Gene Tree Improvement Tool: Tree Fix

May15-10

Ben Streit: Team leader
Paul Leichty: Communication Leader
Cole Poffenberger: Key Concept Holder
Ian Ray: Webmaster

Advisor: Ruchi Chaudhary
Client: Gordon Burleigh

Table Of Contents

1 Abstract

This document covers the work done by team May15-10 over the past two semesters working on the Gene Tree Improvement Tool project (Treefix). It details general information regarding Treefix, the requirements our client laid out, and an in depth look at our design and testing process. This document also includes a brief tutorial on how to use Treefix, as well as relevant code sections from our work implementing the two cost algorithms created for our client.

2 Definition of Terms

TreeFix - TreeFix is a program for improving gene tree reconstructions. It uses a test statistic for likelihood equivalence, and a species tree aware (reconciliation) cost function to produce an optimal gene tree topology.

Cost Algorithm - An algorithm used by TreeFix to find the total cost of reconciling a gene and species tree.

(Tree) Reconciliation - a computation that produces a reconciliation cost between two trees. It bases this computation on its inputs, which are the two trees specified.

DupLoss - or Duplication Loss and is a biological cost method.

RF - or Robinson-Foulds distance is a mathematical cost, computed by counting number of bipartitions present in one tree but not in the other. RF distance is defined for the tree pair on the identical leaf label set (NOT multi-set).

MulRF - Multiple Robinson Folds Cost which is a mathematical cost method.

Deep Coalescence - Biological cost method that uses edges to find the distance cost.

Phylogenetic - Relationship between organisms.

Gene Tree - Represents the evolutionary history of the genes in the study.

Optimal Gene Tree - The specific gene tree, in a set topologies, that is statistically most likely to represent the actual phylogenetic relationship.

Species Tree - Represents the genealogy of a group of one or more populations.

Tree Topology - A Tree, but in relation to any specific permutation of its branches and nodes, given a set of leaves.

RAxML - A tool written in C which produces randomized gene tree topologies.

3 Introduction

3.1 General Background

TreeFix is an open source MIT project currently used by biologists to accurately predict optimal gene trees. The program produces an optimal gene tree by randomly generating a forest of different gene tree topologies, and then comparing an intelligently selected sample of these generated trees with a related species tree. Specifically, It seeks the minimal cost producible from reconciling the species tree with each of the sample gene trees. The sample is created by selecting only those gene tree topologies which implicitly possess a statistical likelihood (of existence) above a certain threshold.

Treefix is attractive to biologists for its low resource-consumption and extensible design. On the topic of design, for instance, the cost-function-module operates through an *interface*, meaning that cost calculations are interchangeable; new calculation methods simply need to be implemented. This makes treefix a potentially diverse tool, if given a wide set of cost modules. Currently, TreeFix only uses a single biological cost, called the Duplication-Loss (DupLoss). This is only one of many cost models that could, in theory, be used to find the optimal gene tree. Furthermore, the DupLoss cost-model is not the right fit for all biological computations. Along this vein, our client came to us needing the implementation of two new cost model options: Multiple Robinson-Folds, and Deep Coalescence.

3.2 Technical Problem

TreeFix is currently created to only be able to handle one cost model. This creates the problem of whether a new version of TreeFix would have to be created for every cost model created or if the existing code structure could be adapted to implement them all into one application. Having everything in the same application is ideal, yet for that to be possible the previous version of TreeFix would need to be extended and adapted appropriately.

TreeFix was found to be fairly modular, allowing for additional command line prompts. The additional prompt allows for the user to choose the cost model they desire and it will then calculate the cost using the corresponding algorithm. This allows for easy transitions between costs within the same application.

Next, the actual new cost algorithms needed to be both created and optimized. The ones being implemented for this project are, Multiple Robinson-Folds and Deep Coalescence. This requires some tweaking of the existing modules that were created to be solely DupLoss dependent. This would otherwise slow down the overall run time by stepping through extra steps in order to get more information that would be necessary for all cost models. By rewriting these and creating an optimal algorithm the run time and memory use was able to be kept at a level appropriate for TreeFix.

3.3 Operating Environment

TreeFix runs through Unix-based Operating System, and uses appropriate applications in order to run python through the command line. There was also a python GUI created to help with the creation of scripts but is not a necessity for those who prefer the command line.

