Verification of Phylogenetic Inference Programs Using Metamorphic Testing

Md. Shaik Sadi[1], Fei-Ching Kuo[1], Joshua W. K. Ho [2],
Michael A. Charleston[2,3,4] and T. Y. Chen[1]

[1]Faculty of ICT, Swinburne University of Technology, VIC, Australia
[2]School of Information Technologies, The University of Sydney, Australia
[3]Centre for Mathematical Biology, The University of Sydney, Australia
[4]Sydney Institute for Emerging Infectious Diseases and Biosecurity, Australia

March 25, 2011

Abstract

Most phylogenetic inference programs use aligned sequences of characters, typically DNA or amino acid, to infer evolutionary relationships among taxa. The evolutionary history of a set of taxa is represented by a tree structure called a phylogenetic tree. A number of methods are available to estimate these trees, which have been implemented in programs in several languages for automatic tree estimation. However, it is in most cases impossible to verify the correctness of the tree returned by these programs, as the correct evolutionary history is generally unknown, and unknowable. This difficulty is known as the Oracle problem of software testing: there is no Oracle we can approach for the guaranteed-correct answer. This makes it very challenging to test the phylogenetic inference programs. Here, we apply *Metamorphic Testing* (MT) which can ameliorate the Oracle problem to test the correctness of phylogenetic inference programs. We have used two types of input, namely real and random, to evaluate the effectiveness of metamorphic testing, and found that metamorphic testing can detect failures effectively in faulty Phylogenetic inference programs with both types of inputs.

Keywords: verification, software testing, metamorphic testing, Phylogenetic inference programs, bioinformatics, PHYLIP.

1 Introduction

A fundamental concept in biology is that different taxon evolve from a common origin. The description of the evolutionary history of a group of taxa is called a phylogeny, and is typically inferred from molecular sequences of different taxon, and represented as an evolutionary tree or phylogenetic tree. Phylogenetic inference is used in modern pharmaceuticals research for drug discovery, designing genetically enhanced organisms, and understanding rapidly mutating viruses [1].

Different statistical and computational methods are available to infer phylogenetic trees. Development of such methods has been a major research problem of computational phylogenetics for more than 30 years [1][2]. However, much less attention has been paid to the practice on how these methods are implemented

in software. It is obvious that incorrect implementation of the methods can lead to incorrect data analysis, and hence false estimation of phylogenetic trees.

We believe that the programs that generate phylogenetic trees are tested much less stringently as they should be. One main reason for this is that most phylogenetic inference algorithms are computationally complex, so the "correct" evolutionary history against which we can check the estimated tree is unknown, not least because it is very computationally expensive to estimate. This issue is known in software testing as the Oracle problem. In this article, we address the Oracle problem of testing phylogenetic inference programs by a well-known testing method called *Tetamorphic Testing* (MT) [3].

A previous study on biological network simulators using MT inspired us to explore its applicability and effectiveness on phylogenetic inference programs [4]. In this project, we have chosen three phylogenetic inference programs and tested them with two types of inputs: (*i*) DNA from existing taxa, and (*ii*) randomly generated sequences.

The rest of the paper is organized as follows: a brief description of the phylogenetic inference programs we tested is given in Section 2, which is followed by Section 3 containing the details of Metamorphic Testing. Section 4 contains our proposed metamorphic relations to test the Phylogenetic inference programs. Section 5 describes the experimental set-up and Section 6 is comprised of the experimental results. Section 7 discusses the findings and concludes the paper.

2 Phylogenetic Inference Programs

A number of Phylogenetic software packages namely *PHYLIP* [5], *PAUP* [6], *MEGA* [7], *MRBAYES* [8], *RAxML* [9] etc. are available for generating Phylogenetic trees. We have chosen *dnapars*, *dnapenny* and *dnaml* programs from *PHYLIP* version 3.68 for testing. *dnapars*, *dnapenny* and *dnaml* have 8600, 7781 and 9527 lines of code excluding the comment lines, respectively. All the programs are written in programming language C. The programs present a user menu for users to execute different functionalities of the program. Different execution options may require different input files. Inputs, outputs and algorithms of these three programs are described below.

