions T' into four rooted trees, T_1, T_2, T_3, and T_4. b) The leaves in each of the four source trees are relabeled by the index of the tree T_i containing that leaf, producing relabeled source trees S_1^r, S_2^r, S_3^r, and S_4^r. c) Each S_i^r is further processed by repeatedly removing sibling nodes with the same label, until no two siblings have the same label; this results in trees S_i^c. d) The MR encoding, note S_3^c does not contribute a parsimony informative site and is excluded. e) The result of the MRP analysis on the matrix given in (d). f) The tree resulting from identifying the root of each T_i, where i = 1,2,3,4, with the node i in the tree from (e). 18

2.4 MinCut algorithm. Given rooted trees T_1 and T_2, we construct weighted graph S_T. Since it is connected, we construct S_T' by removing all edges of size 2. All three edges in S_T' belong to some minimum cut set of S_T'. By deleting the corresponding edges from S_T, we get four components $\{e\}, \{d\}, \{c\}, \{a, b\}$. Calling $MinCut(T)$ on each component and connecting the roots to a new root results in the MinCut supertree of T_1 and T_2 on the right. 21

2.5 F is called a forest for T if (1) each component of F is a *refinement* of T restricted to the taxon set of that component (A tree T' is a refinement of a tree T if T' can be obtained from T by contracting edges), (2) The subtrees obtaining from restriction of T to each component of F are edge disjoint, and(3) the components of F include all taxa in T. The Agreement Forest of two rooted phylogenetic trees is a forest of both. 24

3.1 Rooted RF distance. T_2 and T_3 represent two possible rooting of the same unrooted tree T_1. The set of non-trivial clusters of T_2 is $\{bcd, cd\}$, and the set of non-trivial clusters of T_3 is $\{dab, ab\}$. Thus the rooted RF distance between T_2 and T_3 is 2. 26

3.2 rSPR operation. 29

3.3 An instance of the input for RFS algorithm: S and T, and $v \in T$ to be pruned. Tree R is obtained from T by pruning v and regrafting it back to the root of T. The node Q represents the clade consisting of all possible regraft places for v. 32

3.4 For each of the non-trivial clades u_i in S, where $1 \leq i \leq 5$, the node a_i represents its LCA in R. 33

3.5 Tree S' is obtained from S by suppressing all nodes m whose LCA is in R_v, i.e. clades $\{f, g\}$ and $\{e\}$. Clades u_3 and u_4 are actually clade $\{c\}$ in S'. Hence, their LCA is clade $\{c\}$ in R which are denoted by b_3 and b_4, respectively. 34

3.6 Three source trees with randomly re-weighted 50% of edges (clusters) to 0. 37

3.7 Three source trees on the taxa set $S = \{a, b, c, d, e, f, g\}$. 40

3.8 A supertree, T, on the taxa set $S = \{a, b, c, d, e, f, g\}$. 40

3.9 SPR neighborhood. There are 7 edges on which we can regraft (f, g) back to the tree, excluding the one it was attached to previously. Here we have three of those neighbors which are the result of regrafting (f, g) to the edge above the clades $\{b, c\}$, $\{d, e\}$, or $\{a\}$, respectively. 41

3.10 In order to calculate RF score between supertree *T* and source tree *S*, we first need to remove non-shared taxa from *T* (c). Then, we suppress all non-labeled leaves and degree 2 internal edges which results in the tree in (d). The RF distance between *T* and *S* is defined as the RF distance between this restricted *T* and *S*. 44

3.11 Parsimony score calculation. Given matrix representation of part *a* and the supertree in part *b*, we calculate the parsimony score of the supertree for each site (column) of the matrix (such as u_2). In above example, the parsimony score corresponding to the column 2 of the matrix is 1. The final parsimony score of the supertre is defined as the sum of the parsimony scores over all columns. 45

3.12 RF score comparison of MRP, SuperFine-MRP, RFS, and three versions of ER algorithm: MRP+ER, SuperFine-MRP+ER, and RFS+ER. 48

3.13 Parsimony score comparison of MRP, SuperFine-MRP, RFS, and three versions of ER algorithm: MRP+ER, SuperFine-MRP+ER, and RFS+ER. 49

3.14 Running time comparison of MRP, SuperFine-MRP, RFS, and three versions of ER algorithm: MRP+ER, SuperFine-MRP+ER, and RFS+ER. 50

4.1 Acceptance Probability function for when T_{i+1} has higher RF score than T_i, i.e. $s_{i+1} > s_i$. 58

4.2 RF score comparison of SA-RB and SA-RR on four datasets given the same initial supertrees, MRP, SuperFine-MRP, and RFS. 60

5.1 RF score comparison of RFS, SuperFine-MRP+ER, RFS+ER, RFS+SA-RR, and RFS+SA-RB. 65

5.2 Parsimony score comparison of MRP, SuperFine-MRP, RFS, SuperFine-MRP+ER, RFS+ER, RFS+SA-RR, and RFS+SA-RB. 65

5.3 Running time comparison of MRP, SuperFine-MRP, RFS, SuperFine-MRP+ER, RFS+SA-RR, and RFS+SA-RB. 65

A.1 RF score comparison of RFS+RFS and RFS+ER. 77

A.2 RF score comparison of MRP+RFS and MRP+ER. 78

A.3 RF score comparison of SuperFine-MRP+RFS and SuperFine-MRP+ER. . . 78

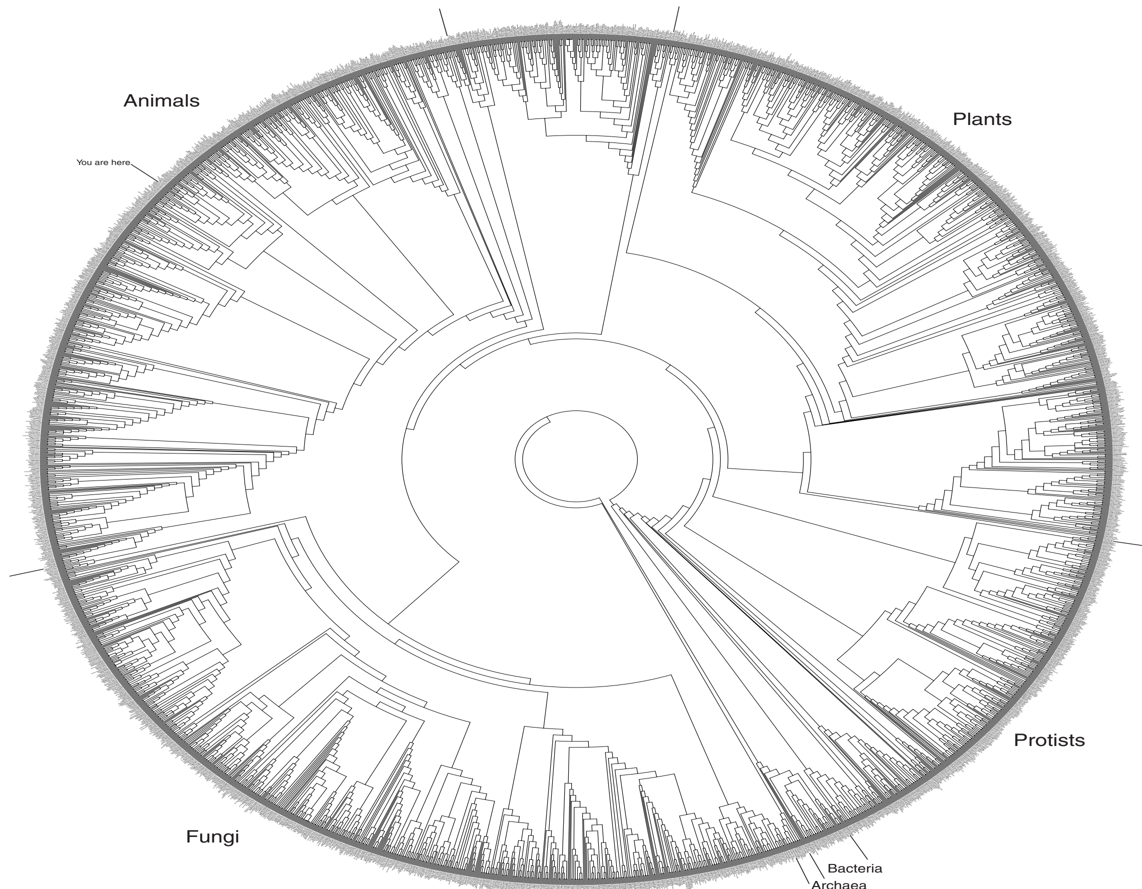
1. INTRODUCTION AND BACKGROUND INFORMATION

1.1 Supertree of Life

Charles Darwin changed the way we look at the life on Earth forever. Darwin's theory of *evolution* encompasses two main ideas. First, evolution occurs, in other words, organisms change over the time. Second, evolution occurs by *natural selection* which roughly says that only the fittest ones will survive over a long period of time. This amazing theory suddenly connected all the dots to explain the extraordinary diversity of life on Earth, and how it has been developed. The theory of evolution explains how all the biodiversity on earth has developed from a common ancestor, and thus all species on earth are related to one another.

Tree of Life (ToL) depicts shared ancestry and the evolutionary relationships of all biodiversity on Earth. Around 1.8 million species have been identified, and biologists have predicted the total number of species on Earth to be 8.7 million [2]. Figure 1.1 shows how the ToL will look like for a very small subset of known species. Such a comprehensive tree can provide significantly useful knowledge to address many critical contemporary issues such as fighting diseases, improving global agriculture, and protecting ecosystems to name a few. However, building such a tree is among the most complicated and challenging scientific problems.

In the literature, there are two different approaches to construct ToL. The first approach, which is called *combined analysis* or *total evidence* or *supermatrix analysis* tries to put together the whole data available, and build a tree at once from all data together. For example, sequence data from multiple *loci* -the specific location or position of a gene's DNA sequence, on a chromosome- is concatenated and is considered as one super-gene. Then the whole data is analyzed by a phylogenetic construction method to build a com-



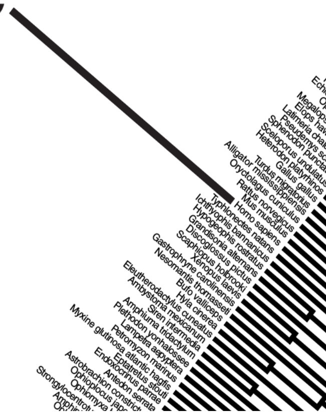


Figure 1.1: This phylogenetic tree, created by David Hillis, Derreck Zwickil and Robin Gutell, University of Texas, depicts the evolutionary relationships of about 3,000 species throughout the Tree of Life. Less than 1 percent of known species are depicted [1].

prehensive tree on all the species under the study. For example, given the sequence data in Figure 1.2, the supermatrix approach, using some phylogenetic construction method, is depicted in Figure 1.3.

However, there are some serious issues using this approach. First, DNA or protein sequence data is available only for a small fraction of species. Second, even if we have molecular data for all species, with existing computational approaches, it is not feasible to construct ToL directly from molecular data for large number of species at once. Third, the available sequence data for different species corresponds to different genes. When we combine them together into one huge sequence-species matrix, the resulting matrix will have huge portion of *missing data* which can potentially decrease the accuracy. On the other hand, biologists have produced tens of thousands of phylogenies for small group of species from different datasets which provide a valuable knowledge to construct ToL. We would like to incorporate all these data in ToL construction.

If we cannot construct ToL at once, it is intuitive to think of a divide and conquer approach as a second approach for ToL construction. In this approach, we build smaller phylogenies with partially overlapping taxa, and then we somehow combine these phylogenies into a more comprehensive tree. The problem of finding such (super)tree is referred to as *Supertree problem* in the literature, and the method by which the tree is constructed

Species	Gene1
a	TCTAAT
b	TCTAAG
c	TCTAGA
d	TCTAAC
g	CATTCA
h	CATACC

Species	Gene2
d	GGTAAC
e	GCTACT
f	GCTAAA
g	GCTAAC

Species	Gene3
a	TATTGA
c	TATTAC
d	TAGTAC
g	TAGTGA
h	TAGTGC

Figure 1.2: Sequence data on three different genes, Gen1, Gene2, and Gene3, on 8 species $\{a, b, c, d, e, f, g, h\}$.

Species	Gene1	Gene2	Gene3
a	TCTAAT	??????	TATTGA
b	TCTAAG	??????	??????
c	TCTAGA	??????	TATTAC
d	TCTAAC	GGTAAC	TAGTAC
e	??????	GCTACT	??????
f	??????	GCTAAA	??????
g	CATTCA	GCTAAC	TAGTGA
h	CATACC	??????	TAGTGC

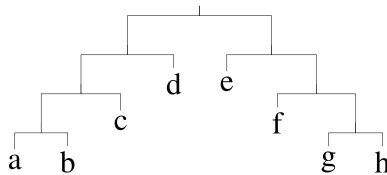
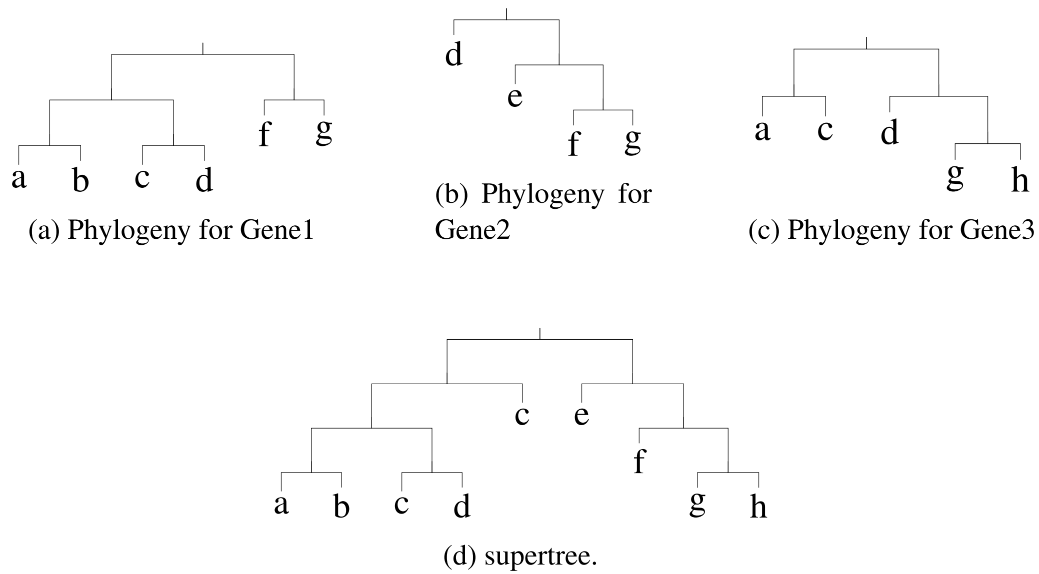


Figure 1.3: Top table is the supermatrix of the sequence data in Figure 1.2. Bottom tree is a (super)tree constructed using one of phylogenetic construction methods from supermatrix above.

is called *supertree inference* or *supertree algorithm*. For instance, given sequence data in Figure 1.2, we can first build gene trees using some phylogenetic tree construction method for each gene data, and then we can use some supertree inference method to build a supertree. Figure 1.4 represents this approach for gene data in Figure 1.2. This approach has several advantages. First of all, since it is a divide and conquer approach, we can divide the problem into sufficiently small sub-problems that we are able to solve. Second, unlike supermatrix method, the problem of missing data is less harmful. Further, existing phylogenies can also be used for ToL construction which provide a great resource.

However, this is not a trivial task. There are several non-trivial questions involved in designing such approach. First of all, how can we handle the conflicts among the given phylogenies? Second, how can we possibly combine a set of trees into a single tree which, in some sense, best represents them? Third, how should we define *similarity* between two



(a) Phylogeny for Gene1

(b) Phylogeny for Gene2

(c) Phylogeny for Gene3

(d) supertree.

Figure 1.4: Supertree inference. We first build a phylogeny for each gene data in Figure 1.2, trees in (*a*), (*b*), and (*c*). Then, we use supertree inference to construct the supertree depicted in (*d*.)

phylogenies, and how can we quantify this measure? Last but not the least, how can we use such measure to build a single tree, called *supertree*, out of the given source trees?

A supertree on a set of input phylogenetic trees is simply a phylogenetic tree which contains all the taxa in all the input trees and, in some sense, preserves as much information as possible from input trees. A supertree algorithm should answer all the above mentioned questions. Supertree algorithms have been considered as the main tool to construct ToL in the literature, and supertree inference has been an active research area in the last two decades.

Another complication in supertree inference is that the true species tree is not available for biological datasets. Therefore, the only way to evaluate the accuracy of a supertree is to measure its similarity to the input source trees. In other words, after constructing a supertree, we evaluate its accuracy by measuring its similarity to input trees. This similarity is usually captured by some phylogenetic tree distance measure. There are several

different such distance measures in the literature. Among these distance measures, *Robinson Foulds* distance (RF distance) and *Parsimony Score* have been widely used for both constructing supertrees and evaluating their accuracies.

1.2 Our Research

Several criteria have been proposed for supertree construction. Surprisingly, the supertree problem turns out to be NP-hard for most of the interesting measures introduced. This means that we are not optimistic about finding an efficient solution for these problems. Thus, we usually have to resort to heuristic search algorithms. It is interesting to know the size of the solution space for this problem, i.e. the number of all possible supertrees on a given set of taxa. For n taxa, there are $\frac{(2n-5)!}{(n-3)\,2^{n-3}}$ number of unrooted trees, and $\frac{(2n-3)!}{((n-2)\,2^{n-2})}$ number of rooted trees. For example, the number of rooted and unrooted trees with only 10 taxa, is 34 459 425 and 2 027 025, respectively. This observation eliminates any hope for exhaustive search in the solution space.

There are many different approaches for supertree inference. However most of supertree algorithms can be categorized into one of three main groups: matrix representation (MR) methods, graph based methods with polynomial running time, and topological distance based methods. In this work we will focus on the third category, topological distance based methods. These algorithms are of special interest because they directly aim at minimizing the distance between the supertree and the source trees. In simple words, given k source trees $T_1, T_2, ..., T_k$, and a well-defined tree distance measure d, the supertree problem is defined as follow: find a supertree with minimum cumulative distance $\sum_{n=1}^{k} d_i$, where d_i is the distance between the supertree and the source tree T_i.

More specifically, we focus on *RF supertree problem*. In RF supertree problem, we are given a set of rooted input source trees, and our goal is to find a *binary* supertree with minimum cumulative RF distance to source trees. RF distance captures the smallest unit

of information in phylogenetic trees. Further, this measure has extensively been used for supertree evaluation. Lastly, the RF supertree problem has been shown to be NP hard [3] which opens a lot of space for improvements of the heuristic algorithms.

The best existing algorithm for RF supertree problem was introduced in [4]. Bansal et al. proposed a fast hill climbing heuristic, RFS algorithm, for RF supertree problem. Although they have shown that their algorithm is able to obtain very accurate supertrees with regard to RF distance, we believe that there is still some space to improve upon their algorithm.

One common problem to all hill-climbing algorithms, including RFS algorithm, is that they only guarantee to obtain a local optimum. This local optimum might correspond to a very poor quality solution for the problem at hand. This problem is known as *local optimum problem* which means that a hill climbing algorithm will get stuck at the first local optimum solution which might be a very poor solution. However, we might be able to find much better quality solutions if we could somehow manage to escape from the local optimum and starting to "climb" another "hill" in the solution space.

Our goal in this work is to propose new methods to improve accuracy of the supertree with regard to RF distance. We have two main contributions in this work. Firs, we propose a new technique, called *Edge Ratchet*, to deal with local optimum problem in hill-climbing algorithms such as RFS. Second, we design a new *Simulated Annealing* algorithm for RF supertree problem.