3.4 Intended Users and Uses

TreeFix was created for computational biologists in order to provide a fast, lightweight application that can find the optimal gene tree. This was previously a long and slow process and TreeFix has helped change that.

3.5 Assumptions and Limitations

- The optimal tree is found by TreeFix correctly
- TreeFix uses RAxML to create a forest of gene trees. Due to the nature of RAxML, along with the size and complexity of it, it can be assumed that the gene trees output will be binary.
- TreeFix must be fast otherwise it will not be useful to the users

4 Design Requirements

4.1 Design Objectives

- Modularize TreeFix in a way that allows for multiple cost algorithms to be added as they are created
- Allow for user to choose which Cost Algorithm to use as a command line option
- Addition of two new Cost Modules
- Creation of an optional GUI

4.2 Functional Requirements

- To accept Both binary and nonbinary trees
- Can calculate multiple types of cost algorithms

4.3 Design Constraints

- RAxML is a part of the project which our advisor and group decided should be black boxed

4.4 Measurable Milestones

- Understanding how the MulRF algorithm works
- MulRF implementation
- Testing the MulRF algorithm
- Understanding Deep Coalescences
- Deep Coalescences implementation
- Testing Deep Coalescences
- Fixing all known bugs

6 End-Product Description

Improvements to TreeFix include two new cost models and a graphical user interface. The two new cost models include MulRF and Deep Coalescence. In order to implement both of these algorithms into the existing TreeFix program some adjusts need to be made so that users are able to select which cost model they want to run. The graphical user interface allows users to create and run TreeFix scripts. The GUI takes in all the required elements including paths

to the species tree, species map, log file and alignment file, it also allows for the selection of one of the cost models as well as various other option configurations for the program.

7 Approach and Design

There were two different possible approaches that could have been used to implement a new cost model into TreeFix. The first of these options was to replace the current cost model with the new cost model. This would be the most simple way to achieve the end result but also would require a separate application instance for every cost model that gets added.

The second option is to implement a command line parameter that allows for the user to select between which cost model they would like to use. This will require more work overall due to the necessity of modularizing TreeFix in a way that will allow for these additions.

7.1 Technical Approaches

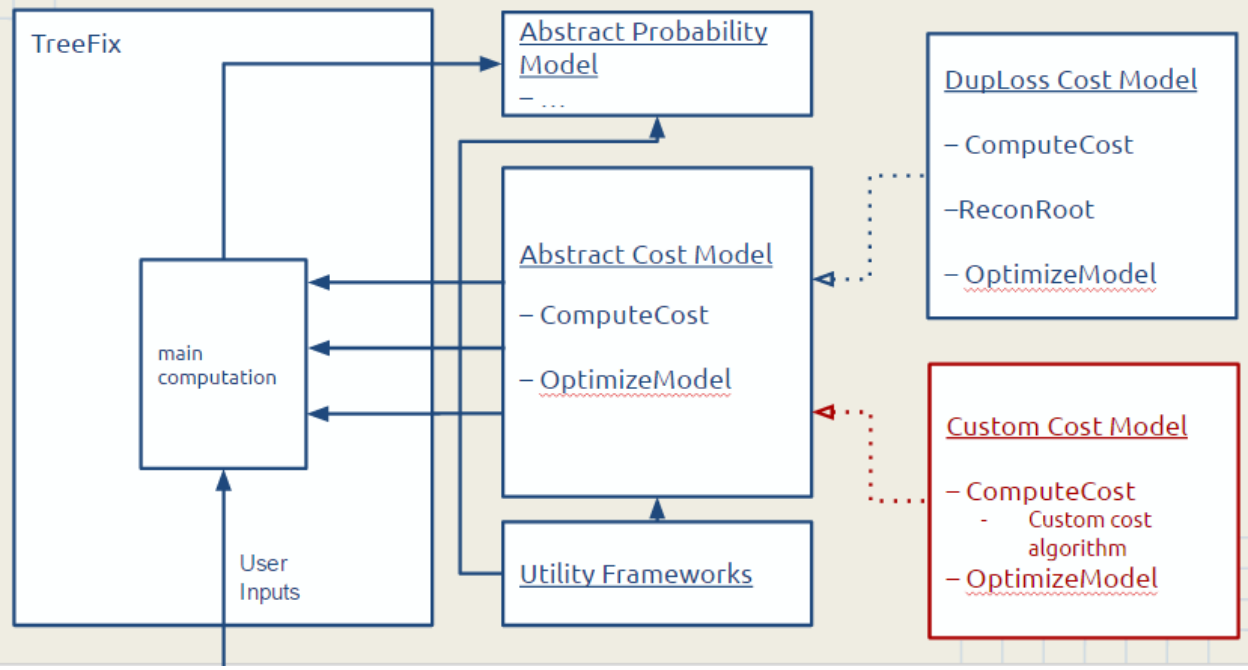
The most practical option to approach the design is to modularize the current version of TreeFix in order to integrate more cost models and allowing the user to select which one they would like to use. This needs to be done by identifying functions that are solely used for the current duploss cost model and which ones will be useful for any cost model that will be used. This process also helped create an understanding of the structure of TreeFix