2.1 Inputs

One essential input file called "infile" to *dnapars*, *dnapenny* and *dnaml* programs is the one consisting of DNA sequences of multiple taxa. For each taxon, the DNA sequence is made by IUPAC characters [10], where each character is called a "nucleotide". n taxa and m nucleotides in DNA sequence are presented by a $n * m$ matrix (n rows and m columns), shown in the input file. The column of nucleotides are called "site" by the Phylogenetic inference scientists. A sample input file is shown in Figure 1, where 20 is the number of taxa and 50 is the number of sites as shown in the first line of the file.

For *dnaml* program, menu options ("U" and "L") used to accept another input file namely "intree". The "intree" file contains the Phylogenetic tree in newick format (to be further discussed in Section 2.3) [11]. As *dnaml* program also generates Phylogenetic trees in newick format, the tree shown in Figure 5 can be an example of intree.

2.2 Program Description

There are several methods to construct Phylogenetic trees, including the maximum parsimony and maximum likelihood methods. Different Phylogenetic inference programs implement different methods. The methods implemented by the chosen programs of this experiment are described in the following subsection.

2.2.1 *dnapars* and *dnapenny*

Both *dnapars* and *dnapenny* implement the maximum parsimony method to construct Phylogenetic trees. Based on the given DNA sequences of taxa, these two programs calculate nucleotide changes (or called

```
20 50
spe0      ATGAGGCTCTGAGGGTCCCACTCAGTGTGTGCCGCCGAGCTTCCCCGCGC
spe1      AAAGTTTAACCTGACTTGTGCGGTAGTTCAGGCGCATCAAGGTAGCCAAG
spe2      TTATCAGCCCACGTATAACTGCGTTCATGGTCTTACTGTGCGGCTACATG
spe3      GGCCTCGCCGCCGCAGACTTACGGTGCACAACACCAGCGAAAGGCCCCTA
spe4      ATTGAGCAGATGGCCGGGCAAAAATAATAGGGAGGCCTTCAAGTTCTACA
spe5      TAAGTAAAAGCTGCTCGAATCGATCCCTCCGGAGCTGTTTTCCTCCCTTT
spe6      CGTGCTAGACGCGATGCCGCCTAGCAACCGAATATCCTTCGTCAGCGGGG
spe7      CATGTACCGGCTGAAACAGGTGGTGATTCACGTTTGCGCGTACGCCATCA
spe8      CGCCACTGGTAGGACGATACTAAACTTGCGGCAGCTAGGTTTTAGGCAAC
spe9      CGATTCGGCGCAGTACTCCAGTTCGGACATTTAGCCAGCTCGGGATACTA
spe10     TCACGGTCTCACGGTGAGGTAGGAGTACGCTTTGCAGTTATTTACAACAA
spe11     CCGTATGGTGTGGGAAGCTCTACTGACTTAGAAGGCGCCTTTCCCGCCTA
spe12     TAGACCAGCTTAGTGCTTCCAGGCCTATCTCTCATCTGCCCCACGATCTC
spe13     TACCTTGACCCGGAAACTAAGTCGAAAGGTCAACCCGGACCATCACACCC
spe14     GGACGGGGGGACGCCAGCGAATTTTAGAATTATGCGACTTTTCCTGAAGC
spe15     TATGCGATCGATGGATGGAAACCTTCCTGAGATGTAACCCTTGCGTAGGG
spe16     GAAATCGGCGCAGGATTGCTAACCAGGACGACAGCTGCTATGGCACACTG
spe17     TCAAGGCTCTGGGGCCCCGGCCTAGCTTAGATCGATAGCATGTACGCCGC
spe18     TAGAGCCGGTAAGCGGCCCTTGTCGTAGTAAAGCAAAGGATGGAGCACGT
spe19     ACATAAATACTTGTTGTGTGGGAACACAAAGATCAATCAGAACGCGCTAT
```

Figure 1: Input file (infile) consisting DNA sequences

evolutionary steps) among sites, to generate Phylogenetic trees. Evolutionary steps calculated for the entire tree is called the total length, while those calculated for the branch is called the branch length. The maximum parsimony method aims to minimize the number of evolutionary steps to generate the maximum parsimony tree. Maximum parsimony tree is a Phylogenetic tree which has least total length. For getting the maximum parsimony tree, an initial Phylogenetic tree is prepared with the first n taxa of the infile ($n = 3$ for *dnapars*, $n = 2$ for *dnapenny*). These programs then read one sequence of DNA at a time and works out the maximum parsimony tree for the read in data. Read in data means the DNA sequences of taxa that already read by the program.