1.2.1 Edge Ratchet Robinson Foulds Supertree Algorithm

We propose a new algorithm, Edge Ratchet Robinson Foulds Supertree (ER-RFS) Algorithm, which is essentially a hill climbing algorithm equipped with *Edge Ratchet* technique to deal with local optimum problem. This work is inspired by the ratchet search strategy introduced in [5]. Ratchet search is an iterative search method that has been

shown to be quite effective to address local optimum problem for branch-swapping hill-climbing algorithms for parsimony problem. In each iteration of ratchet search two branch swapping searches is applied. In the first phase, a randomly selected subset of characters, columns of parsimony matrix, are given a different weight and branch swapping is performed on this re-weighted data set to obtain a local optimum. Then in the second phase, all the weights are set back to the original weights, and branch swapping is applied on the original data set using the tree obtained in the first phase as initial tree. The output of the second phase of each ratchet iteration is used as initial tree of the next ratchet iteration. This is done for a prespecified number of iterations.

Bansal et al. introduced fast local search algorithms for Robinson-Foulds supertree problem and developed a powerful hill-climbing algorithm for this problem in [4]. They used a ratchet search heuristic based on parsimony ratchet [5] to deal with the local optimum problem. They treat each *source tree* as one character in the ratchet search. More specifically, the re-weighing of characters in the first phase of each ratchet iteration is done by removing two third of source trees. This means that ALL bipartitions of the chosen source trees are ignored in the first phase of each ratchet iteration. Although they have shown their algorithm is capable of obtaining supertrees with high quality with regard to RF distance, we believe that, by using our edge ratchet strategy instead, we can improve upon their algorithm.

In the edge ratchet strategy, we treat each internal edge (essentially a bi-partition) of each source tree as one character instead of the whole source tree. Then in the first phase of each ratchet iteration, we re-weight, say, 30% of internal edges of each source tree with a new weight and perform a hill-climbing search using weighted source trees. Then, we set back all the edge weights to 1, and the resulting supertree from previous search is fed as initial supertree for the second phase that performs a hill-climbing search using (unweighted) original source trees.

An effective strategy to deal with local optimum problem is extremely important for specifically branch swapping methods. Because of the nature of these search algorithms, they could potentially get stuck in areas of solution space, called *tree islands* [5]. A tree island is defined as a set of trees in the solution space having the same score (based on whatever optimality criteria used), and this score is better than that of any other tree reachable by a single branch swapping move. When the algorithm reaches a tree island, it alters between numerous trees with the same quality that differ by minor rearrangements, for a long time, without any improvement in the solution. This can potentially have an adverse effect on the final solution. Thus, a good strategy that can help the algorithm escape from these parts of the solution space can potentially have a huge impact on the resulting solution since the algorithm can search different areas of the solution space which increases the chance of finding better solutions.

1.2.2 Simulated Annealing Algorithm

Simulated annealing (SA) algorithm is inspired by annealing process in metallurgy, a technique involving heating and controlled cooling of a material to increase the size of its crystals and reduce their defects. There are three main reasons for using simulated annealing for RF supertree problem. First, simulated annealing algorithm is known to be an effective way to handle local optimum problem. Although our experiments show that ER-RFS algorithm is able to handle local optimum and improve upon RFS algorithm, we do not know if there are better local optimums in the solution space. The way SA algorithm works could help to even improve the accuracy of the supertree.

Second, we observed in our experiments that the initial supertree can have a huge impact on the final supertree of ER-RFS algorithm, especially on data sets with small number of source trees. The way SA algorithm works could potentially handle this problem very well. Third, ER-RFS algorithm is very expensive with regard to running time, and it is

a fairly complex algorithm. SA algorithm, however, is simpler, and more importantly, it does not require to search the whole SPR neighborhood. Therefore, if it could handle local optimum problem well in this context, then we might expect that it could obtain close (or even better) results to that of ER-RFS with better running time.

Therefore, as another heuristic, we implement a simulated annealing algorithm for RF supertree problem. In greedy hill-climbing algorithms we always pick a better neighbor in local search, and this will lead us to a (potentially not enough good) local optimum. Simulated Annealing algorithm, deal with this problem by allowing worse neighbors to be picked as well. But, the proper way to handle this is not trivial. If we pick too many worse neighbors, we might get stuck in some areas of the solution space with poor quality solutions, and end up in a supertree that is even worse than the initial supertree. If we are too strict, on the other hand, and we rarely pick a worse neighbor, then the final solution is probably close to that of greedy algorithm. The beauty of SA algorithm is in the way it allows worse neighbors to be selected so that neither of the above cases happen.

1.3 Background Information and Terminology

In this section we present some basic terms and concepts commonly used in phylogenetics. We first define a phylogenetic tree and the relevant terms. Then, we demonstrate the difference between rooted and unrooted phylogenies. Finally, we describe the newick format which is the common way of representing a phylogenetic tree.

1.3.1 Phylogenetic Tree

In evolutionary context, the relatedness of two different species should closely depend on how recent they have had a common ancestor. Such evolutionary relationship can be perfectly modeled as a tree, *phylogenetic tree*, which is very much like a family tree. A phylogenetic tree is a tree structure that depicts the evolutionary relationships among a set of species. Each leaf of the tree represents one of the existing species, called *taxon*, and

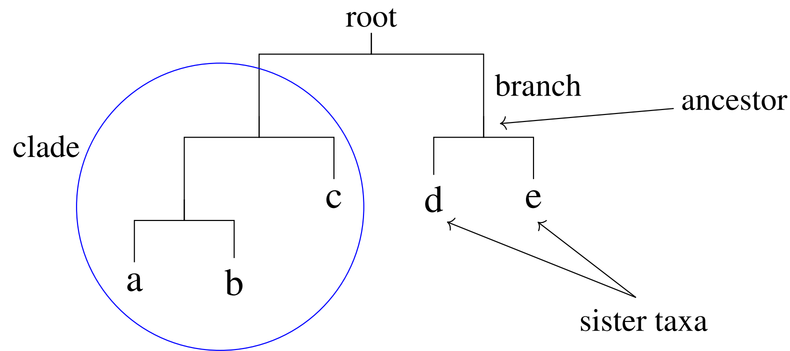


Figure 1.5: A rooted phylogenetic tree on taxa set $\{a, b, c, d, e\}$
.

the internal *nodes* represent the (hypothetical) ancestral species. For example, in Figure 1.5, we can say species *a* and *b* are more related to each other than than they are to other taxa in the tree because their common ancestor is more recent than, say, common ancestor of *a* and *c*. Sometimes, we specify a node in the tree as *root* which represents the common ancestor of all taxa on the tree, and determines the flow of time. Each edge of the tree is called *branch*. Each internal node with all its descendants represent a *clade* (or *cluster*). Two species with the same parent node in a rooted tree are called *sister taxa*. Figure 1.5 illustrates all these terms.

The number of branches connected to a node is called *degree* of that node. Leaves have degree of 1. Note that all the internal nodes in a phylogeny has degree grater than 2. An internal node (in both rooted and unrooted trees) is called *multifurcation* or *polytomy* if its degree is greater than 3. For rooted trees, the root, which is a unique internal node, is called multifurcation if it has more than two children. We say a node in phylogenetic tree is *fully resolved* if it is not a multifurcation. A phylogenetic tree with no multifurcations is called a *binary tree* or *bifurcating tree* or *fully resolved tree*. For example, the phylogeny in Figure 1.5 is fully resolved. Thus, for example, a rooted binary tree is a rooted tree whose root has degree 2 and all other internal nodes have degree 3. Likewise, an unrooted

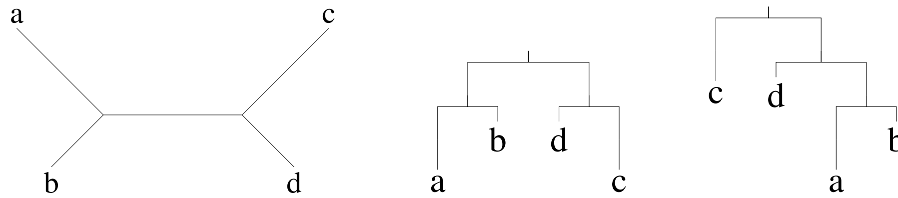


Figure 1.6: Rooting an unrooted tree (on the left): The middle tree is the rooting using midpoint method where the distance between a and c is maximum possible pairwise distance, and the half-way lies in between the two internal nodes. The right tree shows rooting of the same tree using c as outgroup.

binary tree is an unrooted tree whose internal nodes have all degree 3.

1.3.2 Rooted vs Unrooted Trees

A phylogeny can be either *rooted* or *unrooted*. The main difference between rooted and unrooted trees is that rooted trees contain the information about the flow of time, while we do not know which way of evolution proceeded along each edge of an unrooted tree. In rooted trees, there is a unique internal node called *root* which represents the common ancestor of the species in the tree. In rooted trees, each edge represents a parent-child relationship.

There are two common ways of converting an unrooted tree to rooted tree. First approach is called *midpoint rooting*. As the name suggests, this approach roots the tree at its midpoint. This is done by finding the longest taxon to taxon path in the tree, and placing the root at exactly half way between the two taxa. In midpoint rooting we calculate *patristic distance* between each pair of taxa which is the sum of the length of all branches on the path between the two taxa. Each branch of a phylogeny can be assigned a number called *branch length* which represents the amount of genetic change on that branch. The branch length unit is usually nucleotide substitutions per site, i.e. the number of changes or substitutions divided by the length of the sequence. The second approach for rooting a tree, which is more recommended, is to use an *outgroup* to root the tree. An outgroup is a

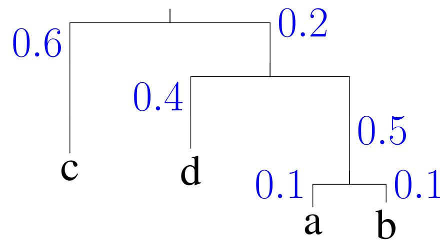


Figure 1.7: Newick format visualization. Tree above representation in newick format is $(c : 0.6, (d : 0.4, (a : 0.1, b : 0.1) : 0.5) : 0.2) : 0.0;$.

species that is known to be more distantly related than everything else in the tree. Figure 1.6 depicts rooting a small unrooted tree using these two approaches.

1.3.3 Newick Format

There is a computer readable and intuitive way of representing a phylogenetic tree called *Newick* format introduced in 1857 by Arthur Cayley. The newick standard takes advantage of correspondence between a tree and nested parentheses. In this representation, a pair of parentheses is used to group sister taxa into one clade. Newick format always starts with an open parenthesis, and ends with a semicolon which marks the end of the tree. Branch lengths, if any, are prefixed by colons following right after the corresponding node in the tree. Internal nodes may or may not be named. If named, the name of each clade comes right after the closing parenthesis corresponding to that clade. (see Figure1.7).

In newick format, the convention is to have an unrooted binary tree with only one multifurcation at some internal node. It is a little tricky to differentiate between rooted and unrooted trees in newick format. The convention, which is used in many software programs, is that if the outermost pair of parentheses in newick representation includes only 2 clades, then it is considered rooted and the root is placed between those two clades. Otherwise, the tree is unrooted. Most of phylogenetic software programs whose output is an unrooted tree place a trifurcation at node corresponding to outermost parentheses to

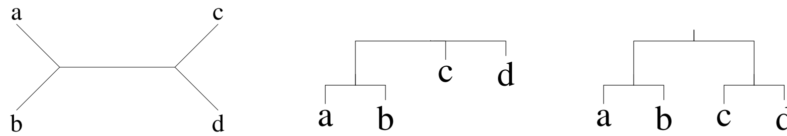


Figure 1.8: The Unrooted tree $(a, b, (c, d))$; is showed on the left. The same tree can be drawn in a way that gives the illusion of being rooted, the middle tree. But note that the "root" has three children, and hence the tree is unrooted (equivalent to the tree on the left). The tree on the right, however, corresponds to $((a, b), (c, d))$; which is actually a rooted tree. Even without that little line on the leftmost internal node, we can recognize that it is rooted tree since the "root" has only 2 children.

emphasize that the tree is unrooted.

We should note that for unrooted trees, some of the tree visualization software programs draw the tree in a way that seems it is rooted at node corresponding to the outermost parentheses, which looks like they draw rooted trees. For example, Figure1.8 depicts the difference between rooted and unrooted trees in newick format. When looking at unrooted trees in visualization programs, it is a good practice to choose the option, if available, that draws the tree like the one on the left side of Figure1.8. Another probable source of confusion in newick format is that a tree topology may have different newick representations. For example, all the following newick representations are equivalent, and represent the left tree in Figure1.8.

(a,b,(c,d));

(b,a,(c,d));

((c,d),b,a);

(b,(d,c),a);

((a,b),c,d);

(c,(a,b),d);

2. LITERATURE REVIEW

There are many different approaches for supertree inference. However most of supertree algorithms can be categorized into one of three main groups: matrix representation (MR) methods, graph based methods with polynomial running time, and topological distance based methods. In what follows we introduce the main supertree methods in each category, and explain them in details.

2.1 MR Based Supertrees

The first MR method,*Matrix Representation with Parsimony* (MRP) , was introduced by Baum [6] and Ragan [7] independently in 1992. MRP has been the most widely used method in the literature because of its superiority in both accuracy and running time. All MR methods have generally two main steps. The first step is to encode the source trees into a large matrix of entries {0,1,?}. Each row of the matrix corresponds to one species, and each column of the matrix corresponds to one internal node of some source tree. The MR entry corresponding to taxon t and internal node n in tree T is 1 if t is a descendant of n, the entry is 0 if t is not a descendant of n but is present in tree T, and finally, the entry is ? if the taxon t is not present in tree T (this is called "Baum-Ragan" encoding). Figure 2.1 represents this encoding for two trees with partially overlapping taxon set. Note that this encoding can be used for unrooted trees in a similar way, except for each internal *edge* we need arbitrarily assign 0 to one side, and 1 to the other side. This works because in parsimony, we are interested in the number of *changes* of state in each site (column).

This clever way to represent a tree in MR form, provides the opportunity to treat the matrix as sequence data. In other words, we can construct a tree from this matrix using any phylogeny construction method from sequence data. In the second step of MR based supertree algorithms, different methods use different optimality criteria to build a supertree

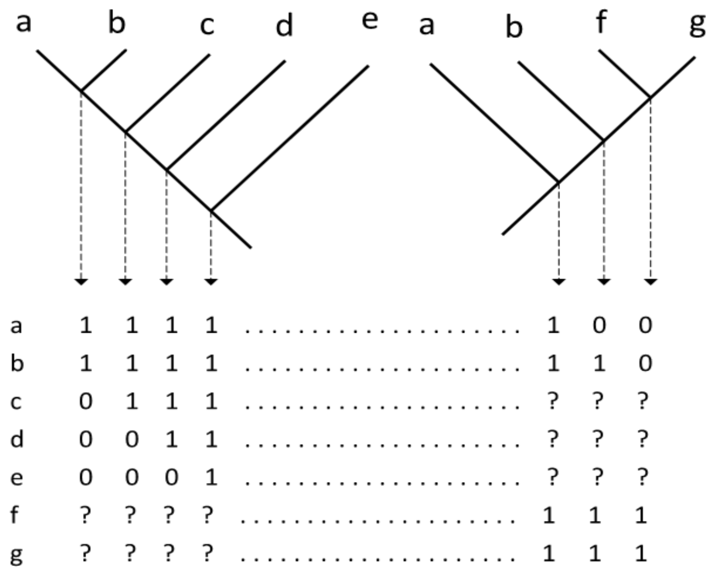


Figure 2.1: Baum-Ragan MR coding of phylogenetic trees.

from the large matrix in the first step. For example, MRP uses parsimony criteria to build a tree from this matrix representation. Parsimony principle says that the simplest explanation is the most likely to be correct. In other words, parsimony suggests that the best tree is the tree that minimizes the number of evolutionary steps (i.e., changes among characters). The *parsimony score* for a supertree refers to the sum of the smallest number of substitutions needed for each site (i.e. each column of character-taxa matrix). The tree with the lowest parsimony score is the most parsimonious tree. Thus, MRP basically seeks a supertree that has minimum parsimony score. Although finding the most parsimonious tree is NP-complete [8], very effective heuristics have been implemented for this problem in PAUP [9] and TNT [10].

Recently, Swenson et al. proposed a new algorithm called *SuperFine* which was an attempt to make MRP even faster and more accurate [11]. The input source trees for this algorithm can be unrooted and non-binary. Their algorithm has two main phases.

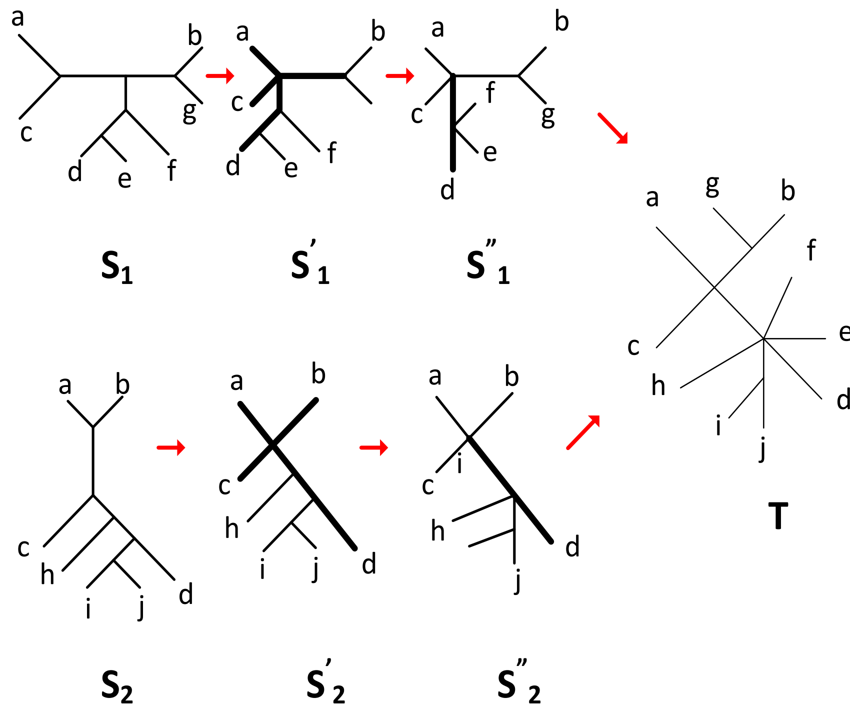


Figure 2.2: SuperFine first stage. SCM of two trees S_1 and S_2. In S_1' and S_2', the strict consensus of S_1 and S_2 restricted to their common taxon set is shown in bold. In S_1'' and S_2'', the branches that are involved in collapsing of a path in S_1' and S_2' are shown in bold, respectively. T is the SCM tree of S_1 and S_2.