7.2 Technical Design

It was decided by the group that we would ‘blackbox’ all of unnecessary methods and classes. These are shown as the TreeFix block in the diagram below, some of the things that are included in this block are; random accelerated model(RAxML) which creates a forest of gene trees that are used to find the optimal gene trees, the tree trimming function which gets rid of trees with too low of a statistical likelihood or too high of a cost, and many more. These functions will affect the code which is being creating in no beneficial way so they are forgotten about.

The utility frameworks block in the diagram below would need to be adapted to have methods that are more versatile than the original application. This is due to some of the methods being designed as beneficial towards cost models in general but have overall inefficiencies due to them doing more things than necessary due to duploss needing those. An example of this was the LCA(Least Common Ancestor) method which creates a mapping between the gene and species trees. This method was needed by other cost models yet slowed down other cost models due to it doing unnecessary steps for every cost model except DupLoss. Adapting these types of methods to be more dynamic helped the overall efficiency and structure of the code.

For the abstract cost model block the extended classes are the cost models that TreeFix will be using. They all need to be able to calculate the cost between trees and also be able to optimize the model. There may be other necessities in specific cost models such as ReconRoot in DupLoss cost model but those are case to case. This implementation of cost models allows for simple and quick additions of new cost models provided a correct cost algorithm



7.3 Testing Description

Testing our cost model implementations was a challenge because the majority of the code resides in the compute_cost method of the module and the algorithm for computing the cost is a mathematical formula. Infrastructure within the compute_cost methods had to be built in order to test the algorithms. This allowed us to print the inputs and output to a file every time the method was invoked. The information we collected included the species tree, gene tree, LCA and cost. The first stage of testing was completed by hand. Our group examined the data collected by hand and computed the cost for each species and gene tree and compared the result to that of the program's output. This was a lengthy process because we had to calculate the LCA for each pair of trees and compare the results. The second stage of testing was an automated oracle that parsed our data file and compared the LCA of each tree. We ran 100,00 test cases through the oracle and found that all passed.

7.4 Risks and Risk Management

Risk	**Probability of occurrence**	**Criticality (0-100)**	**Risk Factor**	**Mitigation Strategy**
It may be infeasible to use MulRF as the cost-evaluation method for TreeFix	.05	95	4.75	Research differences between different cost methods to see if it will behave in the way we would like for it to

In order to replace the current TreeFix modules with new modules may require extensive refactoring of current TreeFix code	.7	5	3.5	We will have to evaluate how much code we will be able to re-use and adjust our schedule accordingly
In order to include non-binary trees as inputs, we will have to rewrite the majority of the raxml file	.2	45	9	Run test cases to understand what will have to be changed. If infeasible, find out early enough to talk to client about it and discuss alternatives.
Get the project done on time	.05	90	4.5	Create and follow a schedule and if obstacles appear adjust schedule accordingly

7.5 Recommendation for Continued Work

Since TreeFix only has one objective, to find the optimal gene tree, extensions to this program are limited to additional cost algorithms. Since there are a lot of different cost algorithms and each one serves it own purpose there is no reason why TreeFix couldn't be extended to include more algorithms. It might also be extended to allow for non-binary gene trees.