In *dnapars*, before and after adding one taxon to the tree, each pair of adjacent branches swap to get the maximum parsimony tree. Once all the taxa are added to the tree, sub tree rearrangements are tried in order to get maximum parsimony tree.

dnapenny uses "branch and bound" algorithm to achieve the maximum parsimony tree [12]. At each step of tree construction, if the length of a branch exceeds the predefined bound, that branch is not extended further and other branches are tried. However, this program consumes a lot of computation time, when dealing with more than ten taxa.

2.2.2 *dnaml*

dnaml uses the maximum likelihood method. In this method, the evolution of taxa is considered as a stochastic process in which "evolutionary changes among sites" depends on some set of probabilities generated by the Markov model [13]. For each set of probabilities generated, the nucleotides are changed to get the likelihood tree. The Markov model then follows the evolution probabilities to calculate the likelihood.

2.3 Outputs

dnapars, *dnapenny* and *dnaml* attemp to generate tree(s) that best describe the evolutionary history among taxa. However, these three programs have a common goal, they use different algorithms to generate the tree that describes the evolutionary history. *dnapars* and *dnapenny* aims to construct trees with the shortest "total length" but *dnaml* aim to generate trees with the highest "likelihood" against an evolutionary model. During the process, some values are outputted to the files (a file called "outfile" or a file called"outtree" or both files). Table 1 summarizes the output values for each of three program.

Table 1: Output description of the programs

Name of the program	Tree	Total length	Branch length	Likelihood
dnapars	yes	yes	yes	no
dnapenny	yes	yes	no	no
dnaml	yes	no	yes	yes

Figures 2, 3 and 4 illustrate one output tree in outfile of *dnapars*, *dnapenny* and *dnaml* respectively for the input of Figure 1. These three programs can generate more than one output tree in the outfile but the total length are same for all trees. Total length 452.00, 465.000 and likelihood -1351.32295 are also printed in Figures 2, 3 and 4.

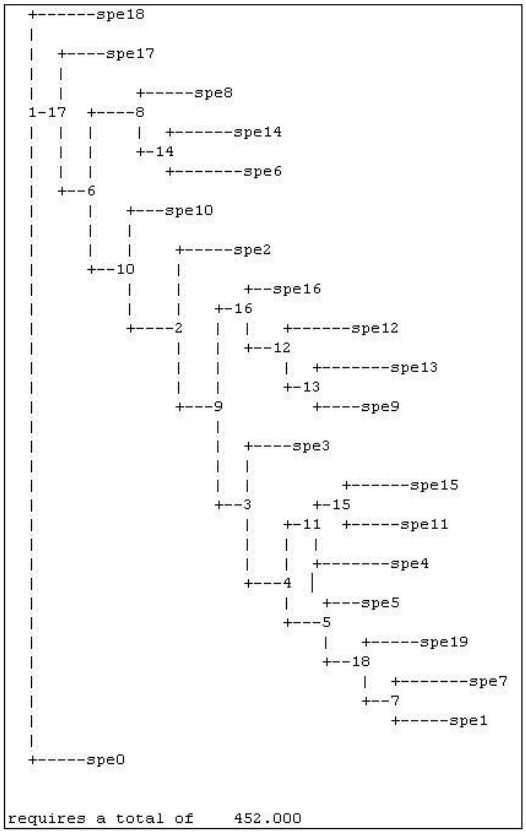

Figure 2: Output tree in outfile by *dnapars* program

"outtree" file presents the output tree in *Newick* format. Figure 5 is an example of *Newick* format of the tree for *dnapars* and *dnaml* programs, where 0.35697, 0.22394, 0.29697 etc. are the branch lengths. *dnapenny* does not output branch lengths to the "outtree" file. Figure 6 shows the tree in newick format for *dnapenny* in outtree file.