First, they build a partially resolved unrooted tree that only contains edges on which all source trees agree, called *Strict Consensus Merger* (SCM). An example of SCM is shown in Figure 2.2. In the second phase, they resolve the polytomies in SCM. To resolve a polytomy u of degree d, they first divide the SCM into several partitions by removing u. Then each partition is give a label i, and all the taxa in source trees belonging to partition with label i are relabeled by i. This results in a set of new source trees each with at most d taxa. Then, repeated taxa in each source tree are removed by removing sibling nodes with same label until all leaves are unique. Finally, MRP heuristic is run on this new set of modified source trees to get a supertree on reduced taxon set. The resulting tree can

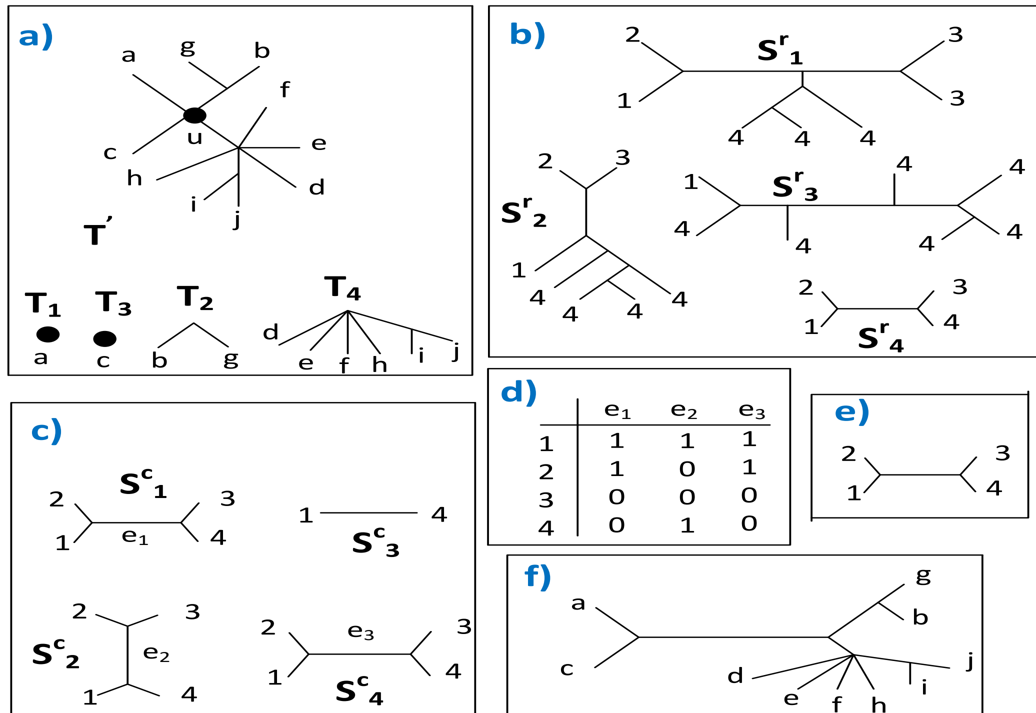


Figure 2.3: SuperFine second stage. a) The deletion of the polytomy u from the tree T' partitions T' into four rooted trees, T_1, T_2, T_3, and T_4. b) The leaves in each of the four source trees are relabeled by the index of the tree T_i containing that leaf, producing relabeled source trees S_1^r, S_2^r, S_3^r, and S_4^r. c) Each S_i^r is further processed by repeatedly removing sibling nodes with the same label, until no two siblings have the same label; this results in trees S_i^c. d) The MR encoding, note S_3^c does not contribute a parsimony informative site and is excluded. e) The result of the MRP analysis on the matrix given in (d). f) The tree resulting from identifying the root of each T_i, where i = 1,2,3,4, with the node i in the tree from (e).

easily be used to resolve polytomy u. This procedure is depicted in Figure 2.3. The reason why their algorithm is faster than MRP is that instead of running one MRP on the whole data set, they run several MRP on smaller size inputs. Further, the second phase can be parallelized to make it even faster [12].

Another well known MR based sueprtree is called Matrix Representation with Flipping (MRF) [13]. Like MRP, MRF uses matrix representation of rooted source trees. However,

MRF takes a different approach to construct a supertree. MRF tries to minimize *flip distance*. Flip distance is defined as the number of *flips*, i.e. flipping a 0 to 1 or vice versa in MR, required to convert the matrix into one that perfectly represents a phylogenetic tree. Before explaining the intuition behind this optimization criteria, we should note that there is almost always incompatibilities between source trees, and this is usually because of the existence of error. MRF is actually motivated by the notion of *error correction*. On one hand, the error in source trees can be because of presence of an incorrect taxa in a clade (which is a 1 instead of 0), or the absence of one that should be present (which is a 0 instead of 1). On the other hand, a flip moves a taxon into or out of a clade- perfect match!. Thus, the idea behind this optimization problem is that the *error* in MR prevents it from perfectly representing a phylogenetic tree, and therefore, the attempt to minimize the number of flips to resolve errors is intuitively reasonable. Chen et al. in [14] proved that MRF problem is NP-hard, even when all the source trees have the same set of taxa.

For the special case of MRF consensus tree, where all source trees have the same set of taxa, Böcker et al. presented a $\mathcal{O}(4.83k + poly(m, n))$ fixed-parameter algorithms, for n taxa, m characters, k flips, and $poly(m, n)$ denotes a polynomial function in m and n. Chen et al. in [14] also provided an approximation algorithm with ratio d, where d is the maximum number of ones in a column, for MRF consensus tree. However, for the general case, there is no any approximation algorithms or parameterized algorithms. Chen et al. presented a heuristic for MRF problem based on branch swapping in [15]. Their algorithm is a hill climbing algorithm. The initial tree is obtained through *greedy step wise taxon addition* using a randomly-chosen order. Then they use one of the tree rearrangement operations rNNI, rSPR, or rTBR to produce neighborhood of the current tree. If no neighbor has a lower flip distance, the search stops and the current tree is returned as the estimate of an MRF supertree. Otherwise, the current tree is replaced by its best neighbor. Thus, the algorithm stops at the first seen local optimum.

The MR based supertree algorithms are different from other supertree algorithms in that they use an *indirect* method for supertree construction. In other words, the MR methods *indirectly* solve the supertree problem by encoding the input source trees into matrix representation. Then these methods treat this matrix representation as sequence data and use some phylogeny construction methods to build a supertree.

2.2 Graph Based Supertrees

The graph based supertree algorithms take advantage of graphs to capture topological information of source trees. The main differences between graph based supertree algorithms and other suprtree methods is that they are not usually based on optimizing a global objective function and they often use a local optimization criteria, and they are solvable in polynomial time. The first graph based supertree algorithm, *BUILD*, was developed by Aho et al. in 1981 which was only capable of dealing with non-conflicting source trees [16]. The most well know graph based algorithms are *MinCut* (MC) [17] and its improved version *Modified MinCut* (MMC) [18] which were the first extensions of BUILD to handle conflicting rooted source trees. The conflict in MC is resolved by deleting the minimum amount of information from the input trees in order to allow the algorithm to proceed.

The MC algorithm is implemented by a recursive function $MinCut(T)$ which takes as input a set of rooted source trees. Suppose we are given a set T of k rooted source trees on n taxa. For trivial case of having $n = 1$ or $n = 2$, $MinCut(T)$ returns a single node or a rooted tree with two leaves, respectively. The algorithm first creates a weighted graph S_T, where the nodes are species, and nodes a and b are connected if a and b are in a proper cluster (i.e. any cluster other than root) in at least one of the input trees. The weight of edge between a and b is the number of source trees in which a and b are in a proper cluster. Then, if S_T is disconnected, we recursively call $MinCut(T|S_i)$ for each component S_i, where $T|S_i$ is the set of input trees with any species not in S_i pruned. If S_T is connected,

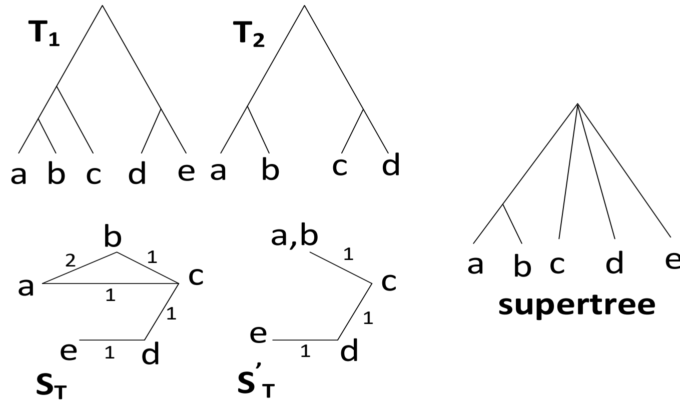


Figure 2.4: MinCut algorithm. Given rooted trees T_1 and T_2, we construct weighted graph S_T. Since it is connected, we construct S'_T by removing all edges of size 2. All three edges in S'_T belong to some minimum cut set of S'_T. By deleting the corresponding edges from S_T, we get four components $\{e\}, \{d\}, \{c\}, \{a, b\}$. Calling $MinCut(T)$ on each component and connecting the roots to a new root results in the MinCut supertree of T_1 and T_2 on the right.

it will be disconnected, and then $MinCut(T|S_i)$ will be called for each component S_i. S_T is disconnected in three steps: First, we contract all the edges in S_T whose weight is n to create S'_T. This guarantees to preserve all the clades appear in all source trees. Then, we find the set of all edges belong to some minimum cut set of S'_T, E' (The minimal cut-set of a weighted graph is defined as a set of edges whose removal makes the graph disconnected, and the sum of the weights of these edges is minimized). Finally, all the corresponding edges to E' in S_T are deleted. in the last step of the algorithm, the supertree is constructed by connecting the the roots of each rooted tree obtained by calling $MinCut(T|S_i)$. The algorithm is depicted in Figure 2.4.

The most expensive part of MinCut algorithm is finding minimum cut sets of S'_T which can be done in polynomial time [19]. Although graph based algorithms have the advantage of having polynomial running time, Brinkmeyer et al. in [20] showed the superiority of MR methods over graph based methods in terms of similarity to source trees (and the

model tree in case of simulated data). However, they also showed that MinCut, MinFlip, and MRP produced more accurate supertrees in compare to other four supertree algorithms they considered, *Build-with-distances, PhySIC, PhySIC IST,* and *super distance matrix*.

2.3 Distance Based Supertrees

In the past few years, several topological distance based supertree algorithms have been proposed. In this family, *Robinson Foulds Supertree* (RFS) [4] and *Supertrees Based on the Subtree Prune-and-Regraft Distance* (SPR supertree) [21] have introduced novel approaches for supertree inference. What these methods have in common is to define an optimization problem based on some tree distance measure, and then to aim to minimize the total distance between the supertree and source trees. Further, both of them propose hill climbing based algorithms.

The RFS algorithm tries to minimize the RF distance between a binary supertree and the rooted source trees. The RF distance captures the number of bipartitions in one tree that do not exist in another. The RF distance metric between two rooted trees is defined to be a normalized count of the symmetric difference between the set of clusters of the two trees (which is equivalent to its definition for unrooted trees). This problem is NP hard [3]. Bansal et al. in [4] introduced fast hill climbing heuristics for RF supertree problem. As mentioned before, the size of SPR and TBR neighborhoods are $\mathcal{O}(n^2)$ and $\mathcal{O}(n^3)$ respectively, where n is the number of taxa. Further, calculating RF distance is possible in $\mathcal{O}(n)$, for example [22]. Thus, the naive algorithms for SPR and TBR local search problems require $\mathcal{O}(kn^3)$ and $\mathcal{O}(kn^4)$ time respectively, where k is the number of source trees.

The main contribution of Bansal et al. is to present fast algorithms to solve local search problems for both SPR and TBR in $\mathcal{O}(kn^2)$, where n is the number of taxa in supertree, which yielded speed-ups of $\mathcal{O}(n)$ and $\mathcal{O}(n^2)$ over existing solutions for these problems,

respectively. First, they proved that given a node v to prune in the tree, the answer to TBR local search can be obtained by optimizing the rooting for the pruned subtree, and optimizing the regraft location separately. And then, they presented $\mathcal{O}(kn)$ algorithms to solve both problems, i.e. given the node to be pruned, finding the best rooting of the subtree, and the best place to regraft it such that RF distance to source trees is minimized. This immediately implies an $\mathcal{O}(kn^2)$ solution for TBR local search.

Sine SPR is a special case of TBR, the algorithm is applicable to SPR search as well. They also used a ratchet search heuristic based on parsimony ratchet [5] to prevent the potential problem of getting caught in local optima. Further, they generated initial supertree by greedy stepwise addition procedure. All these attempts resulted in one of the best existing supertree algorithms. They compared RF supertree against MRP and Triplet supertree[23]. Their empirical results on biological datasets show that RF supertree was able to obtain supertrees with lowest RF distances and competitive parsimony scores.

On the other hand, the SPR distance, d_{SPR}, between two phylogenetic trees is defined as the minimum number of SPR operations required to reconcile two trees. The SPR supertree attempts to minimize the total SPR distance between a binary rooted supertree and rooted source trees in the hope of finding better quality supertrees over RF supertree in presence of *Lateral Gene Transfer* (LGT). The intuition behind this is that presence of substantial LGT can drastically increase RF distance, however, a single SPR operation can accommodate such long-distance transfer. Computing the SPR distance between two phylogenetic trees is NP-hard [24, 25], and thus the optimization problem of SPR supertree is NP-hard as well. However, Whidden et al. in [21] presented practical algorithm to compute SPR distance by taking advantage of two recent advances.

First, Whidden et al. in [26] developed fast FPT algorithms for computing *Maximum Agreement Forest* (MAF) of two trees, and it has been shown that the number of trees in MAF is equivalent to rooted SPR distance [24]. Roughly speaking, an Agreement Forest

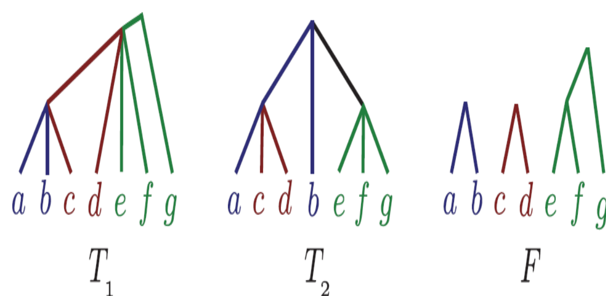


Figure 2.5: F is called a forest for T if (1) each component of F is a *refinement* of T restricted to the taxon set of that component (A tree T' is a refinement of a tree T if T' can be obtained from T by contracting edges), (2) The subtrees obtaining from restriction of T to each component of F are edge disjoint, and(3) the components of F include all taxa in T. The Agreement Forest of two rooted phylogenetic trees is a forest of both.

(AF) is a set of subtrees obtained by cutting edges in a pair of trees until no topological disagreement remains. Figure 2.5 depicts agreement forest of two trees. The MAF of two rooted phylogenies is an agreement forest of them with minimum number of components. The second advance used in SPR supertree is the cluster reduction technique introduced by Linz and Semple [27] for calculating rooted SPR distance. This technique, which uses MAF to calculate SPR distance, allows to divide the source trees into smaller sub-problems that can be solved iteratively.

The SPR supertree algorithm by Whidden et al. [21] is essentially a hill climbing algorithm that uses SPR as edit operation in each iteration to produce the neighborhood of the current solution. The initial supertree is generated by greedy stepwise addition procedure. First, the four most frequent taxa are picked and the the topology with minimum SPR distance to source trees is found. Then the next most frequent taxon is added in the location that minimizes the SPR distance. This continues until all taxa are included. Then for a pre-specified number of iterations, the algorithm produces the SPR neighborhood and picks the the neighbor with lowest SPR distance. To avoid the high cost of exhaustive search in each iteration, they used a bipartition constrain to limit the size of SPR neigh-

borhood. In each iteration, they find all *fixed* bipartitions of the current supertree, and all SPR neighbors that violate any of these bipartitions are disallowed. A bipartition of the supertree is called fixed if it is supported by at least half of the source trees containing two or more taxa from each of the two sets induced by the bipartition. They compared SPR supertree against RF supertree and MRP. Their results show that, on simulated data, SPR supertree outperformed others for *plausible* range of LGT. However, the improvement of SPR supertree was less pronounced on biological data set.

3. EDGE RATCHET ROBINSON FOULDS SUPERTREE ALGORITHM

In this section, we first formally define RF supertree problem. Then, we explain how to develop a hill-climbing heuristic algorithm for this problem. Next, we introduce Edge Ratchet technique to handle local optimum problem in our hill-climbing algorithm. Finally, we incorporate all these building blocks together to design our Edge Ratchet Robinson Foulds (ER-RFS) Supertree algorithm.

3.1 Robinson Foulds Distance

Each internal edge (i.e. an edge whose endpoints are not a leaf) of a phylogenetic tree corresponds to a *bipartition* which refers to partitioning the taxa of the tree into two sets of size greater than 2. The set of bipartitions of a tree includes all possible bipartitions on that tree. The *unrooted RF distance* between two unrooted phylogenies is defined as the number of unique bipartitions in each tree. Note that RF distance is only defined between two trees with the same set of taxa.

On the other hand, rooted RF distance is defined in a similar way for rooted trees. The RF distance metric between two rooted trees is defined to be a normalized count of the

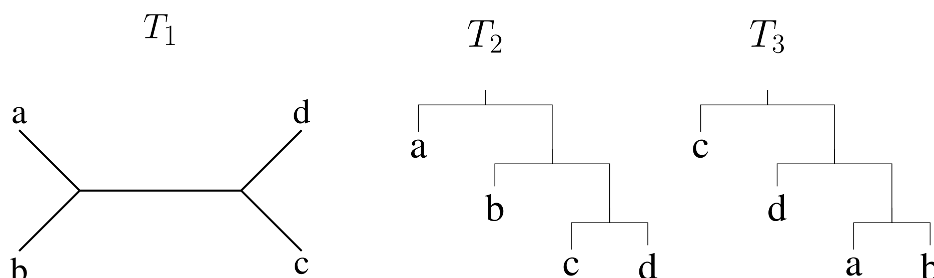


Figure 3.1: Rooted RF distance. T_2 and T_3 represent two possible rooting of the same unrooted tree T_1. The set of non-trivial clusters of T_2 is $\{bcd, cd\}$, and the set of non-trivial clusters of T_3 is $\{dab, ab\}$. Thus the rooted RF distance between T_2 and T_3 is 2.

symmetric difference between the set of *clusters* of the two trees. All descendants of a node in the tree corresponds to a cluster. In this definition, the clades corresponding to the root and the leaves are trivial because the exist in both trees. therefore, they are ignored when calculating the RF distance. Hence, rooted RF distance between T_1 and T_2 with same set of taxa can be formally defined as follow, where $C(T_1)$ and $C(T_2)$ are the set of non-trivial clusters of T_1 and T_2, respectively.

$$d_{RF}(T_1, T_2) = \frac{|C(T_1) - C(T_2)| + |C(T_2) - C(T_1)|}{2}$$

Although rooted and unrooted RF distance are closely related, they could be a little different. For example, Figure 3.1 demonstrates rooted RF distance. Note that the unrooted RF distance between T_2 and T_3 as unrooted trees is 0 since they have only one (non-trivial) bipartition $\{ab \mid cd\}$. However, their rooted RF distance is 2. In the remainder of this work, we refer to rooted RF distance as RF distance for simplicity.

3.2 Robinson Foulds Supertree Problem

Given the formal definition of the RF distance, now we are ready to formally define RF supertree problem as below.

Input: A set of rooted phylogenetic source trees with partially overlapping taxa.

Output: A rooted binary supertree on all the taxa with minimum *RF score*, i.e. cumulative rooted RF distance between the supertree and all source trees.

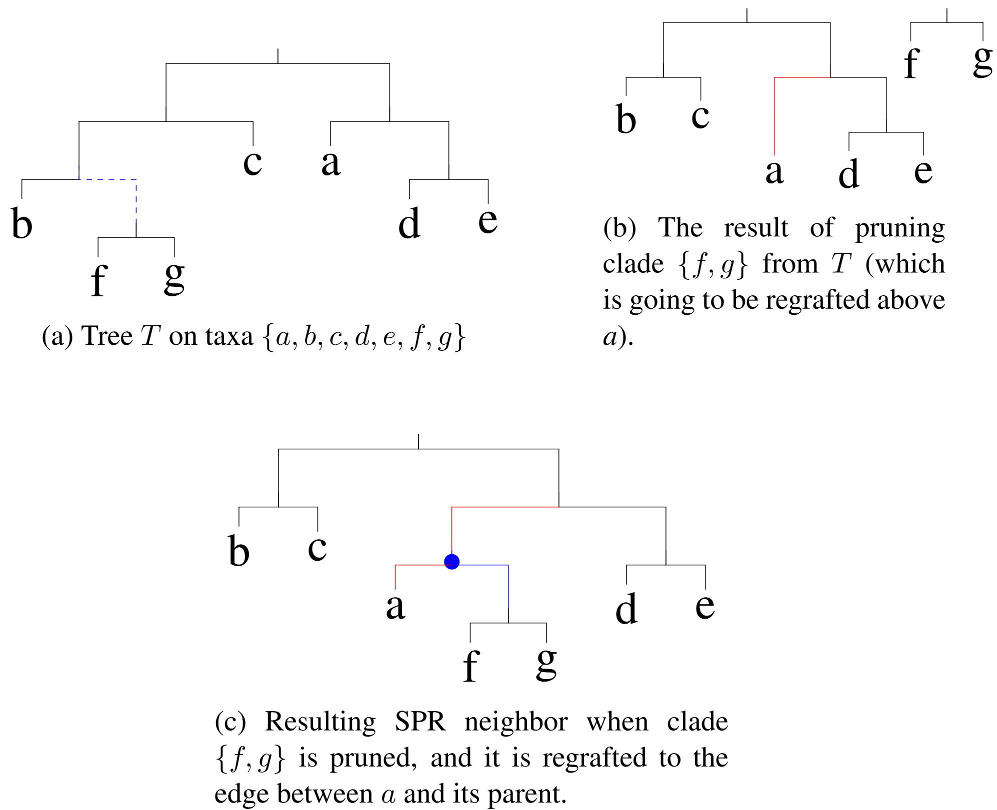
Note that RF distance is defined only between two phylogenies with the same set of taxa. Thus, when calculating the RF distance between supertree and source tree, we should first *restrict* the supertree to the source tree, and then calculate the RF distance. This restriction is done by removing non-shared taxa from supertree and contracting degree 2 edges until the supertree and source tree have the same set of taxa.