8 Budget

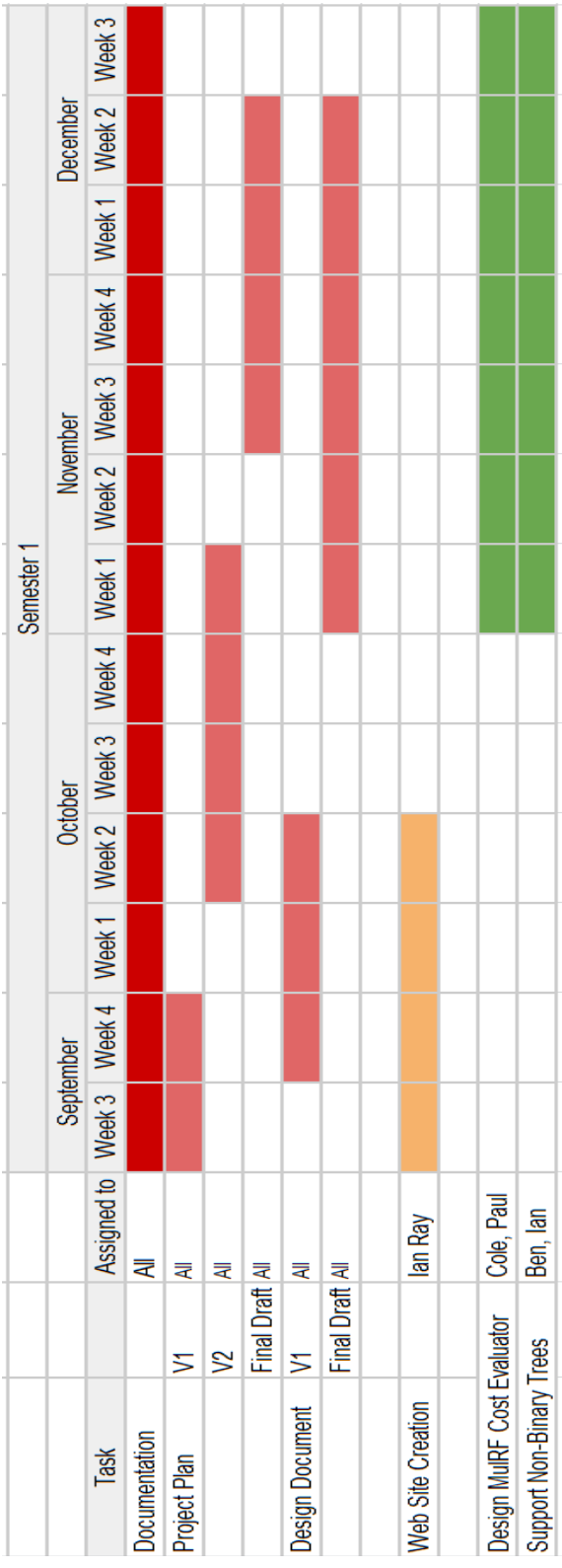
8.1 Financial

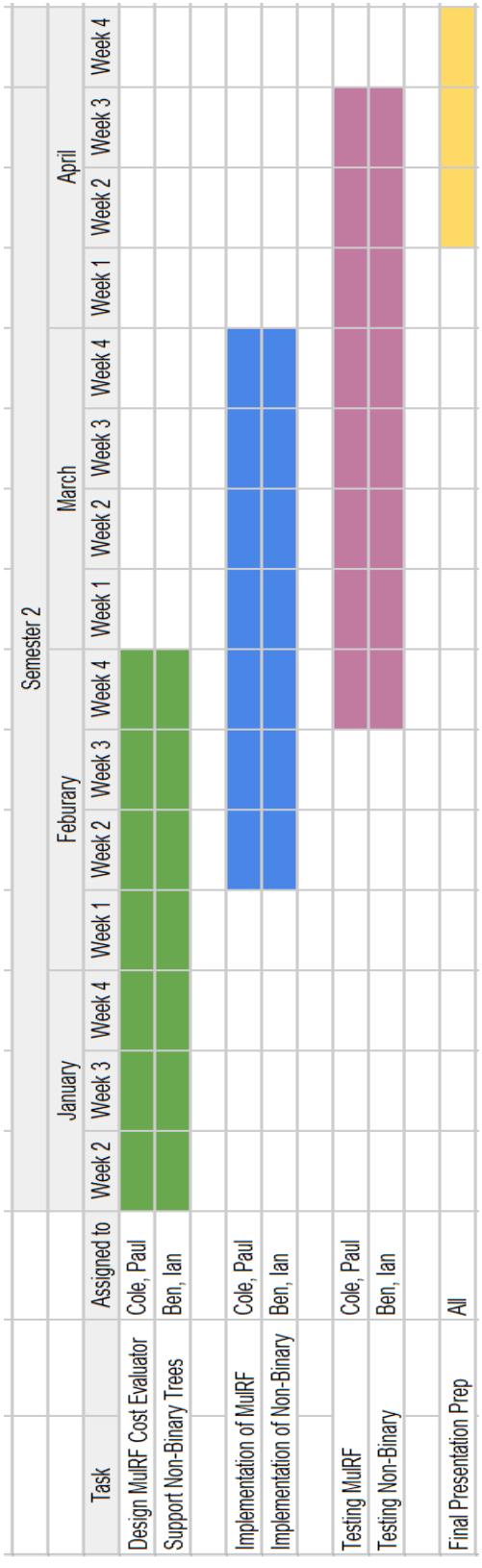
There are no direct costs involved in this project. Treefix is an open source program that can run on even the most limited of machines. This means that the team only had to download the Treefix source code to make changes and run it. This project did not require any additional software or hardware.

8.2 Personnel Effort

In addition to the team members time worked on this project, both our advisor and client helped us complete the project by providing test cases and research articles detailing the specifics about MulRF and Deep Coalescence's algorithms. Our advisor met with us nearly every week and was there to answer any questions we might have and make sure we kept on schedule. We were also in contact with our client in Florida who helped provide some context about how he used Treefix, and what changes he wanted us to implement.

9 Schedule


Semester 1
September
October
November
December
Week 3
Week 4
Week 1
Week 2
Week 3
Week 4
Week 1
Week 2
Week 3
Week 4
Week 1
Week 2
Week 3
Task
Assigned to
Documentation
All
Project Plan
V1
All
V2
All
Final Draft
All
Design Document
V1
All
Final Draft
All
Web Site Creation
Ian Ray
Design MulRF Cost Evaluator
Cole, Paul
Support Non-Binary Trees
Ben, Ian


Semester 2
January
Feburary
March
April
Week 2
Week 3
Week 4
Week 1
Week 2
Week 3
Week 4
Week 1
Week 2
Week 3
Week 4
Week 1
Week 2
Week 3
Week 4
Task
Assigned to
Design MulRF Cost Evaluator
Cole, Paul
Support Non-Binary Trees
Ben, Ian
Implementation of MulRF
Cole, Paul
Implementation of Non-Binary
Ben, Ian
Testing MulRF
Cole, Paul
Testing Non-Binary
Ben, Ian
Final Presentation Prep
All

10 Commercialization