2.4 The Oracle Problem

An oracle in software testing refers to a mechanism that may be used to verify the output of the software: the correctness of a test output is determined by querying the oracle. In phylogenetics it is not in general possible to guarantee that the estimated tree is *correct*, because it is not possible to go back in time and observe the pattern of speciation, so here we will restrict ourselves to verifying that the estimated tree is

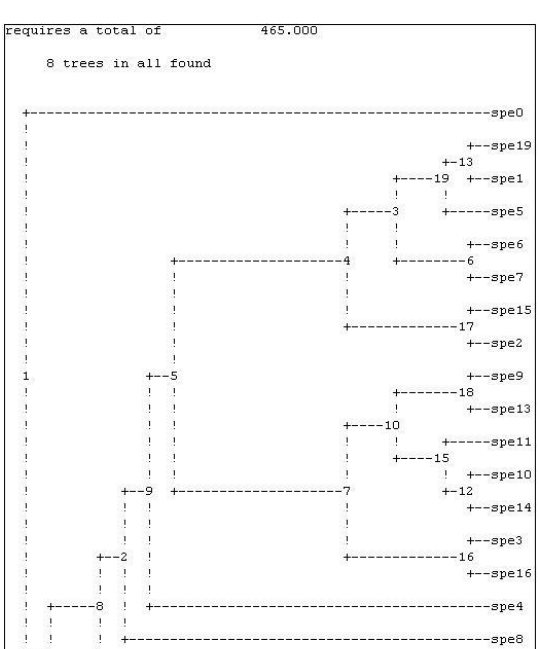

Figure 3: Output tree in outfile by *dnapenny* program

optimal for some criterion. An oracle for phylogenetic inference programs can therefore be constructed by manually applying the underlying algorithm when the number of taxa is small. However, with many taxa it is infeasible to manually verify the correctness of test output. When an oracle to test software cannot be constructed testing of software thus becomes difficult.

3 Metamorphic Testing

Metamorphic testing is an approach to alleviate the Oracle Problem. This approach does not rely on the test oracle to verify the output, but rather, checks the expected relations among inputs and outputs of the program under test. These relations are called Metamorphic Relations (denoted as MRs henceforth). They are derived based on the properties of the algorithm being implemented. In this testing method, some initial test inputs called original test inputs are generated, using any existing test case generation method. According to the derived MRs, new test inputs called follow-up test inputs are generated based on the original test inputs. The program is executed with both original and follow-up test inputs, and their outputs are compared according to the MRs. If the comparison of any source and follow-up test cases pair does not satisfy the corresponding MR then a bug is found. The process of metamorphic testing can be described as follows: Let t be a original test input of a program P. Use one MR (denoted as R) of program P to generate one follow up input *t'* for *t*. The output of the program P on execution of t and *t'* are denoted as *P(t)* and *P(t')* respectively. According to *R*, *P(t)* and *P(t')* should have some relations among them. The whole testing process can be automated easily, including the follow-up test case generation and output comparison [14].

put references at the end of sentences or phrases, so they don't interrupt the flow

Metamorphic testing can be best understood with an example. Let us consider an electronic circuit where we can control the resistance of the circuit to monitor the current flow in the circuit. Suppose the simple relation exists between voltage, current and resistance of $V = IR$. One MR can be derived based on this relation as follows: suppose the original input resistance is R_1 and the follow-up resistance is $R_2 = 2R_1$, then the follow-up current should satisfy $I_2 = I_1/2$ where I_1 and I_2 are the original and

consider using "taxon" and "taxa" through out

Sadi: added

what does t mean?