3.3 Designing SPR Hill-Climbing Algorithm

One of the most successful heuristic methods for supertree problems is to search the space by *branch swapping*. Although this hill climbing approach is not guaranteed to find the optimal solution, its effectiveness has been proven in many cases (For example [4] and [21]). In the context of supertree problem, no matter what the optimization problem is, the algorithms use this heuristic usually share three main steps.

First, we need a starting initial supertree. This tree can be provided by using one of the existing supertree algorithms. Another option is to construct a *greedy stepwise taxon addition* tree. In this approach, we randomly pick three taxa, and exhaustively find the topology with minimum distance to source trees. Then, other taxa are randomly added to the current tree one after another as follow. We exhaustively find the best place for the current taxon to be added to the tree that minimizes the total distance to the source trees. This approach, however, can be quite expensive.

Second, we need a *tree rearrangement operation* to generate the *neighborhood*. The neighborhood of a supertree can be obtain by applying all possible such rearrangement operations on the current supertree. The trees in the neighborhood have a topology that is a little different than that of the current supertree. There are three widely used tree rearrangement operations: *Nearest Neighbour Interchange (NNI)*, *Subtree Pruning and Regrafting (SPR)*, and *Tree Bisection and Reconstruction (TBR)*. For unrootd trees, NNI consists of swapping two of the subtrees on the opposite ends of an internal branch. Two rearrangements are possible for each internal edge, and thus NNI results in a neighborhood of size $2(n-3)$, where n is the number of taxa. SPR rearrangement consists of identifying and removing a subtree, and reattaching it to some branch of the remaining tree. SPR results in a $\mathcal{O}(n^2)$ neighborhood. And finally, in TBR, we first divide the tree into two parts, then we reconnect them by each possible pair of branches. Thus, TBR results in



(a) Tree T on taxa $\{a, b, c, d, e, f, g\}$

(b) The result of pruning clade $\{f, g\}$ from T (which is going to be regrafted above a).

(c) Resulting SPR neighbor when clade $\{f, g\}$ is pruned, and it is regrafted to the edge between a and its parent.

Figure 3.2: rSPR operation.

$\mathcal{O}(n^3)$ neighborhood. It is not that difficult to see that NNI is a special case of SPR, and SPR is a special case of TBR.

These rearrangements have been defined for both rooted and unrooted trees. In this work, we will use *Rooted Subtree Pruning and Regrafting (rSPR)* operation as the tree rearrangement operation. In rSPR operation, we first choose a non-root node v of the rooted tree, T. Then, the subtree rooted at v is pruned from the tree by removing the edge between v and its parent. After suppressing the degree-two node, the pruned subtree is regrafted to an internal edge of the remaining tree. This operation is depicted in Figure 3.2. In the remainder of this work, we refer to the rSPR as SPR for simplicity.

Finally, we need fast algorithms to find the best supertree in the neighborhood, i.e. the

neighbor with minimum RF score. This problem is usually called *local search problem*. We will use RFS algorithm [4] to solve our local search problem. Section 3.4 is dedicated to describe this algorithm in details.

To sum up, we will use existing supertree algorithms to generate initial supertree. Further, we will use rSPR tree rearrangement operation to generate the neighborhood of the current tree in search. Then, we use RFS algorithm to solve the local search problem. Lastly, the current supertree is replaced by its best neighbor, i.e. a neighbor with lowest RF score to the source trees, and we go back to the second step, and repeat. One simple common stopping criteria for hill-climbing algorithms is to stop when reaching a local optimum. However, we will use Edge Ratchet technique, as it will be explained later, as stopping criteria, instead.

3.4 Solving Local Search Problem by RFS Algorithm

In order to implement ER-RFS algorithm, we need a fast local search algorithm to be able to find the best supertree (with minimum RF score) in the SPR neighborhood. Fortunately, such algorithm has already been introduced in [4]. However, the implementation of the algorithm is not available. Hence, we need to implement this algorithm. For this reason, before explaining the ER-RFS algorithm in more details, we are going to give a detailed explanation of the RFS algorithm. In the local search problem, we are given a supertree, and the goal is to find its best SPR neighbors which has the lowest RF score (cumulative RF distance to the source trees). The naive algorithm for SPR local search problem require $\mathcal{O}(kn^3)$ time, where k is the number of source trees, and n is the number of taxa. This is because there are $\mathcal{O}(n^2)$ SPR neighbors, and calculating RF distance between two trees takes $\mathcal{O}(n)$ time. The RFS algorithm provides a $\mathcal{O}(kn^2)$ algorithm for SPR local search problem.

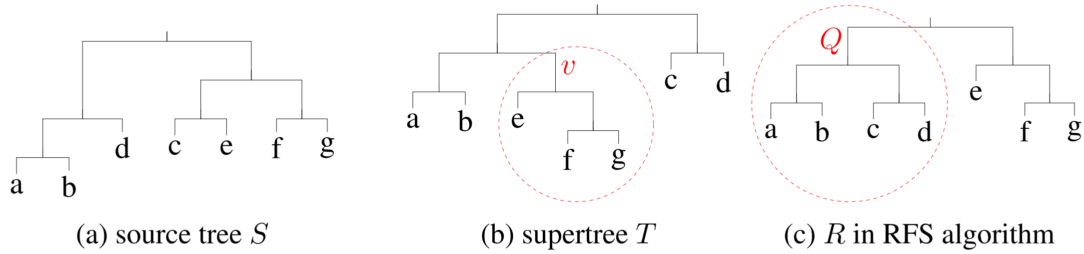
Technically, the core of RFS algorithm, is the algorithm that solves the *restricted SPR*

local search problem. In this problem, we are given a set of source trees and a supertree, and also a specific node in the supertree, v, to be pruned. The goal is to find the best regraft location for v such that the RF score is minimized. The naive solution for this problem takes $\mathcal{O}(kn^2)$. The RFS algorithm solves this problem in $\mathcal{O}(kn)$. This immediately leads to the $\mathcal{O}(kn^2)$ above mentioned algorithm for SPR local search problem.

Note that, in this work, we use *RFS algorithm* to refer to two algorithms: 1- the overall RFS algorithm as to solve RF supertree problem [4], and 2- the algorithm used in RFS algorithm to solve the restricted RF local search problem. However, from the context, it will be clear which one we are referring to.

The idea behind RFS algorithm is to somehow find the best regraft place for v without actually calculating RF distance for all possible regraft places. For simplicity, assume we have only one source tree S and a given supertree T, and that both trees have the same set of taxa. Note that if T has more taxa than S, then we can restrict it to the taxa set of S in $\mathcal{O}(n)$ time. Further, it is trivial to extend the algorithm to the case where we have more than one source tree. Now, suppose we are given a node v in T to be pruned, and an internal node in S, u. The idea is that, for each node u, we can determine for which regraft places the clade u will or will not exist in the resulting SPR neighbor. We can do this efficiently,and then we can use this information from all such nodes u to find the best regraft place. For example, suppose node u is clade $\{f, g\}$ in tree S in Figure 3.3. Since this u exists in v (as well as T), no matter where we regraft v, u will exist in the resulting neighbor, too. Thus, node u does not play a role in the *change* in RF score for this supertree when v is pruned.

In RFS algorithm, we first change T into a new tree R which has exactly same SPR neighborhood as T when v is pruned. But, using R in the algorithm makes it much easier to implement. For example, in Figure 3.3, tree R is obtained from T by pruning v and regrafting it back to the root of T. Note that the node Q represents the clade consisting



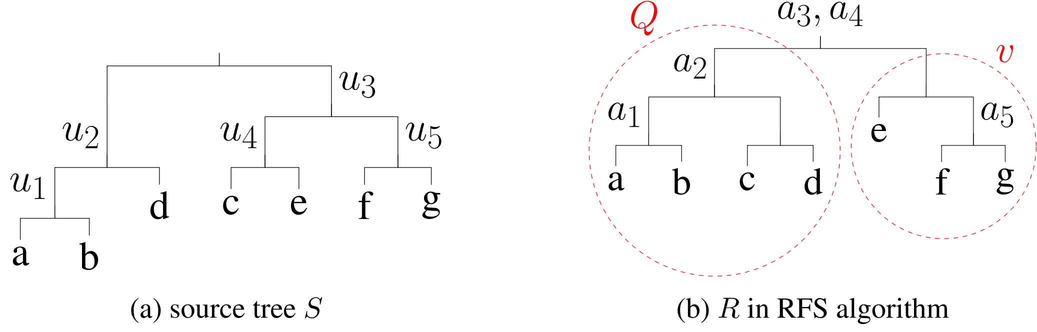
(a) source tree S (b) supertree T (c) R in RFS algorithm

Figure 3.3: An instance of the input for RFS algorithm: S and T, and $v \in T$ to be pruned. Tree R is obtained from T by pruning v and regrafting it back to the root of T. The node Q represents the clade consisting of all possible regraft places for v.

of all possible regraft places for v, and hence, for given v, T and R have the same SPR neighborhoods.

In order to determine if a clade u in source tree exists in supertree, all we need to do is to obtain its *Least Common Ancestor (LCA)*, i.e. the most recent common ancestor of the taxa in cluster u in T, called a. Then, clade u exists in T if and only if a and u has the same number of taxa in their clades. Both of these two can be done in $\mathcal{O}(n)$ [4] [28], where n is the total number of taxa.

There are actually only four cases in RFS algorithm to consider for a given v and arbitrary u. We consider tree R instead of T in the RFS algorithm. Suppose the LCA of u in tree R is a. For example, in Figure 3.4, the LCA of u_1 is a_1, and since both have 2 taxa in their clades, we know clade u_1 exists in R. On the other hand, the LCA of u_2 is a_2. Since the number of taxa differ in u_2 and a_2, we know u_2 does not exist in R. Before presenting those four cases, we first need to introduce a couple of notations (the same way the are defined in [4]): 1- $L(T)$ is the leaf set of tree T. 2- T_v is the subtree of T rooted at v, for some $v \in L(T)$. 3- the partial order $\leq_T$ is defined according to the ancestor-descendant relationship. For nodes x and y in T, we say $x \leq_T y$ if y is a node on the path between x and root of T. 4- $f_T(u)$ for node $u \in S$ is a boolean function that says whether



(a) source tree S

(b) R in RFS algorithm

Figure 3.4: For each of the non-trivial clades u_i in S, where $1 \leq i \leq 5$, the node a_i represents its LCA in R.

clade u exists in T, i.e. $f_T(u) = 1$ means u exists in T, and $f_T(u) = 0$ means u does not exist in T.

Lets assume tree T' is the resulting SPR neighbor when v is pruned from T. Also, lets denote the regraft place by x, i.e. $x \in Q$. It can easily be proven [4] that there are only four cases given a v to be pruned, and an arbitrary $u \in S$.

(i) if $a \in R_v$ then $f_{T'}(u) = f_R(u)$ for any $x \in Q$. For example, node u_5 in Figure 3.4.

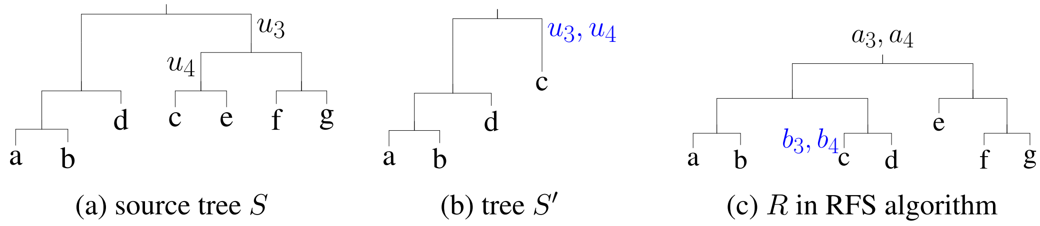
(ii) if $a \in Q$ and $f_R(u) = 1$, then

- $f_{T'}(u) = 0$, for $x \leq_R a$, and
- $f_{T'}(u) = 1$, otherwise

For example, node u_1 in Figure 3.4.

(iii) if $a \in Q$ and $f_R(u) = 0$, then $f_{T'}(u) = 0$ for any $x \in Q$. For example, node u_2 in Figure 3.4.

(iv) if a is root of R, we first find tree S' which is obtained from S by suppressing all nodes m whose LCA is in R_v. Figure 3.5 illustrates how S' is constructed from S. Then we find LCA of u in S' in R, and we call it b.



(a) source tree S (b) tree S' (c) R in RFS algorithm

Figure 3.5: Tree S' is obtained from S by suppressing all nodes m whose LCA is in R_v, i.e. clades $\{f, g\}$ and $\{e\}$. Clades u_3 and u_4 are actually clade $\{c\}$ in S'. Hence, their LCA is clade $\{c\}$ in R which are denoted by b_3 and b_4, respectively.

- $f_{T'}(u) = 1$ if and only if $x \leq_R b$ and $\mid L(R_b) \mid + \mid L(R_v) \mid = \mid L(S_u) \mid$. For example, node u_3 and u_4 in Figure 3.4. Note, however, the second condition is true only for u_3, and $f_{T'}(u_3) = 1$ if v is regrafted back at c.

Now, we are ready to describe RFS algorithm. For a given node v to be pruned in T, for any $x \in Q$, let $A(x) = \mid \{u \in S : f_R(u) = 0$, but $f_{T'}(u) = 1\} \mid$, and $B(x) = \mid \{u \in S : f_R(u) = 1$, but $f_{T'}(u) = 0\} \mid$, where T' is the result of the SPR operation when v is pruned and regrafted on top of x. By definition, the best x (with lowest RF score) is the a node x for which $\mid A(x) \mid - \mid B(x) \mid$ is maximized. The RFS algorithm efficiently calculates $A(x)$ and $B(x)$ at each node x as follow.

In a preprocessing step, we first construct R from T, and we compute the LCA of all (non-trivial) clades $u \in S$ in R. We also compute the size of leaf set for all nodes in both S and R. Further, we initialize two counters $\alpha(x)$ and $\beta(x)$ at each node $x \in Q$ to 0. This takes $\mathcal{O}(n)$. At the end of the algorithm, the values of $\alpha(x)$ and $\beta(x)$ will be the values of $A(x)$ and $B(x)$, for each node $x \in Q$.

The algorithm then traverses S and considers each non-trivial node u in S. There are three cases:

1. If u satisfies the precondition of (i) or (iii), then $f_{T'}(u) = f_R(u)$ for any $x \in Q$, and

we do nothing.

2. If u satisfies the precondition of (ii), then we increment the values of $\beta(x)$ for each node $x \in T_a \backslash \{a\}$.

3. If u satisfies the precondition of (iv), and if $\mid L(R_b) \mid + \mid L(R_v) \mid = \mid L(S_u) \mid$, then we increment the values of $\alpha(x)$ at each node $x \in T_b$.

Note that node u in S is a non-trivial clade, i.e. an internal node (except root). However, the node v can be any internal node or a leaf. It is not hard to see that all these can be done in $\mathcal{O}(n)$ [4]. Applying this algorithm for all possible v's in T results in $\mathcal{O}(n^2)$ algorithm for SPR local search problem.

Note that RFS algorithm can easily be extended to the case where the edges in the source trees are weighted. First, note that an edge in some *rooted* source tree corresponds to the cluster of the child endpoint of that edge. Further, in RFS algorithm, we visit each internal node in all source trees, and we update $\alpha(x)$ and $\beta(x)$ based on the position of its LCA and the conditions explained above. Hence, if an edge in some source has a weight of w, this can be interpreted as if the corresponding cluster appears w times. In other words, it is the same as assuming that the cluster is visited w times. Therefore, all we need to do is to increment values of $\alpha(x)$ and $\beta(x)$ by w instead of 1.

3.5 Edge Ratchet Technique to Deal With Local Optimum Problem

Given RFS algorithm to solve restricted SPR local search problem as described above, we can easily solve the SPR local search problem in $\mathcal{O}(n^2)$ by applying the algorithm for all possible nodes to be pruned in the tree. Therefore, we can easily develop a hill-climbing algorithm using this local search algorithm as explained in section 3.3. Algorithm 1 outlines this hill-climbing heuristic algorithm.

Using Algorithm 1 we can improve the accuracy of the initial supertree in terms of RF

Algorithm 1: SPR HILL-CLIMBING Algorithm

Input: A set Σ of rooted phylogenetic trees with partially overlapping taxa, and a rooted binary initial supertree T

Output: A supertree of the input trees which is a local optimum for RF supertree problem

1 **while** *(true)* **do**
2 Find the best SPR neighbor with lowest RF score, T', by applying RFS algorithm for all possible nodes $v \in T$ to be pruned.
3 **if** *(T''s RF score is lower than that of T)* **then**
4 replace T with T'
5 **else**
6 *break*
7 **return** T

score if it is not already a local optimum. However, like other hill-climbing algorithms, this approach may get caught in a local optimum with relatively high RF score while there are better supertrees around in the solution space. Edge Ratchet is a technique that will be used to deal with this problem.

How can we possibly make a perturbation at the local optimum so that the hill-climbing algorithm can escape from the current hill and start the search at another hill with a potentially better local optimum. There are three main directions by which to achieve this goal.

First, we can try to somehow change the topology of the supertree at the local optimum in the hope that this change drifts us to a better hill in the solution space. One simple way to achieve this is to perform one or several *random* SPR moves on the supertree at local optimum, and start over the search. We tested this idea. However, the results were not promising. If we perform only one SPR move, then in the first iteration of the next search, we will get back to the same local optimum where we were previously, no surprise! If we perform, say, three or more random SPR moves, then this usually causes a huge increase

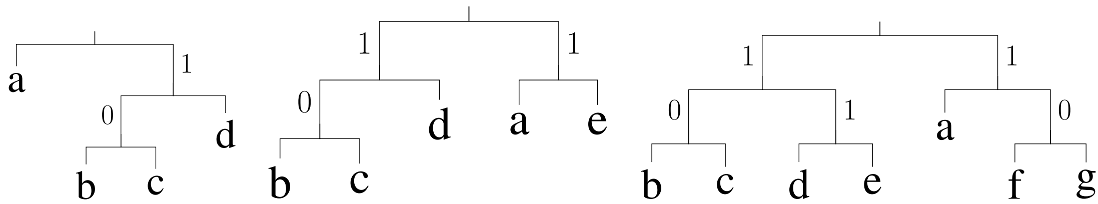


Figure 3.6: Three source trees with randomly re-weighted 50% of edges (clusters) to 0.

in the RF score so that the next hill-climbing search has to spend a lot of time to improve the resulting tree. Surprisingly, we almost always ended up to the same local optimum anyways.