There is currently no monetary income value to be had from this project. The extensions that are being created to TreeFix in this project will be invaluable to our client, Doctor Burleigh, in his research. He also believes that if TreeFix is allowed to have many cost method implemented, it will help biologists everywhere by having a quick and low computer memory resource gene tree mapping software. This will allow for further advances within the field of molecular evolution.

There are other programs that do the same thing as TreeFix, but it is one of very few that has a low dependency on computer resources. This makes it the ideal program for people who do not have access to anything more than their everyday laptop/desktop. This will be the biggest draw for the product, getting consumers to use this over the competitors.

11 Lessons Learned

11.1 Technical

- Learned how to program in python as well as some limitations of the language.
- Learned how to optimize code to decrease run time through various tricks in python.
- Proper modularization of an application makes new additions a lot simpler to implement
- Learned how to effectively and efficiently evaluate a framework in the absence of documentation.

11.2 Non-Technical

- When making extensions to a program it is very important to learn and detail the design of the program. This makes it easier to make changes to the code, and keep it modular and extendable for further changes.
- Learned how to talk to clients to understand design requirements and map out features.
- Learned how to budget time/resources effectively to finish a product within a set time frame
- Learned how to delegate responsibility between team members to accomplish tasks
- Learned how to effectively test a scientific algorithm through various means.

12 Summary

Our team is hopeful that MulRF Treefix will positively affect our client's work. We approached the project with the most minimal understanding of the biology involved, but we managed to reconcile for ourselves all the most important aspects in order to get the job done right. The design approach applied by the team will allow MulRF Treefix to remain a flexible application, one that is able to run with various cost models. The testing process was extremely thorough, with zero failures out of over one hundred thousand test cases.

13 References

1. Banal, Mukul S., Yi-Chieh Wu, Eric J. Alm, and Manolis Kellis. "Improved Gene Tree Error Correction in the Presence of Horizontal Gene Transfer." *Improved Gene Tree Error Correction in the Presence of Horizontal Gene Transfer*. Oxford Journals, 5 Dec. 2014. Web.
2. MulRF: a software package for phylogenetic analysis using multi-copy gene trees Ruchi Chaudhary, David Fernández-Baca, and J. Gordon Burleigh, *Bioinformatics*, doi:10.1093/bioinformatics/btu648.
3. Inferring Species Trees from Incongruent Multi-Copy Gene Trees Using the Robinson-Foulds Distance Ruchi Chaudhary, J. Gordon Burleigh, and David Fernández-Baca, *Algorithms for Molecular Biology* 8:28, 2013.

14 Appendices

Appendix A. Operations Manual

Tutorial

*Software Requirements, Installation Instructions and Explanation of Command Line Options are revised from the compbio.mit.edu/treefix/tutorial.html website

Software Requirements:

- Python (2.5.4 or greater): http://python.org
- C compiler (gcc)
- SWIG (1.3.29 or greater): http://www.swig.org
- Numpy (1.5.1 or greater): http://www.numpy.org
- If Numpy is not found, TreeFix uses Python's built-in 'random' module.
- Scipy (0.7.1 or greater): http://www.scipy.org
- If Scipy is not found, TreeFix uses internal libraries to approximate the normal distribution (so p-values may be slightly off).
- Additionally, Python modules are required for computing (1) the p-value for likelihood equivalence and (2) the reconciliation cost.

Installation Instructions:

The package contains a file called INSTALL.txt which has detailed instruction on how to install the software.

Detailed step-by-step are as follows:

Step-by-step installation instructions:

1. Create a new directory called TreeFix in your home directory.

```
mkdir TreeFix
```

2. Download TreeFix from http://compbio.mit.edu/treefix and copy it to the newly created directory.

3. Extract TreeFix from the tarball and enter the extracted folder.

```
tar -xvzf treefix-1.1.7.tar.gz
cd treefix-1.1.7
```

4. Run the installation scripts.

```
python setup.py build
python setup.py install
```

If both the above steps were successfully executed, then TreeFix is now installed and ready to be used and you may proceed to the installation instructions for TreeFix-DTL below.

5. If users do not have system permissions to install in the default location then the install step above will fail. If this happens then the --prefix flag can be used to specify the directory where TreeFix should be installed. Thus, if the build step above succeeded but the install step failed, then please execute the following command:

```
python setup.py install --prefix=~/TreeFix/sw
```

6. Finally, if you used the --prefix option above, then to ensure that the operating system can find the newly installed scripts and executables, set the PATH and PYTHONPATH variables to the installation directory as follows:

```
export PATH=$PATH:~/TreeFix/sw/bin
export PYTHONPATH=$PYTHONPATH:~/TreeFix/sw/lib/python2.6/site-packages/
```

7. We recommend adding the two lines above to the .bashrc, .bash_profile, or another similar file. Otherwise, you will need to execute the two lines above each time you start a new command line session to use TreeFix. Also note that "python2.6" in the PYTHONPATH may change depending on the Python version installed.

Explanation of Command Line Options

-s <species tree>, --stree=<species tree>
specifies the location of the species tree file (in newick format)

-S <species map>, --smap=<species map>
specifies the location of the file mapping gene names to species names

-A <alignment file extension>, --alignext=<alignment file extension>
alignment file extension (default: ".align")

-o <old tree file extension>, --oldext=<old tree file extension>
old tree file extension (default: ".tree")

-n <new tree file extension>, --newext=<new tree file extension>
file extension for the file where the reconstructed gene tree will be written
(default ".treefix.tree")

-l <log file>, --log=<log file>
log filename. Use '-' to display on stdout.