Second, we can change the optimality criteria by which we choose a neighbor. For example, we can use some other tree distance measures as optimality criteria. We attempted this idea by using *quartet distance* as secondary optimality criteria. A *quartet* is an unrooted tree with only four taxa. The set of quartets of a tree is defined as all such quartets induced by considering each internal edge of the tree, and picking each possible two taxa on each side. The quartet distance between two unrooted phylogenies is defined as the number of unique quartets in each tree. We tested this idea by performing several hill-climbing searches one after another altering the optimality criteria between RF distance and quartet distance. The final tree in each hill-climbing search was used as initial supertree of the next hill-climbing search. However, this strategy was not promising either. We usually ended up with supertrees with greater than or equal RF score than that of the initial supertree. Changing the optimality criteria from RF distance to quartet distance and vice versa does not seem to be effective at drifting the us to a better hill in the solution space.

The third strategy is to somehow change the source trees! This is what *edge ratchet* technique does. Note that the RF score depends on the bipartitions (or clusters) exist in each source tree. Further, there might be conflicts among these bipartitions. i.e. achieving

a RF score of 0 is usually impossible. The idea behind edge ratchet is to pick a random subset, usually 30%, of the clusters in each source tree, and give them a different weight. For example, if we give a weight of w to some cluster, this will be interpreted as having that cluster w times, in the algorithm . Or giving a weight of 0 to a cluster, is equivalent to removing that cluster. Figure 3.6 illustrates one possible way of re-weighting a potion of clusters (or edges) in the source trees of Figure 3.7. Note that giving a new weight to a cluster, in this context, is the same as giving that weight to the corresponding edge in the tree.

Our results show that this simple idea can be effective in drifting the local optimum to a better hill in the solution space. Note that, using this strategy, we are not going to end up in a completely irrelevant supertree. Actually, the resulting supertree still minimizes the RF distance, but to a subset of bipartitions instead of all bipartitions. Our experiments show that the resulting tree is different enough to happen to be on another hill with better RF score in most of the cases.

3.6 ER-RFS Algorithm

Now we have all the building blocks to design ER-RFS algorithm. To recapitulate, the optimality criteria for ER-RFS algorithm is rooted RF distance. Further, we use rooted SPR edit operation, rSPR, to produce the neighborhood in each iteration of the hill-climbing search. Applying all such operations on a supertree with n leaves, will result in $\mathcal{O}(n^2)$ SPR neighbors. We use RFS algorithm on each node of the supertree to be pruned to find the best SPR neighbor (with minimum RF score) among all its $\mathcal{O}(n^2)$ SPR neighbors. This results in a hill-climbing algorithm described in Algorithm 1.

Further, in order to deal with local optimum problem, we use edge ratchet technique to re-weight edges in each source tree. Using this technique along with Algorithm 1 results in ER-RFS algorithm depicted in Algorithm 2.

Algorithm 2: ER-RFS Algorithm

Input: A set Σ of rooted phylogenetic trees with partially overlapping taxa, and a rooted binary initial supertree T

Output: A supertree of the input trees which is a local optimum for RF supertree problem

```
1  best_score = RF score of T
2  best_supertree = T
3  for i ← 1 to 50 do
4      T = Algorithm 1 (Σ, T)
5      if (T's RF score ≤ best_score) then
6          best_score = T's RF score
7          best_supertree = T
8      if (i ≤ 50 ) then
9          Re-weight 30% of internal clusters of each source tree to 0 to obtain Σ'
10         T = Algorithm 1 (Σ', T)
11     Reset all edge-weights to their original value, 1
12 return best_supertree
```

To illustrate these algorithms, suppose we are given a set Σ of three input trees as in Figure 3.7, and an initial supertree T as Figure 3.8. The set of SPR neighbors of T consists of all possible SPR moves on this supertree. For example, suppose we want to prune the clade (f, g), and produce all possible SPR neighbors corresponding to regrafting it back to the tree, Figure 3.9b. There are seven such SPR neighbors. Figure 3.9 illustrates three of those neighbors when (f, g) is regrafted to the edge above clades (b, c), (d, e), or a, respectively.

For each node *v* in the supertree to be pruned, we apply RFS algorithm to find its best regraft place. Among all such best SPR moves, we pick the one with the lowest RF score. Then, if this neighbor has a better RF score than that of the the current supertree, we replace the current supertree with this new neighbor. We continue this until we reach

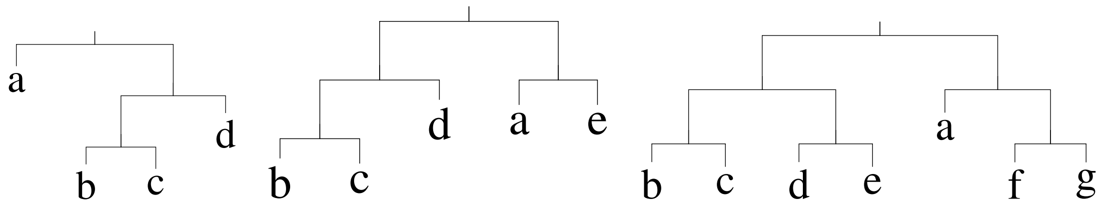


Figure 3.7: Three source trees on the taxa set $S = \{a, b, c, d, e, f, g\}$.

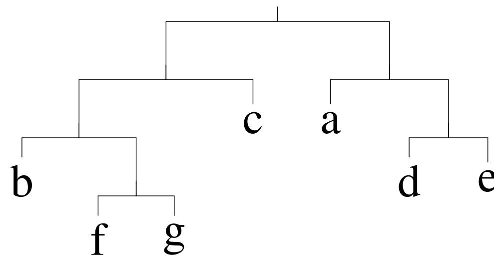
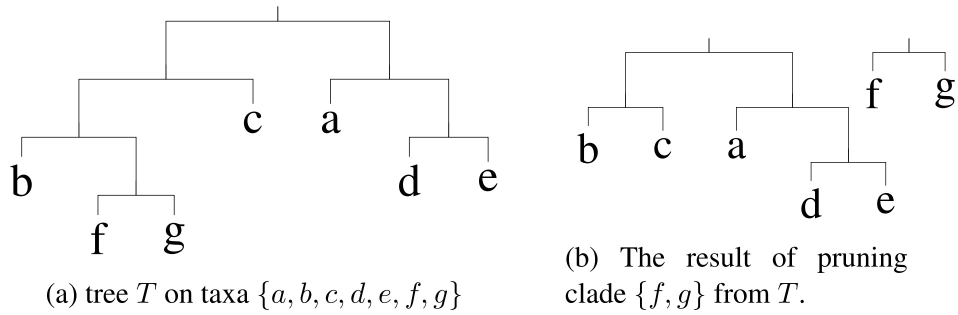


Figure 3.8: A supertree, T, on the taxa set $S = \{a, b, c, d, e, f, g\}$.

a local optimum. Then we replace T with the result of this hill-climbing search. This is what the line 4 of the Algorithm 1 does.

after line 4 of the Algorithm 1, the current tree, T, is a local optimum. At this point, we use ER technique, and we randomly reweight, say, 30% of internal edges in each source tree to a new weight of 0. As we mentioned earlier, Figure 3.6 illustrates one of possible ways to reweight the edges in source trees from Figure 3.7. This is what line 9 of the Algorithm 2 does.

Now we perform another hill-climbing search starting from current (local optimum) supertree, and weighted source trees, line 10 of the Algorithm 2. As it was explained earlier, it is fairly straightforward to extend RFS algorithm for the weighted source trees. Remember in RFS algorithm, we visit each internal node in each source tree, and based on its LCA position in supertree we modify $\alpha(x)$ and $\beta(x)$. Now suppose some cluster has a weight of 0. In this case, the RFS algorithm simply ignores this node. The rest of the



(a) tree T on taxa $\{a, b, c, d, e, f, g\}$

(b) The result of pruning clade $\{f, g\}$ from T.

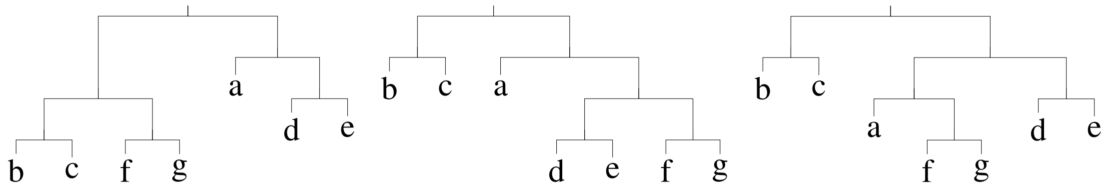


Figure 3.9: SPR neighborhood. There are 7 edges on which we can regraft (f, g) back to the tree, excluding the one it was attached to previously. Here we have three of those neighbors which are the result of regrafting (f, g) to the edge above the clades $\{b, c\}$, $\{d, e\}$, or $\{a\}$, respectively.

algorithm remains the same. To finish this ratchet iteration, we reset the weight to their original value 1, line 11 of the Algorithm 2.

These two consecutive hill-climbing searches are considered as one ratchet iteration. The supertree at the end of one ratchet iteration is used as the initial supertree of the next ratchet iteration. We perform 50 of these ratchet iterations, except in the last iteration, we only perform the first hill-climbing search since the solution of the second search is not a valid output. Lines $5-7$ of the Algorithm 2 just keep track of the best seen supertree. The best seen supertree is returned at the end of the algorithm.

The time complexity of the ER-RFS algorithm depends on four parameters: the number of taxa on the supertree n, the number of source trees k, the number of ratchet iterations performed (50 here), and the number of iterations required to reach a local optimum at each call of the Algorithm 1, m. We have no control over the latter, and it mainly depends on

the quality of the initial supertree used in that search. Lets assume M to be the maximum number of iterations required to reach a local optimum given any initial supertree. Note that each call to the Algorithm 1 requires $\mathcal{O}(Mkn^2)$ time, where n is the number of taxa on the supertree, and k is the number of source trees. Therefore, the time complexity of the ER-RFS algorithm is $\mathcal{O}(100Mkn^2)$, or $\mathcal{O}(Mkn^2)$.

3.7 Experimental Results

In this section, we present the experimental results of the ER-RFS algorithm comparing it to the well-known existing supertree algorithms. ER-RFS algorithm is implemented in C++. Note, all experimental tests were executed on the Texas A&M Brazos supercomputer using 2.5Mhz, 1 core nodes each with 100-300 MB of memory based on the size of data set. In what follows, we first explain our experimental methodology, and then we present the results.

3.7.1 Datasets

In our experiments, we used four empirical datasets including Sea birds (121 taxa, 7 source trees) [29], Placental mammals (PM, 116 taxa, 726 source trees) [30], marsupials (267 taxa, 158 source trees) [31], and Temperate herbaceous papilionoid legumes (THPL, 558 taxa, 19 source trees) [32]. Most of these datasets where previously used to evaluate supertree methods; all datasets have rooted source trees.

3.7.2 Measurements and Other Supertree Algorithms

As it was mentioned above, we are going to use empirical data for experimental analysis. Since the true species tree is not available for empirical data, we have to evaluate the accuracy of the supertree based on its similarity to the source trees. This similarity is usually quantified by a tree distance measure. Thus, we can evaluate the accuracy of our supertree algorithm by comparing its similarity to source trees against other supertree

algorithms.

Among different phylogenetic tree distance measures, parsimony score and RF distance are the two most widely used ons. We use these two measures along with running time to compare different supertree algorithms. We will compare our algorithm with three well-known, successful supertree algorithms in the literature: MRP, SuperFine-MRP (SuperFine used with MRP in the resolution step), and RFS. MRP is the most widely used supertree algorithm in the literature. SuperFine algorithm is an improvement to MRP algorithm, and has been shown to be able to produce high quality supertrees [11]. RFS algorithm is the best existing algorithm for RF supertree problem. We used the implementation of these algorithms from `http://phylosolutions.com/paup-test/` (*PAUP**), `https://github.com/dtneves/SuperFine/`, and `http://genome.cs.iastate.edu/CBL/RFsupertrees/` for MRP, SuperFine-MRP, and RFS, respectively. Further, for parsimony ratchet, we used the code provided in the appendix of [33].

3.7.3 Calculating RF Score

As it was mentioned before, RF distance is defined only for trees with the same set of taxa. Thus, when calculating RF distance between the supertree and any of the source trees, we first restrict the supertree to the taxa set of the source tree against which it is compared. This is done by removing non-shared taxa from supertree and contracting degree 2 edges until the supertree and source tree have the same set of taxa, Figure 3.10.

3.7.4 Calculating Parsimony Score

As it was mentioned in Section 2.1, the parsimony score for a tree refers to the sum of the smallest number of substitutions needed for each site (column) of the taxa-character matrix. Further, we illustrated in Figure 2.1 how to obtain the matrix representation of a set of given source trees. Given a supertree and the matrix representation of the source trees, we can calculate parsimony score using, for example, *Fitch's* algorithm. The Fitch's

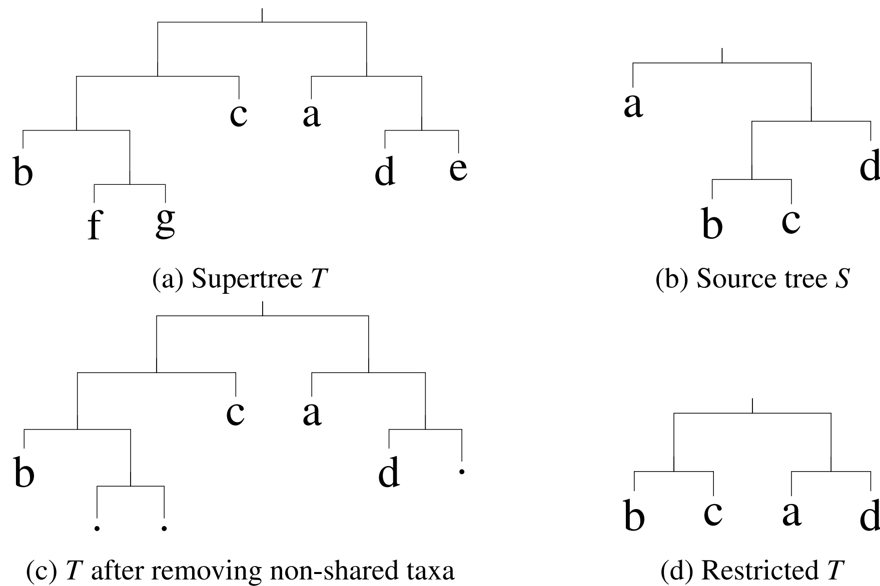


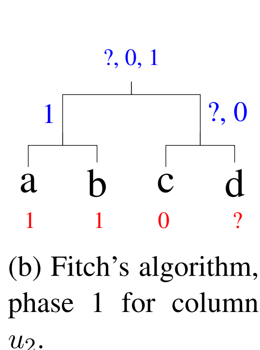
Figure 3.10: In order to calculate RF score between supertree T and source tree S, we first need to remove non-shared taxa from T (c). Then, we suppress all non-labeled leaves and degree 2 internal edges which results in the tree in (d). The RF distance between T and S is defined as the RF distance between this restricted T and S.

algorithm actually solves *small parsimony problem*. In this problem, we are given a tree topology and a taxa-character matrix, and th goal is to find the character states in the internal nodes which results in minimum parsimony score (i.e. minimum number of character state changes along the edges of the tree). Unlike *large parsimony problem*, which is the goal of MRP supertree algorithm and an NP-hard problem, the small parsimony problem can be solved efficiently in polynomial time.

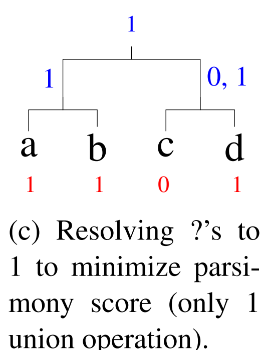
To illustrate Fitch's algorithm, suppose we are given the two source trees as in Figure 3.11. We can make the matrix representation of the given trees as it is shown in Figure 3.11a. Now suppose we have obtained a supertree of these source trees using some supertree algorithm, Figure 3.11b. The parsimony score of the tree is defined as the sum of the parsimony scores of all the columns of the matrix. For example, in Figure 3.11, we have shown calculations for column 2 of the matrix, corresponding to clade u_2. We first

taxa	u_1	u_2	u_3	u_4
a	1	1	1	1
b	1	1	1	1
c	1	0	?	?
d	?	?	1	0

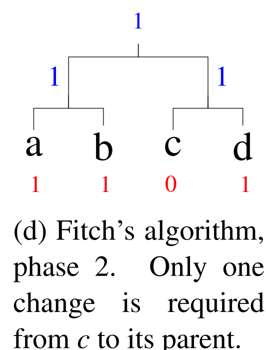
(a) Two source trees an taxa set $\{a, b, c, d\}$, and their matrix representation.



(b) Fitch's algorithm, phase 1 for column u_2.



(c) Resolving ?'s to 1 to minimize parsimony score (only 1 union operation).



(d) Fitch's algorithm, phase 2. Only one change is required from *c* to its parent.

Figure 3.11: Parsimony score calculation. Given matrix representation of part *a* and the supertree in part *b*, we calculate the parsimony score of the supertree for each site (column) of the matrix (such as u_2). In above example, the parsimony score corresponding to the column 2 of the matrix is 1. The final parsimony score of the supertre is defined as the sum of the parsimony scores over all columns.

write down each character on its corresponding leaf on the tree. The Fitch's algorithm has two phases. In the first phase of the algorithm, we calculate all possible character states, R_i, for all internal nodes using a post-order traversal of the tree using the following rule, Figure 3.11b and 3.11c. The number of union operations is actually the parsimony score (hence we are done here for the purpose of calculating parsimony score. But we will complete the algorithm to obtain labeling of internal nodes as well).

$$R_i = \begin{cases} R_j \cap R_k, & \text{if } R_j \cap R_k \neq \emptyset \\ R_j \cup R_k, & \text{otherwise} \end{cases}$$

In the second phase, we perform a pre-order traversal of the tree using below rule to determine character states of the internal nodes that is most parsimonious. Note that ?'s can be either treated as a valid character state, or they can be resolved to the whichever character state that minimizes parsimony score. In Figure 3.11d, we resolved ?'s to 1's to minimize parsimony score. Let s_i denotes the state of the internal node *i* with parent *j*.

$$s_i = \begin{cases} s_j, & \text{if } s_j \in R_i \\ any \in R_i, & \text{otherwise} \end{cases}$$

We used *r8s* software [34]) to obtain Baum-Ragan matrix representation of the source trees. Further, we obtained the parsimony score of each supertree using this taxa-character matrix with PAUP [9].

3.7.5 ER Configuration

The ER algorithm can be configured based on the portion of internal edges to be re-weighted, and the new weight which is assigned to selected edges. We tested the algorithm on four available biological datasets (explained below), using different configurations. We tried 10%, 20%, 30%, 60%, and 80% of the edges to be re-weighted with a new weight of 0, 2, 5, 10, and 50 (a total of 25 configurations on each data set).

Among these configurations, it was observed that re-weighting 30% of the edges to 0 as new weight is almost always more effective in obtaining a better supertree, and yields a lower RF score. Thus, in the results reported below, we stick to this configuration in all experiments. Further we used 50 ratchet iterations in all experiments.

3.7.6 Terminology

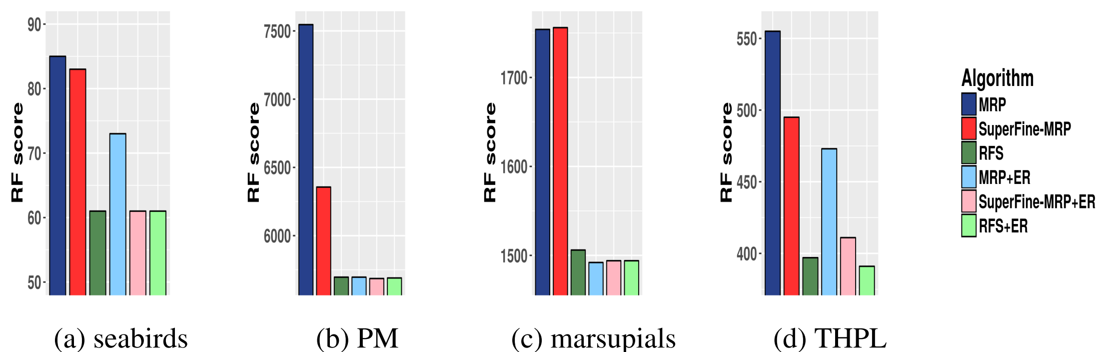
In order to present the results we need a consistent terminology. Note some algorithms, by default, build a greedy stepwise addition taxa tree, and use it as initial supertree. This is usually done by randomly picking three taxa, and then finding the best triplet to optimize the optimality criteria used, exhaustively. Then, another taxon is pick randomly, and is added to the tree in a place where optimizes the optimality criteria used, exhaustively. This process continues until all taxa are added to the (super)tree.

For simplicity, we refer to ER-RFS algorithm as ER algorithm. We use a simple representation when referring to a specific algorithm: $<$*initial supertree*$>$ + $<$*algorithm*$>$, where initial supertree can be the output of another algorithm. For example, suppose we run RFS algorithm on some dataset with its default stepwise taxon addition tree to obtain a supertree, T. Then, RFS+ER refers to the the ER algorithm when T is used as initial supertree for ER algorithm. When no initial tree is provided, we should use *default TA* + $<$*algorithm*$>$ terminology. However, for simplicity, we omit the *default TA* part. For example, RFS refers to the RFS algorithm when no initial tree is provided.

3.7.7 Results

To evaluate effectiveness of the ER algorithm, we compare it with three well-known supertrees: MRP, SuperFine-MRP (SuperFine used with MRP in the resolution step), and RFS. Our goal is to improve these supertrees. We use each of them as initial supertree for our algorithms and test if they can improve RF score of the initial supertrees. Especially, we are interested in improving RF score of RFS algorithm which is the best known algorithm for RF supertree problem.

Because of the random steps involved in both RFS and ER algorithms, we had 10 runs of each algorithm on each data set, and the supertree with best RF scores in each case are reported below. Although in some cases, the best score only happened once, usually the



(a) seabirds (b) PM (c) marsupials (d) THPL

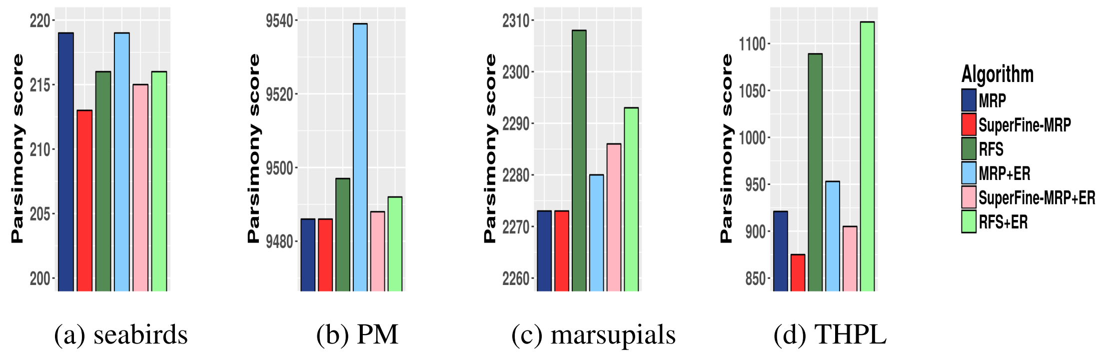
Figure 3.12: RF score comparison of MRP, SuperFine-MRP, RFS, and three versions of ER algorithm: MRP+ER, SuperFine-MRP+ER, and RFS+ER.

supertrees obtained in the ten runs have very close RF scores.

Note that RFS supertree is optimized for RF score, and is indeed a relatively good local optimum already. Thus, if ER algorithm is able to obtain a lower RF score, this is significant, and it could provide evidence on effectiveness of ER algorithm to deal with local optimum problem.

Below we compare MRP, SuperFine-MRP, and RFS with three versions of ER algorithm (MRP+ER, SuperFine-MRP+ER, and RFS+ER). First we compare the RF score (cumulative RF distance to source trees), then we compare parsimony score, and at the end, we compare the running times.

As we can observe in Figure 3.12, ER algorithm can effectively deal with local optimum problem, and improve RF score considerably. Specifically, ER algorithm can improve RF score of RFS supertree (which already is a very good local minimum) on all datasets which provides evidence on the success of ER algorithm to deal with local optimum problem. However, note that because of the large scaling in the plots, this improvement is not that clear. On seabirds dataset, RFS and RFS+ER both get a RF score of 61. On PM dataset, RFS get score of 5696, and RFS+ER is able to get a score of 5690. Overall, FRS seems to do a good job to obtain a supertree with low RF score. Feeding this



(a) seabirds (b) PM (c) marsupials (d) THPL

Figure 3.13: Parsimony score comparison of MRP, SuperFine-MRP, RFS, and three versions of ER algorithm: MRP+ER, SuperFine-MRP+ER, and RFS+ER.

supertree to ER algorithm can make it even better.

MRP+ER and SuperFine-MRP+ER both improve considerably the RF score of MRP and SuperFine-MRP, respectively. Notably, on the two datasets with larger number of trees, PM and marsupials, ER algorithm is able to obtain almost the same RF scores no matter what initial supertree is. However, on the other two datasets (with small number of source trees), initial supertree seems to have a huge effect on the accuracy of the final supertree. This can be explained by the fact that having larger number of source trees, usually provides more information. In other words, we expect to have an increase in the number of shared clusters in the source trees as the number of source trees increase. This can help the algorithm to better handle conflicts among source trees.

MRP+ER has relatively higher RF score on seabirds and THPL in compare to other versions of ER algorithm. This is probably because of the small number of source trees in these data sets, and that the MRP has higher RF score on these datasets.

Figure 3.13 compares the parsimony score of the algorithms. As we mentioned earlier, both MRP and SuperFine-MRP minimize MRP score, and as we expect, they they usually have better parsimony score when compared to other supertrees. SuperFine-MRP obtains the best parsimony score on all datasets.

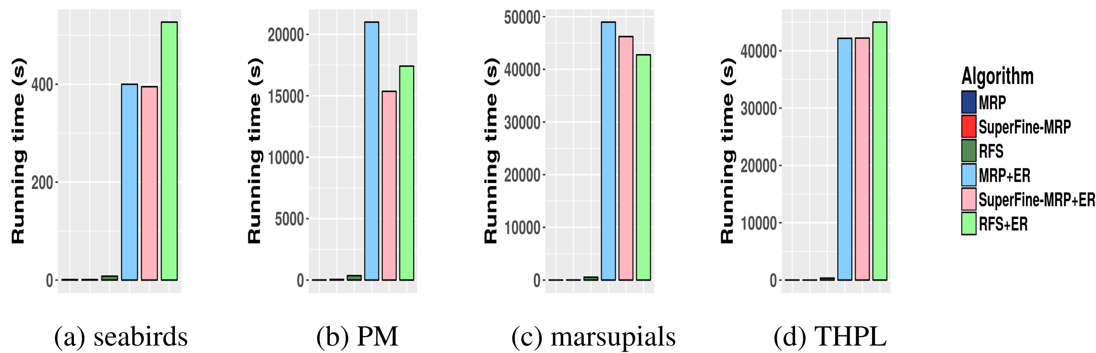


Figure 3.14: Running time comparison of MRP, SuperFine-MRP, RFS, and three versions of ER algorithm: MRP+ER, SuperFine-MRP+ER, and RFS+ER.

RFS and RFS+ER usually have higher parsimony score. The difference between their parsimony scores and that of other algorithms increases as the size of dataset increases (with regard to the number of taxa). This suggests that the initial supertree can have a considerable impact on the final solution of the ER algorithm. In other words, we expect to find a final supertree with better parsimony score if the initial supertree is already optimized for parsimony score.

Although one might expect that RF score and parsimony score have correlations, we can observe that this is not the case on all these four biological datasets. Specially on the two datasets with higher number of source trees (PM and marsupials), MRP+ER has much higher parsimony score than MRP, while it has much lower RF score in compare to MRP. Another interesting observation here is that SuperFine-MRP+ER has relatively good parsimony score. Namely, it has better parsimony score than MRP on three datasets. Remember from Figure 3.12 that it always has much better RF score than both MRP and SuperFine-MRP. This is quite interesting because it shows that we can have supertrees with relative low RF and parsimony scores simultaneously!

Figure 3.14 compares the running time of all these algorithms. Clearly, ER algorithm is by far the most expensive (running time) algorithm. There are several reasons why

ER algorithm is more time consuming. First, because of the way ER algorithm works, we have to perform many hill-climbing searches (100 in our case) each of which could take several local search iterations which is quite expensive. Second, we used 50 ratchet iterations. But after looking at the score changes over iterations, we noticed most of the time the best supertree is obtained with less than 10 iterations. However, there were some cases in which some improvements happened after 25 iterations. This means that we could potentially obtain similar supertrees in about 80% less running time. Finally, one of the most expensive and frequently used operations in the algorithm is finding LCA of a given set of taxa on a tree. We used a very simple algorithm which is $\mathcal{O}(n)$, where n is the number of taxa on the tree. However, there are more complicated $\mathcal{O}(n)$ algorithms (like the one used in RFS) with much lower coefficients and constants which could improve running time dramatically.

One might wonder if it is worthwhile to spend a lot of time to improve the accuracy of the supertree slightly. First, note that in the case of supertree problem, the algorithm is going to be used only once. Thus, as long as the algorithm can finish in a reasonable time, it is perfectly fine. Further, even slight improvements of the accuracy of the supertree means resolving more conflicts among source trees which in turn might be very valuable for post analysis of the supertree by biologists. Note that we do not claim that ER algorithm is applicable to very large datasets, but we rather state that as long as it can finish in a reasonable time, it is a worthy algorithm.

4. SIMULATED ANNEALING ALGORITHM

4.1 Motivation and Algorithm Description

The local optimum problem is a critical issue in most of hill-climbing algorithms. Although ER algorithm is able to handle local optimum problem and improve RF score upon RFS algorithm, we do not know if there are better local optimums in the solution space. *Simulated Annealing* (SA) is known to be able to handle local optimum problem. Further, because of the way it works, it could potentially alleviate the effect of the initial supertree on the final supertree. Finally, since this algorithm does not require to search the whole SPR neighborhood, it could be much cheaper in terms of running time, and hence applicable to larger datasets.

Unlike greedy hill-climbing algorithms where we always pick a better neighbor in local search, simulated annealing algorithm, allows worse neighbors to be picked as well. Whether a worse neighbor is picked or not in SA algorithm, depends on three things: cost of current solution, cost of new neighbor, and current *temperature*, t. The SA algorithm introduces the notion of temperature which is used to define an *acceptance probability function* which provides a smart way to accept (better and maybe worse) neighbors. The higher the temperature, the higher the chance of a worse neighbor to be selected. The *initial temperature* is usually high, and it decreases every time a new neighbor is generated by a prespecified *cooling rate*.

In RF supertree problem, we are solving a minimization problem based on RF score. Thus, the cost function for a solution can be defined as its RF score. Suppose the current supertree T_i has RF score of s_i. Then, we generates a new *random* neighbor supertree T_{i+1}, and calculate its RF score, s_{i+1}. The acceptance probability function for T_{i+1} is defined as below.

$$P(T_{i+1}) = \begin{cases} 1, & \text{if } s_{i+1} \leq s_i \\ \exp(\frac{s_i - s_{i+1}}{t}), & \text{otherwise} \end{cases} \tag{4.1}$$

Where t is the current temperature. As we can see, a neighbor with lower RF score will always be picked since its acceptance probability is 1. But if the RF score of the new neighbor is worse (higher), it may or may not be picked based on some probability. At the beginning of the algorithm, the temperature t is relatively high, and this allows higher probability for worse neighbors to be picked. As the algorithm continues, the temperature decreases, and thus the probability of accepting a worse neighbor becomes very small.

The stopping criteria of the algorithm is determined by a prespecified *absolute temperature*. Every time a new neighbor is generated, the current temperature t is decreased by a fixed cooling rate η. The algorithm stops when the current temperature goes below the absolute temperature. Algorithm 3 shows how SA algorithm works.

4.2 Generating a Random Neighbor

In line 6 of the Algorithm 3 we need to generate a random neighbor of T. There is a very simple method to generate a new random neighbor in SA algorithm which we call *Random Prune, Random Regraft* (RR). In Random Prune, Random Regraft (RR) method, we first select a random node in the current supertree, *v*, to be pruned. Then all of its valid regraft nodes are found. These nodes consist of all nodes in the tree except all descendant of *v*, *v*'s parent, and *v*'s sibling. Finally, among these valid regraft nodes, we pick one randomly, and the corresponding SPR operation is performed to generate the new neighbor.

Although RR seems to be a good candidate to generate new neighbors for SA algorithm, our initial experiments showed that using this approach might require a lot of iterations until the algorithm converges. One reason for this behavior could be the fact that

Algorithm 3: SIMULATED ANNEALING Algorithm

Input: A set of rooted phylogenetic trees with partially overlapping taxa Σ, and initial supertree T

Output: A supertree of the input trees that is a local optimum for RF supertree problem

```
1  t_0: initial temperature
2  t_abs: absolute temperature
3  η: cooling rate
4  T_0: initial supertree with RF score of s_0
5  while t > t_abs do
6  |   generate a random neighbor of T_0, T, and calculate its RF score s
7  |   if s ≤ s_0 then
8  |   |   T_0 = T
9  |   else
10 |   |   T_0 = T with probability exp((s_0 − s)/t)
11 |   t = t(1 − η)
12 return T
```

there are usually very small number of *better* neighbors (with lower RF score) in the SPR neighborhood, specially when we are close to a local optimum. We will talk about this in more details in the results section. Nevertheless, the need for a better approach to generate a random new neighbor was unavoidable.

We needed some way to generate a neighbor which is random, but not as random as RR. In other words, it should generate a random neighbor with better RF score than RR, in average. Remember that RFS algorithm solves restricted RF local search problem. In this problem, we are looking for the best regraft place of a given node to be pruned in the supertree. This makes it a very suitable match for our purpose. We first choose a random node in the current supertree to be pruned, then we use RFS algorithm to find its *best* regraft place. Performing corresponding SPR operation will generate a new neighbor which is both random, and has relatively lower RF score than that of RR! We call this

method of generating neighbors *Random Prune, Best Regraft* (RB).

Note that RR is much faster than RB because in RB, we have to perform RFS algorithm once. There is a trade off here. With RR, we are able to perform much more number of iterations, but it might not result in good local optimums for RF supertree problem. On the other hand, we might be able to obtain good quality local optimums with RB with fewer number of iterations because it generates a better candidate at each iteration. However, there is no way to draw concrete conclusions before performing comprehensive experiments. We will compare these two approaches in the result section.

4.3 Annealing Schedule, the Challenging Part

In Simulated Annealing, the *Annealing Schedule*, or schedule for short, plays a key role in the performance of the algorithm. There are three main adjustable parameters in the algorithm:

- Initial Temperature: The temperature at the beginning of the algorithm.
- Cooling Rate: The speed by which the temperature decreases after each iteration [1].
- Absolute Temperature: The algorithm stops when the current temperature becomes smaller than the absolute temperature.

Each of these parameters could potentially have a huge effect on the performance, but in a different way. The higher the initial temperature, the higher the chance of accepting worse neighbors at early iterations of the algorithm. For example, choosing a very high initial temperature allows the algorithm to choose many neighbors with worse RF score at

[1] In the context of simulated annealing algorithm, an iteration simply refers to generating a new random neighbor and deciding whether to take it. This should not be confused with an iteration in the context of a hill climbing search algorithm where it refers to a complete neighborhood search and choosing the best one. Further, in the context of ER algorithm, an iteration refers to two complete complete hill-climbing searches, one with original weights, and one after re-weighting

early iterations. However, it is not trivial whether this help the algorithm to obtain a better final solution, or it will adversely affect the final solution.

On the other hand, absolute temperature controls the stopping criteria of the algorithm. The smaller we make the absolute temperature, the larger the number of iterations will be tried in the algorithm, and hence it is more likely to get a better RF score. However, making absolute temperature too small may only make the algorithm run for a longer time without any improvements. This is because at very low temperatures the algorithm acts like a greedy algorithm. On the other hand, if the algorithm is given enough time, it would eventually get into a local optimum. Therefore, like any other greedy algorithm, SA algorithm will get stuck at that local optimum when the temperature is very small. Note, however, the neighbors with same score might be selected depending on the definition of acceptance probability. In any case, the only way to examine the effect of the absolute temperature, like the other two parameters, is to perform experimental analysis.

Lastly, cooling rate controls how fast the temperature decreases. Since the probability of accepting a worse (higher RF score) neighbor depends on the current temperature, smaller values for cooling rate will give the algorithm more chances to pick worse neighbors at early iterations. But again, without any experiments it is very difficult to predict what values for cooling rate will yield better final supertrees.

4.4 Our Strategy to Choose Annealing Schedule

If we did not care about running time at all, then finding proper annealing schedule would have been very simple: choose a sufficiently large value for initial temperature, and a very small value for cooling rate, and absolute temperature. However, this is not practical. The goal here is to find proper values for these three parameters of SA algorithm which yield relatively lower RF score and running time simultaneously, in most of the cases. However, this is not a trivial task.

Finding a proper schedule is challenging in practice for a couple of reasons. First, even with restricting the possible values for each parameter, trying out all possible schedules is prohibitively long process. Second, although there are some schedules that outperform some other schedules, there is usually no single schedule that "wins" all the cases. Finally, there is usually a trade off between the RF score and running time, i.e. the longer the running time, the higher the probability of the algorithm to reach a better local optimum.

In order to find a proper schedule, we started by finding proper values for initial temperature. This is because, as formula 4.1 suggests, the probability of accepting worse neighbors, depends on the temperature, and the difference between RF scores. We do not have much control on the latter, but we can empirically find proper values for the former which will result in the final better RF scores.

After trying several different schedules with a focus on the impact of the initial temperature, we observed that, having a high initial temperature will almost always lead to converging to a worse RF score than that of the initial supertree. This could be explained by looking at the distribution of RF scores in the SPR neighborhoods. First of all, there are usually less than 1% of the SPR neighbors that have better RF score specially when we are close to a local optimum. This makes the chance of choosing a better neighbor at random very low. Thus, this will let the algorithm choose many worse neighbors at the beginning. Second, it seems that usually there is an upper bound on the variation of RF scores in an SPR neighborhood, i.e. it is very unlikely that a supertree with very high RF score, has a neighbor with much smaller RF score. In other words, with high initial temperature, the algorithm will quickly move to an area of the solution space with high RF scores, and usually it gets stuck there until the temperature cools down. At this point, the algorithm behaves more greedily and only picks supertrees with lower RF scores. But it seems we do not have much chance to find a good local optimum by random SPR moves from a supertree with that high RF score.

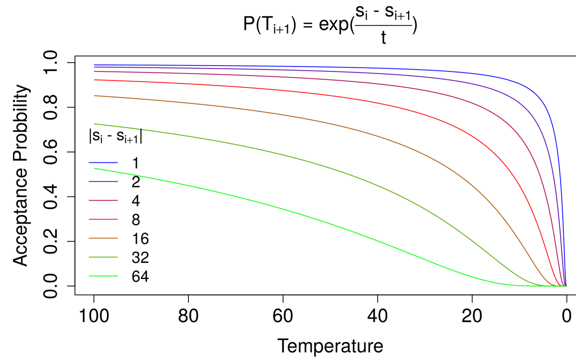


Figure 4.1: Acceptance Probability function for when T_{i+1} has higher RF score than T_i, i.e. $s_{i+1} > s_i$.