-V <verbosity level>, --verbose=<verbosity level>
verbosity level of the log file (0=quiet, 1=low, 2=medium, 3=high)
The default value is 0 (i.e., no log file will be created), but we recommend setting this value to 1.

-e <extra arguments to module>, --extra=<extra arguments to module>
extra arguments to pass to the program that computes likelihoods (the default implementations of TreeFix and TreeFix-DTL use RAxML)
The primary use of this option will be to pass along the RAxML likelihood model to be used by TreeFix or TreeFix-DTL. Further details appear below.

--niter=<# iterations>
number of search iterations to be performed (default: 100 for TreeFix and 1000 for TreeFix-DTL)

--smodule<path of cost module to be used>
Decides which cost model is going to be used by referencing the path, paths are listed below.
Default path is DupLoss
MulRF path => treefix.modules.mulrfmodel.MulRFModel
Deep Coalescence => treefix.modules.deepcoalescenceModel.DeepCoalescenceModel

Example Implementation:

cd ~/TreeFix/treefix-1.1.7/examples/

treefix -s config/fungi.stree -S config/fungi.smap -A .nt.align -o .nt.raxml.tree -n .nt.raxml.treefix.tree -V 1 -l sim-fungi/0/0.nt.raxml.treefix.log sim-fungi/0/0.nt.raxml.tree --smodule treefix.modules.mulrfmodel.MulRFModel

Appendix B. MulRF Cost Model Code

```
from treefix.models import CostModel
import optparse
from rasmus import treelib
from compbio import phylo
import StringIO
class MulRFModel(CostModel):
    """Computes Robinson-Foulds or MulRF costs"""
    def __init__(self, extra):
        """Initializes the model"""
        CostModel.__init__(self, extra)
        self.VERSION = "1.0.1"
        self.mincost = 0
        self.count = 0
        self.log = open('matched.txt', 'w')
    def optimize_model(self, gtree, stree, gene2species):
        """Optimizes the model"""
        CostModel.optimize_model(self, gtree, stree, gene2species)
        if not (treelib.is_rooted(gtree) and treelib.is_binary(gtree)):
            raise Exception("gene tree must be rooted and binary")
        if not (treelib.is_rooted(stree) and treelib.is_binary(stree)):
            raise Exception("species tree must be rooted and binary")
        try:
            junk = phylo.reconcile(gtree, stree, gene2species)
        except:
            raise Exception("problem mapping gene tree to species tree")
    def compute_cost(self, gtree):
        """Returns the duplication-loss cost"""
        recon = {}
        find_dup = {}
        streeCopy = self.stree.copy()
        def recon_dup(stree, gtree, recon, dup):
            leaves = gtree.leaves()
            for node in leaves:
                sNode = stree.nodes[self.gene2species(node.name)]
                recon[node] = sNode
                if not sNode in dup:
                    dup[sNode] = []
                dup[sNode].append(node)
```

```
        def duplicate_nodes(tree, find_dup):
            leaves = tree.leaves()
            for node in leaves:
                if node in find_dup and len(find_dup[node]) > 1:
                    node.parent.children.remove(node)
                    parent = treelib.TreeNode(name = streeCopy.new_name())
                    node.parent.children.append(parent)
                    count = 0
                    append = parent.children.append
                    while count < len(find_dup[node]):
                        count = count + 1
                        append(node)
        def printLca(tree, lca):
            for node in lca:
                if  len(node.leaves()) > 1 and (node is not tree.root):
                    print >> self.log, node
                    print >> self.log, lca[node]
        def lca(node, lca_dict):
            """Creates a dictionary of (node, lca) pairs from given tree"""
            if node.is_leaf():
                lca_dict[node] = []
                lca_dict[node].append(node)
            else:
                lca_dict[node] = []
                append = lca_dict[node].append
                for child in node.children:
                    lca(child, lca_dict)
                    for x in lca_dict[child]:
                        append(x)
        stree_lca_dict = {}
        gtree_lca_dict = {}
        recon_dup(streeCopy, gtree, recon, find_dup)
        duplicate_nodes(streeCopy, find_dup)
        lca(gtree.root, gtree_lca_dict)
        lca(streeCopy.root, stree_lca_dict)
        cost = len(stree_lca_dict) + (len(gtree_lca_dict) - len(gtree.leaves()) - 1 )
        for sNode in stree_lca_dict:
            if len(stree_lca_dict[sNode]) > 1 and sNode != streeCopy.root:
                for gNode in gtree_lca_dict:
                    if len(stree_lca_dict[sNode]) == len(gtree_lca_dict[gNode]) and gNode !=
gtree.root:
                        stree_lca = []
                        for x in stree_lca_dict[sNode]:
                            stree_lca.append(x)
                        match = True
                        for node in gtree_lca_dict[gNode]:
                            if recon[node] not in stree_lca:
                                match = False
                                break
                            else:
                                stree_lca.remove(recon[node])
                        if match:
                            cost = cost - 2
            else:
                cost  = cost - 1
        return cost
```

Appendix C. Deep Coalescence Cost Model Code

```
# treefix libraries
from treefix.models import CostModel

# python libraries
import optparse

# rasmus libraries
from rasmus import treelib

# compbio libraries
from compbio import phylo

#debug
import StringIO
#=============================================================================

class DeepCoalescenceModel(CostModel):

	def __init__(self, extra):
	"""Initializes the model"""
	CostModel.__init__(self, extra)

	self.VERSION = "1.0.1"
	self.mincost = 0

	def optimize_model(self, gtree, stree, gene2species):
	"""Optimizes the model"""
	CostModel.optimize_model(self, gtree, stree, gene2species)

	# ensure gtree and stree are both rooted and binary
	if not (treelib.is_rooted(gtree) and treelib.is_binary(gtree)):
		raise Exception("gene tree must be rooted and binary")
	if not (treelib.is_rooted(stree) and treelib.is_binary(stree)):
		raise Exception("species tree must be rooted and binary")
	try:
		junk = phylo.reconcile(gtree, stree, gene2species)
	except:
		raise Exception("problem mapping gene tree to species tree")

	def compute_cost(self, gtree):
	"""Returns the duplication-loss cost"""
	#recon = phylo.reconcile(gtree, self.stree, self.gene2species)

	recon = {}

	def recon_dup(stree, gtree, recon):
		#Recon plus find duplications in gene tree, for each gene tree leaf we map it
to a stree leaf, we also keep track of duplications in the gene tree
		leaves = gtree.leaves()
		for node in leaves:
		sNode = stree.nodes[self.gene2species(node.name)]
		recon[node] = sNode
```

```
#LCA method
def lca(node, lca_dict):
	"""Creates a dictionary of (node, lca) pairs from given tree"""
	if node.is_leaf():
	lca_dict[node] = []
	lca_dict[node].append(node)

	else:
	lca_dict[node] = []
	append = lca_dict[node].append
	for child in node.children:
		lca(child, lca_dict)
		for x in lca_dict[child]:
		append(x)

stree_lca_dict = {}
gtree_lca_dict = {}

geneToSpeciesMap = {}

recon_dup( self.stree, gtree, recon)
lca(gtree.root, gtree_lca_dict)
lca( self.stree.root, stree_lca_dict)

#create a mapping from gene nodes to species nodes
for gNodeLca in gtree_lca_dict:
	if not gNodeLca.is_leaf():
	gNode = gtree_lca_dict[gNodeLca][0] #Get the first node in the lca
	sNode = recon[gNode]

	gene2speciesLca = []
	for node in gtree_lca_dict[gNodeLca]:
		gene2speciesLca.append(recon[node])

	found = False

	while not found:

		for node in gene2speciesLca:
			if node not in stree_lca_dict[sNode]:
			break
		else:
		found = True
		geneToSpeciesMap[gNodeLca] = sNode

		sNode = sNode.parent

nodeCount = 0
map = {}
#count distance
count = 0
for node in geneToSpeciesMap:
	nodeCount = 0
	for child in node.children:

	if child.is_leaf():
```

```
                snode = recon[child]
        else:
                snode = geneToSpeciesMap[child]

        while snode != geneToSpeciesMap[node]:
                nodeCount +=1
                snode = snode.parent
        map[node] = nodeCount
        count += nodeCount
#assuming the number of edges equals the number of nodes - 1
def countChildern(node):
        count = len(node.children)
        for child in node.children:
        count += countChildern(child)

        return count


count = count - countChildern(self.stree.root)

return count
```