Figure 4.1 shows how the acceptance probability changes as a function of temperature when the new neighbor has a worse score, and the difference between current score and new neighbor's score, $|s_i - s_{i+1}|$, is 1, 2, 4, 8, 16, 32, or 64. Note there is nothing special about these numbers, and we just chose them for the sake of illustration. We can see that at a temperature as high as 100, the chance of accepting a worse neighbor with an RF score which is 64 higher than the current RF score is about 50%, which is very high. Only picking a few of such neighbors, will cause the algorithm quickly end up in a supertree with much higher RF score. Given the huge size of the solution space, and the fact that most of the SPR neighbors have worse RF scores when we are close to the local optimum, we can expect that the algorithm is not that likely to be able to find a good local optimum from a very "bad" solution. Thus we have to stick to low initial temperatures, i.e. temperatures less than 3. With such initial temperatures we can set the cooling rate and absolute temperatures to the values that lets the algorithm runs enough moves so that it converges to a relatively good RF score as fast as possible.

In sum, here is our strategy to pick the schedule. We first tested several initial temperatures with very small cooling rate and absolute temperature so that we have some estimate

of the best RF score can be achieved given that initial temperature in each case. After finding some relatively good candidate values for initial temperature, we tried several values for cooling rate and absolute temperature such that the final RF score stays almost the same while running time is minimized, i.e. we tried to find some values for cooling rate that makes convergence faster, and we adjusted absolute value so that the algorithm doesn't waste time after convergence. Overall the following ranges of values seem to be working well in this problem:

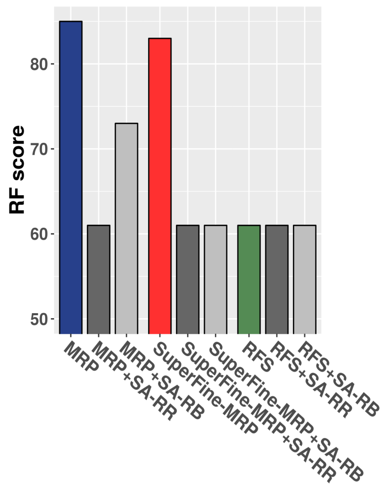
- Initial temperature: 0.01-2
- Cooling Rate: 0.0001-0.001
- Absolute temperature: $\leq$0.0001

4.5 Experimental Results, SA-RR vs SA-RB

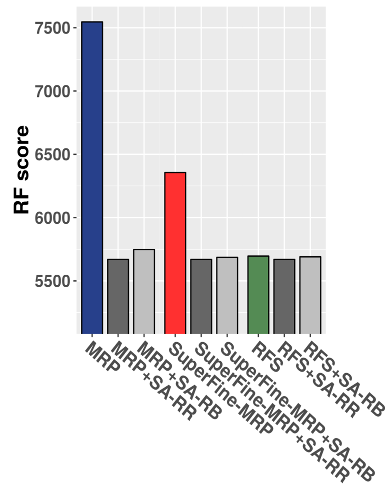
In this section, we compare the two methods to generate a random neighbor in SA algorithm: RR and RB. We refer to theses algorithms by *SA-RR* and *SA-RB*, respectively. Because of the randomness involved in SA algorithm, each algorithm has been run ten times in each case, and the best RF score is reported below. Further, for each version of the SA algorithm, we first obtained proper annealing schedules as described above. The final schedules we used for SA-RR and SA-RB are as follow.

Table 4.1: Annealing schedules for SA algorithm

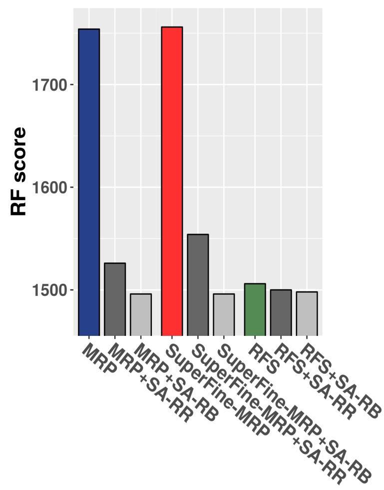
Parameter	**SA-RB**	**SA-RR**
initial temperature	0.1	0.01
cooling rate	0.001	0.0001
absolute temperature	0.00001	0.0001



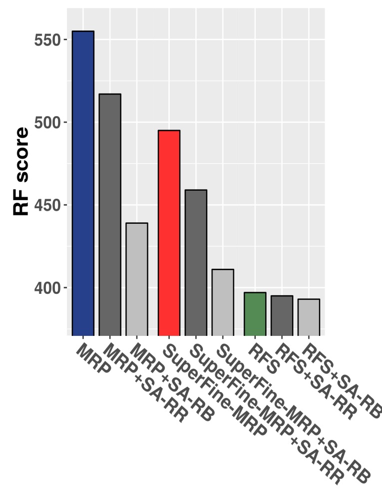
(a) seabirds



(b) PM



(c) marsupials



(d) THPL

Figure 4.2: RF score comparison of SA-RB and SA-RR on four datasets given the same initial supertrees, MRP, SuperFine-MRP, and RFS.

Figure 4.2 shows RF score comparison between SA-RR and SA-RB on four datasets. As we can see, both SA-RR and SA-RB are able to improve the RF score of the given supertree for all the cases, on all datasets. This improvement is more pronounced when MRP or SuperFine-MRP are used as initial supertree. This is because these supertrees has higher RF score, and hence there is more space to improve their accuracy. Further, both SA-RR and SA-RB are able to make some improvements over RFS which is already a very good local optimum for RF supertree problem. However, because of the large scale of the numbers, this improvement is not that bold. In the next section, we will present the results excluding MRP and SuperFine-MRP to be able to take a closer look at the performance of RFS, ER-RFS, SA-RR, and SA-RB

On the two datasets with smaller number of taxa, seabirds and PM, SA-RR performs very well, and is able to get the exact same RF scores on these two datasets with all different initial supertrees. Notably on placental mammals, SA-RR gets a RF score of 5670 which is the best score among all algorithms (compare it with RF score of RFS, 5696, and RFS+ER, 5690). This could provide evidence that SA algorithm has the potential to deal with variations in the initial supertree. However, the performance of the SA-RR algorithm decreases as the number of taxa increases. On THPL and marsupials, although SA-RR algorithm is able to improve the RF score of the initial supertree, the RF score increases as the RF score of the initial supertree increases. This could be because of the larger solution space of these two datasets, and the fact that SA-RR uses only random SPR moves. As we mentioned earlier, we have observed that the portion of the neighbors with better RF score decreases dramatically as we are getting closer to the local optimums. Therefore, on these two datasets probably performing *random* SPR moves is not enough to manage to a "good" local optimum.

SA-RB, on the other hand, seems to have the opposite behavior of the SA-RR. On seabirds and PM datasets, although SA-RB is able to improve the RF score of the ini-

tial supertree in all the cases, there are some variations on the final RF scores obtained. Specially, it has a poor performance on seabirds when MRP is used as initial supertree. This is a strange behavior. We ran the algorithm several times, and we never got a better RF score. Remember that ER-RFS algorithm had relatively poor performance on seabirds when MRP used as initial supertree, as well. This might be because of a special characteristic in that area of the solution space which prohibits RR operations to be able to let the algorithm escape from that hill in the solution space.

However, SA-RB has much better performance on the two datasets with larger number of taxa. Notably on marsupials, it is able to obtain very close RF scores no matter what the initial supertree is. On THPL, however, the RF score seems to have a direct correlation with the RF score of the initial supertree. The same reasoning as for poor performance of SA-RR on this datasets can be used to justified this behavior of SA-RB. THPL dataset, is probably the most difficult dataset among these four datasets. It has the highest number of taxa while it contains only 19 source trees. The poor performance of our algorithms on this dataset can be associate with lack of enough "information" in this dataset.

This could be because of the larger solution space of these two datasets, and the fact that SA-RR uses only random SPR moves. As we mentioned earlier, we have observed that the portion of the neighbors with better RF score decreases dramatically as we are getting closer to the local optimums. Therefore, on these two datasets probably performing *random* SPR moves is not enough to manage to a "good" local optimum.

Overall, although SA-RR seems to outperform SA-RB on smaller datasets, SA-RB is the winner on larger datasets. In simple words, RR operation *explores* larger number of neighbors in the solution space. On the other hand, RB operation is more greedy, and tries to *exploit* more what we have at hand. There is a trade off here. On larger datasets, we might expect that being more greedy could be more effective in obtaining better local optimums. This intuition could justify the different behaviors of the SA-RR and SA-RB

algorithms.

Formally speaking, generally we expect that RB to obtain lower RF scores than RR in average given the same supertree. Further, the size of SPR neighborhood is $\mathcal{O}(n^2)$, where n is the number of taxa in the supertree. Hence, given that we have used fixed number of iterations (same schedule) on all datasets, we can expect that RR's chance to pick a random neighbor with better RF score decreases as n increases over the whole run of the algorithm. Thus, SA-RR might not be as successful as SA-RB on larger datasets.

The time complexity of the SA-RR algorithm depends on three parameters, the number of taxa on the supertree n, the number of source trees k, and the number of SA iterations performed. The latter is merely determined by the parameters of the SA algorithm. Lets assume the number of iterations performed in SA algorithm is T. It takes $\mathcal{O}(kn)$ time to calculate RF score in each iteration. Hence, the time complexity of the SA-RR algorithm is $\mathcal{O}(knT)$. On the other hand, the time complexity of the SA-RB algorithm has one more component: in each iteration, we make a call to RFS algorithm to solve the restricted local optimum problem. This takes an additional time of $\mathcal{O}(kn)$. Therefore, the time complexity of the SA-RB algorithm is $\mathcal{O}(k^2n^2T)$.

Note, however, that the number of iterations in SA-RR and SA-RB does not need to be the same. In fact, in SA-RB algorithm, we need much fewer number of iterations to converge to a local optimum in compare to the SA-RR algorithm. As we will demonstrate later, SA-RB algorithm requires much less time than SA-RR. But, we save the comparison of the running time and the parsimony score between SA-RR and SA-RB for the next section since we are going to make a comprehensive comparison of all the algorithms together in the next section. The next section, summarizes all the results, and concludes our work.

5. SUMMARY AND CONCLUSIONS

5.1 Overall Comparison

In this section, we compare all of the algorithms together. Among different versions of ER algorithm (with different initial supertrees), RFS+ER and SuperFine-MRP+ER had overall better performance. On the other hand, among different versions of SA-RR and SA-RB algorithms (with different initial supertrees), RFS+SA-RR and RFS+SA-RB had a better performance. Therefore, we only compare them with other three algorithms: MRP, SuperFine-MRP, and RFS. Note that when comparing RF scores, we are going to exclude MRP and SuperFine-MRP since they have much higher RF scores in all the datasets. This will help to take a closer look at the difference among other algorithms.

Figure 5.1 shows RF score comparison of RFS, SuperFine-MRP+ER, RFS+ER, RFS+SA-RR, and RFS+SA-RB. On seabirds, all the algorithms get the same score of 61. Although not provable, this might suggest that 61 is the actual global optimum on this dataset.

With an exception of SuperFine-MRP+ER on THPL, all of our four algorithms, SuperFine-MRP+ER, RFS+ER, RFS+SA-RR, and RFS+SA-RB, are able to get better RF scores than RFS algorithm which is the best existing algorithm to minimize RF distance! On PM dataset, RFS+SA-RR is able to get a considerably better RF score. This can be justified by the fact that this dataset is probably the easiest dataset with the smallest number taxa, and the largest number of source trees. Further, RR operation provides the SA-RR algorithm the opportunity to better explore the relatively small solution space, and take advantage of the abundant information available in the source trees.

In Figure 5.2, we can see that SuperFine-MRP gets the best parsimony score on all datasets. MRP has also good parsimony score in most of the cases. However, on seabirds dataset, MRP has the highest parsimony score among the supertrees. Interestingly, SuperFine-

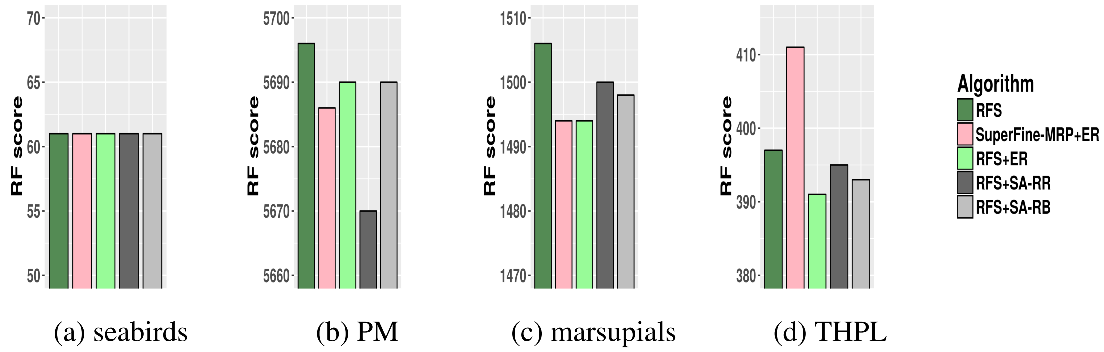


Figure 5.1: RF score comparison of RFS, SuperFine-MRP+ER, RFS+ER, RFS+SA-RR, and RFS+SA-RB.

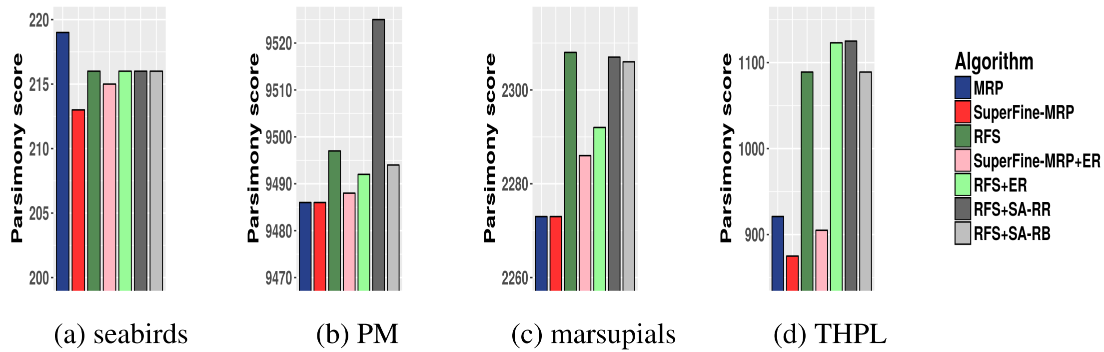


Figure 5.2: Parsimony score comparison of MRP, SuperFine-MRP, RFS, SuperFine-MRP+ER, RFS+ER, RFS+SA-RR, and RFS+SA-RB.

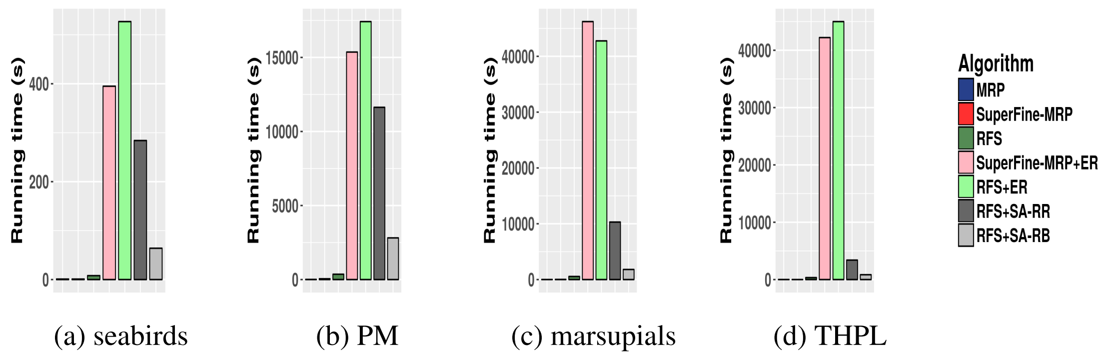


Figure 5.3: Running time comparison of MRP, SuperFine-MRP, RFS, SuperFine-MRP+ER, RFS+SA-RR, and RFS+SA-RB.

MRP+ER has better parsimony score than MRP.

SuperFine-MRP+ER has lower parsimony score in compare to RFS+ER, RFS+SA-RR, and RFS+SA-RB on all datasets. This can be associated with the low parsimony score of the initial supertree used (SuperFine-MRP). Although one might expect that optimizing RF score and parsimony score should be correlated, this is not always the case. Specifically, we have two counter examples in our results. First, on PM dataset, SA-RR has the lowest RF score while it has the highest parsimony score. Second, SuperFine-MRP+ER has higher RF score than RFS, RFS+ER, RFS+SA-RR, and RFS+SA_RB. However, it has lower parsimony score in compare to them.

Nevertheless, there are actually cases where we have relatively low RF score and parsimony score, simultaneously. For example, except on THPL, SuperFine-MRP+ER has both relatively low RF scores and parsimony scores. This is quite interesting observation! Given the two above counter examples, it is hard to make a definite statement about this observation. But, it is quite reasonable to make the following recommendation. When using branch swapping hill-climbing algorithms, it is always a good idea to try several different initial supertrees. Specifically if the goal is to minimize RF score, we should always try both supertrees that are local optimum with regard to RF score, and supertrees that are local optimum with regard to parsimony score. This will give us the opportunity to compare them, and pick the one that suits our needs for that specific dataset.

Finally, in Figure 5.3, we can observe that MRP, SuperFine-MRP, and RFS have all much better running times in compare to our algorithms. ER-RFS is the most expensive algorithm. As we mentioned earlier, there are several opportunities to make optimizations in ER algorithm to make it faster. But, we cannot avoid the huge running time of this algorithm because it requires to perform many hill-climbing searches each of which could take many iterations to reach a local optimum. On the other hand, SA-RR is able to converge to a local optimum with a running time that is up to 85% shorter than that of ER

algorithm! Also, SA-RB requires up to 98% shorter amount of time to converge than the ER algorithm! This improvement in running time is more pronounced on larger datasets (number of taxa). We can see that the running time of the SA-RB gets very close to that of MRP, SuperFine-MRP, and RFS as the number of taxa increases. This is because of the fixed cooling schedule used for this algorithm.

5.2 Conclusion

In this work, we focused on the RF supertree problem. Our goal was to improve RF score of the best existing algorithms. we proposed the Edge Ratchet technique to deal with local optimum problem. Further, we developed our hill-climbing algorithm equipped with this technique, and we showed that this algorithm is able to improve the RF score of the well-known, existing algorithms. Further, we developed two versions of Simulated Annealing algorithm for RF supertree problem. Our results show that all of these algorithms are able to improve the RF score of the initial supertree considerably. More specifically, we showed that these algorithms are able to make improvements in RF score of the three well-known supertree algorithms: MRP, SuperFine-MRP, and RFS.

We had several interesting observations in our results. First of all, different datasets have different characteristics that could potentially make them easier or more difficult for the algorithm at hand. These differences could cause variations in the final results. There are two main parameters that could play a key role with this regard: the number of taxa, and the number of source trees. The size of the solution space increases dramatically as the number of taxa increases (double factorial). This can potentially make the dataset harder for the hill-climbing algorithm to solve. On the other hand, the increase in the number source trees is usually helpful for the algorithm. This is because, having larger number of trees increases the number of available clusters to the algorithm. Since we expect the source trees to have a lot of agreement with each other, increasing the number of source

trees will increase the portion of non-conflicting clusters, and hence the algorithm can take advantage of them to better handle conflicts. Therefore, we could conclude that the third important property of a dataset is the amount of agreement between source trees. It may not be trivial to measure this property in the datasets. But one possible way is to construct a consensus tree of the source trees, and measure its RF score to the source trees. Overall, the datasets with smaller number taxa and larger number of source trees are easier to solve, and the datasets with larger number taxa and smaller number of source trees are harder to solve.

The second interesting observation is that we could build supertrees with both low RF and parsimony scores. Our results suggest that using our algorithms with an initial supertree with low parsimony score, such as MRP and SuperFine-MRP, is a good idea for this purpose. Generally, when using hill-climbing algorithms or SA algorithm, it is a good idea to try several different initial supertrees.

Finally, for SA algorithm, we empirically obtained proper schedules on these four datasets. However, this schedule may not work well on other datasets. This could be a huge disadvantage for this algorithm because, as we mentioned earlier, obtaining a proper schedule can be a very challenging task. Although we might come up with some general guidelines on how to find such schedule, this task could be quite expensive and probably not practical for very huge datasets.

5.3 Future Work

There are several directions to extend this work and make improvements. First, there are several possibilities to improve the accuracy of the ER algorithm. Second, there are several opportunities to improve the running time of both ER and SA algorithms. Third, as we observed in our results, usually optimizing RF score comes at the cost of an increase in the parsimony score. It is desirable to find ways to reduce this cost. Below, we briefly

describe some ideas for each of these directions.

We believe that there is still some room to improve the accuracy of ER algorithm with regard to RF score. For example, when generating the SPR neighborhood, there are usually more than one best neighbor. In ER algorithm we pick one of them (the first one encountered). However, it could be the case that picking another best neighbor at some iteration results in a final lower RF score. This is specially important at local optimums. There are usually several supertrees with the same RF score of the local optimum (for example more than 20 in seabirds data set). Although the supertree picked by ER algorithm happens to be a local optimum, the other trees with the same score could have better SPR neighbors.

We performed some initial experiments to investigate above hypothesis. For these experiments we just performed one hill-climbing search algorithm without any ER iterations. We first tried an exhaustive approach. We changed the greedy algorithm (without ER) to consider all best neighbors found at a SPR neighborhood and then we started a new search from each of those neighbors the same way. However, this makes the algorithm very time consuming and impractical, since the branching factor (number of greedy searches) usually increases exponentially because each of the neighbors with best RF score might have several neighbors with best score, and so on. To deal with this problem, we made a change in the algorithm. Suppose we are at some iteration i. All best SPR neighbors are picked. But instead of starting whole new search from each of those, we looked at the best score can be achieved from each of them in one SPR neighborhood. Then, the one with best SPR neighbor is picked, and those two consecutive SPR moves that yields the best score are performed to get the algorithm to the next iteration. In this manner, at each iteration, we actually move 2 SPR neighborhoods away from current supertree. This algorithm obtained slightly better scores than regular greedy on some datasets. We also made some experiments using this modified version of greedy algorithm in ER algorithm. Although

in few cases it yielded a better score, but most of the runs did not finish within 48 hour time limit.

There are several ways to improve the running time of the ER-RFS algorithm. First, there are a lot of optimization opportunities in the algorithm implementation to improve running time. For example, two of the most expensive operations in the algorithm are calculating LCA, and calculating RF score of the newly selected neighbor. To find LCA, we used a simple $O(n)$ algorithm. However, Bansal et al. used a more complicated algorithm which is still $O(n)$, but with much lower coefficient/constant. Finding LCA is probably the most frequent expensive task, and small improvements in its algorithm could potentially has a huge impact on the overall running time. Second, whenever a new SPR neighbor is selected, we calculate its RF score by calculating the RF distance between that tree and each of the source trees. However, we should be able to calculate this score more efficiently by taking advantage from knowing the RF score of the previous supertree, and α β values. Third, we could parallelize some parts of the algorithm. For example, in current implementation, we calculate RF score of the current supertree in search by calculating its RF distance to each source trees. This part of the algorithm can be parallelized. Finding LCA of the internal nodes of the source trees can also be parallelized. Last but not least, we used 50 ratchet iterations. However, when looking at the RF score changes across these iterations, we observed that in most of the cases, the best RF score is obtained in the first 10 iterations. This means that, with ER algorithm, we are able to obtain similar results with up to 80% faster than the running time reported.

For SA-RB algorithm, the same above mentioned approaches can be used to make SA algorithm faster so that we are able to perform more number of iterations. To improve the accuracy of the SA algorithm, one interesting idea is to use dynamic values for parameters to imitate the behavior of the ER technique. For example, we can start with some configuration and let the algorithm run for sum number of iterations, then we can periodically

increase the initial temperature to make some turbulence in the search before convergence. Or we can make this turbulence whenever the algorithm does not make any improvements for a pre-specified number of iterations.

Finally, our results showed that obtaining a low RF score usually comes at the cost of high parsimony score. On the other hand, we observed that it is possible to obtain relatively low RF and parsimony scores simultaneously (SuperFine-MRP+ER). One interesting idea for further investigation is to change the objective function of the ER algorithm so that it minimizes RF score and parsimony score simultaneously. For example, we can simply define the objective function to be the sum of RF score and parsimony score, and then we can use the exact same hill-climbing algorithm with ER iterations. However, because of the different scales of the RF and parsimony scores, this simple objective function might not be the best we can do. It might be the case that we need to give each of them a different weight in the objective function. For instance, we can empirically obtain such proper weights by trying some different combination of values for weights, and comparing the results.

REFERENCES

[1] D. M. Hillis, T. A. Heath, and K. S. John, "Analysis and visualization of tree space," *Systematic biology*, vol. 54, no. 3, pp. 471–482, 2005.

[2] C. Mora, D. P. Tittensor, S. Adl, A. G. Simpson, and B. Worm, "How many species are there on earth and in the ocean?," *PLoS Biol*, vol. 9, no. 8, p. e1001127, 2011.

[3] F. McMorris and M. A. Steel, "The complexity of the median procedure for binary trees," in *New Approaches in Classification and Data Analysis*, pp. 136–140, Springer, 1994.

[4] M. S. Bansal, J. G. Burleigh, O. Eulenstein, and D. Fernández-Baca, "Robinson-foulds supertrees," *Algorithms for Molecular Biology*, vol. 5, no. 1, p. 1, 2010.

[5] K. C. Nixon, "The parsimony ratchet, a new method for rapid parsimony analysis," *Cladistics*, vol. 15, no. 4, pp. 407–414, 1999.

[6] B. R. Baum, "Combining trees as a way of combining data sets for phylogenetic inference, and the desirability of combining gene trees," *Taxon*, pp. 3–10, 1992.

[7] M. A. Ragan, "Phylogenetic inference based on matrix representation of trees," *Molecular phylogenetics and evolution*, vol. 1, no. 1, pp. 53–58, 1992.

[8] L. R. Foulds and R. L. Graham, "The steiner problem in phylogeny is np-complete," *Advances in Applied Mathematics*, vol. 3, no. 1, pp. 43–49, 1982.

[9] D. L. Swofford, "Paup*: Phylogenetic analysis using parsimony (and other methods) 4.0. b5," 2001.

[10] P. A. Goloboff, J. S. Farris, and K. C. Nixon, "Tnt, a free program for phylogenetic analysis," *Cladistics*, vol. 24, no. 5, pp. 774–786, 2008.

[11] M. S. Swenson, R. Suri, C. R. Linder, and T. Warnow, "Superfine: fast and accurate supertree estimation," *Systematic biology*, vol. 61, no. 2, pp. 214–227, 2012.

[12] D. T. Neves, T. Warnow, J. L. Sobral, and K. Pingali, "Parallelizing superfine," in *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, pp. 1361–1367, ACM, 2012.

[13] D. Chen, L. Diao, O. Eulenstein, D. Fernández-Baca, and M. Sanderson, "Flipping: a supertree construction method," *DIMACS series in discrete mathematics and theoretical computer science*, vol. 61, pp. 135–162, 2003.

[14] D. Chen, O. Eulenstein, D. Fernandez-Baca, and M. Sanderson, "Minimum-flip supertrees: Complexity and algorithms," *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, vol. 3, no. 2, pp. 165–173, 2006.

[15] D. Chen, O. Eulenstein, D. Fernández-Baca, and J. Burleigh, "Improved heuristics for minimum-flip supertree construction," *Evolutionary Bioinformatics*, vol. 2, 2006.

[16] A. V. Aho, Y. Sagiv, T. G. Szymanski, and J. D. Ullman, "Inferring a tree from lowest common ancestors with an application to the optimization of relational expressions," *SIAM Journal on Computing*, vol. 10, no. 3, pp. 405–421, 1981.

[17] C. Semple and M. Steel, "A supertree method for rooted trees," *Discrete Applied Mathematics*, vol. 105, no. 1, pp. 147–158, 2000.

[18] R. D. Page, "Modified mincut supertrees," in *International Workshop on Algorithms in Bioinformatics*, pp. 537–551, Springer, 2002.

[19] J. B. Orlin, "Max flows in o (nm) time, or better," in *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pp. 765–774, ACM, 2013.

[20] M. Brinkmeyer, T. Griebel, and S. Böcker, "Polynomial supertree methods revisited," *Advances in bioinformatics*, vol. 2011, 2011.

[21] C. Whidden, N. Zeh, and R. G. Beiko, “Supertrees based on the subtree prune-and-regraft distance,” *Systematic biology*, p. syu023, 2014.

[22] S.-J. Sul, G. Brammer, and T. L. Williams, “Efficiently computing arbitrarily-sized robinson-foulds distance matrices,” in *International Workshop on Algorithms in Bioinformatics*, pp. 123–134, Springer, 2008.

[23] H. T. Lin, J. G. Burleigh, and O. Eulenstein, “Triplet supertree heuristics for the tree of life,” *BMC Bioinformatics*, vol. 10, pp. S8 – S8, 2009.

[24] M. Bordewich and C. Semple, “On the computational complexity of the rooted subtree prune and regraft distance,” *Annals of combinatorics*, vol. 8, no. 4, pp. 409–423, 2005.

[25] G. Hickey, F. Dehne, A. Rau-Chaplin, and C. Blouin, “Spr distance computation for unrooted trees,” *Evolutionary Bioinformatics*, vol. 4, 2008.

[26] C. Whidden, R. G. Beiko, and N. Zeh, “Fixed-parameter algorithms for maximum agreement forests,” *SIAM Journal on Computing*, vol. 42, no. 4, pp. 1431–1466, 2013.

[27] S. Linz and C. Semple, “A cluster reduction for computing the subtree distance between phylogenies,” *Annals of Combinatorics*, vol. 15, no. 3, pp. 465–484, 2011.

[28] M. A. Bender and M. Farach-Colton, “The lca problem revisited,” in *Latin American Symposium on Theoretical Informatics*, pp. 88–94, Springer, 2000.

[29] M. Kennedy, R. D. Page, and R. Prum, “Seabird supertrees: combining partial estimates of procellariiform phylogeny,” *The Auk*, vol. 119, no. 1, pp. 88–108, 2002.

[30] R. M. Beck, O. R. Bininda-Emonds, M. Cardillo, F.-G. Liu, and A. Purvis, “A higher-level mrp supertree of placental mammals,” *BMC Evolutionary Biology*, vol. 6, no. 1, p. 1, 2006.

[31] M. Cardillo, O. R. Bininda-Emonds, E. Boakes, and A. Purvis, "A species-level phylogenetic supertree of marsupials," *Journal of Zoology*, vol. 264, no. 01, pp. 11–31, 2004.

[32] M. F. Wojciechowski, M. J. Sanderson, K. P. Steele, and A. Liston, "Molecular phylogeny of the "temperate herbaceous tribes" of papilionoid legumes: a supertree approach," *Advances in legume systematics*, vol. 9, pp. 277–298, 2000.

[33] M. S. Swenson, F. Barbançon, T. Warnow, and C. R. Linder, "A simulation study comparing supertree and combined analysis methods using smidgen," *Algorithms for Molecular Biology*, vol. 5, no. 1, p. 1, 2010.

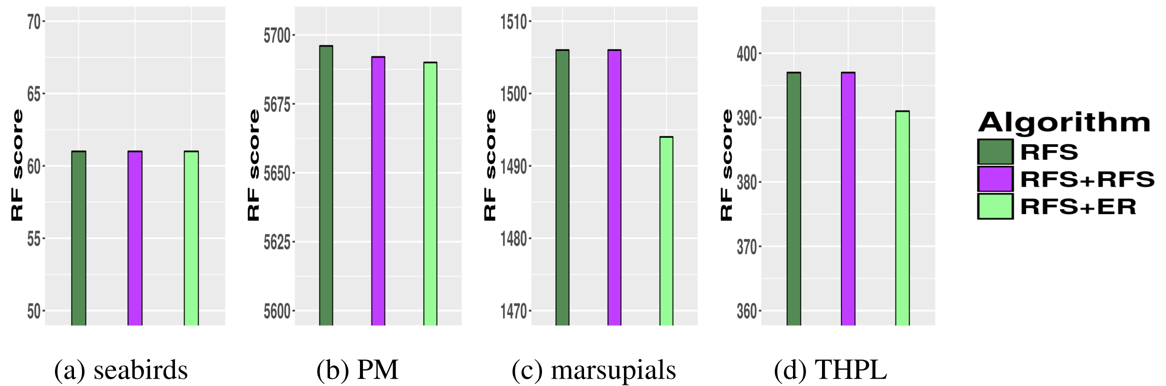
[34] M. J. Sanderson, "r8s: inferring absolute rates of molecular evolution and divergence times in the absence of a molecular clock," *Bioinformatics*, vol. 19, no. 2, pp. 301–302, 2003.

APPENDIX A

ER TECHNIQUE EFFECTIVENESS

Our first contribution in this work was to propose Edge Ratchet technique, and develop a hill-climbing algorithm, ER-RFS, that utilizes this technique to handle the local optimum problem. In our results, we showed that this algorithm is capable of improving RF score of the well-known, existing algorithms. Specially, we showed that ER-RFS algorithm is able to improve RF score of the RFS algorithm which obtains high quality supertrees with regard to RF score. In other words, when we use RFS algorithm as initial supertree of the ER-RFS algorithm, we start the search from a very good local optimum. Because of the fact that ER-RFS algorithm is able to improve the RF score of this supertree on all datasets, we claimed that Edge Ratchet technique is effective in dealing with local optimum problem. However, it is not clear that achieving this improvement in the RF score is actually the result of using ER technique. For example, one might wonder whether RFS algorithm is able to improve itself, i.e. running RFS algorithm on some dataset, and then use the final supertree as the initial supterree of another run of RFS algorithm. In this appendix, we provide some supplementary results to investigate such questions.

Note that the main difference between ER-RFS algorithm and RFS algorithm is that ER-RFS uses Edge Ratchet technique to handle the local optimum, while RFS uses a different ratchet search where each source tree is treated as a character. Therefore, we can investigate the contribution of the ER technique in ER-RFS algorithm by the following experiment. We feed both ER-RFS algorithm and RFS algorithm with the same initial supertrees, and compare the RF score of the final supertrees. We compare RFS and ER-RFS algorithms when they both start from the same initial supertree: RFS, MRP, SuperFine-MRP. Remember that RFS algorithm, by default, generates a greedy taxon addition initial



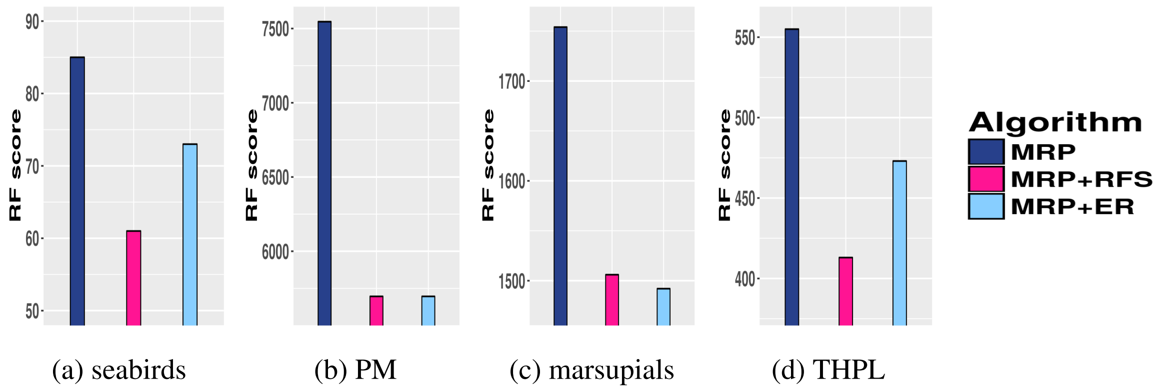
(a) seabirds (b) PM (c) marsupials (d) THPL

Figure A.1: RF score comparison of RFS+RFS and RFS+ER.

supertree.

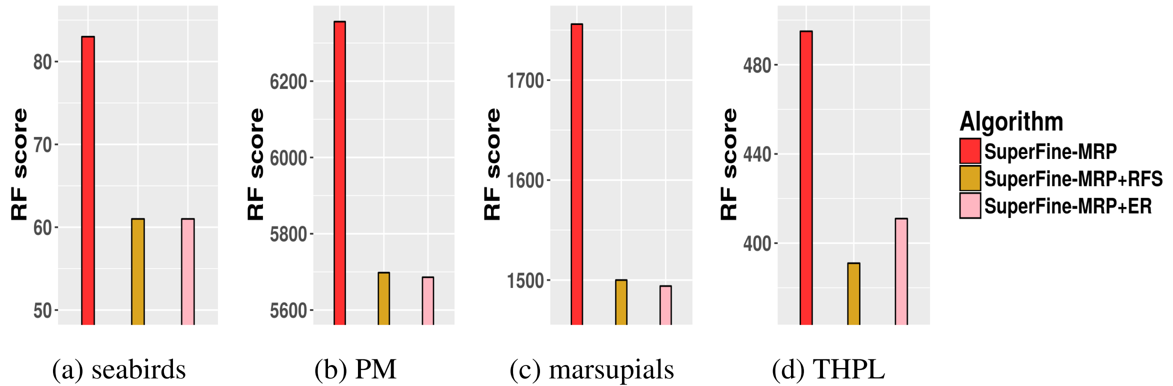
Note that we are mainly interested in the comparison between RFS+RFS and RFS+ER since the initial supertree is already a local optimum with very low RF score. As we can see in Figure A.1, RFS+RFS algorithm obtains the same RF score on three of the datasets with no improvement over the initial supertree. However, RFS+ER algorithm improves the RF score on all datasets (on seabirds, they all obtain the same RF score). On PM dataset, RFS+RFS improves the RFS score slightly, however, RFS+ER algorithm is able to find an even better RF score. This experiment provides more evidence on the effectiveness of the Edge Ratchet technique to handle the local optimum problem.

When the common initial supertree is not a local optimum, however, there is almost a tie between RFS and and ER-RFS algorithm. When using MRP as initial supertree, both MRP+RFS and MRP+ER get the same score on PM dataset. However, MRP+RFS reaches a better local optimum on seabirds and THPL while MRP+ER finds a better local optimum on marsupials. On the other hand, SuperFine-MRP+ER gets a lower RF score on PM and marsupials dataset than SuperFine-MRP+RFS. SuperFine-MRP+ER performs better on THPL by obtaining a better RF score. There is a tie on seabirds dataset, and both get the same final score.



(a) seabirds (b) PM (c) marsupials (d) THPL

Figure A.2: RF score comparison of MRP+RFS and MRP+ER.



(a) seabirds (b) PM (c) marsupials (d) THPL

Figure A.3: RF score comparison of SuperFine-MRP+RFS and SuperFine-MRP+ER.

The reason for this mix behavior must be related to how the supertrees are distributed in the solution space for each dataset, and the specific supertree each algorithm pick on their way to their final solution. Note that MRP and SuperFine-MRP both have high RF scores, and they are probably not a local optimum. Therefore, ER-RFS and RFS algorithms both traverse a long path, probably including several local optimums, in the solution space to reach their final supertrees. On the other hand, when we are in a local optimum, there is no way to figure out which neighbor will actually result in the best final supertree. Sometimes, picking two neighbors with exact same score could result in two completely different final

supertrees. Further, given the random steps involved in both algorithms, they both could potentially pick different neighbors after reaching a local optimum, and hence obtain a different supertree. Therefore, it is not that surprising to observe such mix behavior when they start from a supertree with high RF score.