Sadi: I ha added metham ical r tation MT. Ho this v works

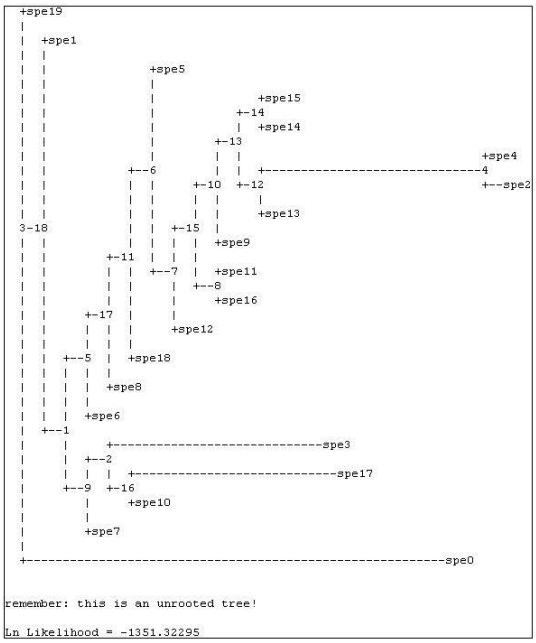

Figure 4: Output tree in outfile by *dnaml* program

```
(spe18:0.35697,(spe17:0.22394,((spe8:0.29697,(spe14:0.34924,spe6:0.40899)
:0.16146):0.24778,(spe10:0.18694,(spe2:0.31770,((spe16:0.15973,(spe12:0.36583,
(spe13:0.38917,spe9:0.25417):0.10000):0.20097):0.17655,(spe3:0.22729,
(((spe15:0.35514,spe11:0.30968):0.16075,spe4:0.37515):0.17567,(spe5:0.18911,
(spe19:0.31587,(spe7:0.41218,spe1:0.27764):0.11297):0.19444):0.18317):0.19235)
:0.14579):0.17364):0.23142):0.18000):0.17056):0.15348,spe0:0.30727);
```

Figure 5: Output tree in newick format in outtree file for *dnapars* and *dnaml*

follow-up output current respectively.

In current flow resistance circuit the value of I_2 is compared with I_1 to check identity relation. This particular example shows how an identity relation can be used to define an MR. Identity relation are mostly used by numerical programs but it is worth to note that MRs are not restricted to identity relations only, they can take any form [15].

Metamorphic testing employs some relations between the input and output of a program for testing, so this testing method does not require the oracle.

4 Metamorphic Relations

By analysing the property of the chosen algorithms for Phylogenetic inference programs, we have defined some relevant metamorphic relations to test the programs. As *dnapars* and *dnapenny* implement the method of maximum parsimony, one set of metamorphic relations were generated for both programs. On the other hand, we developed two different metamorphic relations for *dnaml*.

We will represent the DNA sequences of the input as a matrix X in this section to facilitate the discussion of the MRs. For n taxa and m sites, matrix $X = \{x_{ij} \ |1{\leq}i{\leq}n, \ 1{\leq}j{\leq} \ m\}$, where x_{ij} is a nucleotide from the IUPAC character set (denoted as US). In this study we let $US = \{A, T, C, G\}$ because A, T, C, G are the most common characters encountered in real DNA sequences. An example input X is given in Figure 7.

We use X and X' to denote the original and the follow-up inputs, respectively when describing MRs. We also use T and T' to represent a set of original and follow-up output trees for *dnapars* and *dnapenny* and t and t' are used to denote the corresponding total lengths. For *dnaml*, l and l' represent original and follow-up likelihood, respectively.

```
(spe0,((((((((spe19,spe1),spe5),(spe6,spe7)),(spe15,spe2)),(((spe9,
spe13),(spe11,(spe10,spe14))),(spe3,spe16))),spe4),spe8),spe12),
(spe18,spe17)))
```

Figure 6: Output tree in newick format in outtree file in dnapenny

$$\mathbf{X} = \begin{pmatrix} x_{11} & x_{12} & \dots \\ x_{21} & x_{22} & \dots \\ \vdots & \vdots & \ddots \end{pmatrix} = \begin{bmatrix} \texttt{ATCGAAGCAA} \\ \texttt{AGCGATGTTG} \\ etc. \end{bmatrix}$$

Figure 7: Matrix format of DNA sequences

4.1 Metamorphic Relation for *dnapars* and *dnapenny*

We have borrowed some terminologies from Phylogenetic inference community to explain our MRs. These terminologies are summarized below.

Sites that are highly conserved among all sequences or sites that contain the same nucleotide in all sequences (e.g., site 1, 4 and 5 in Figure 7) are called **conserved sites** or **parsimony-uninformative sites**. If all the nucleotides of a site are different then the site is called **hypervariable sites** (e.g., site 10 in Figure 7). The sites that are mostly conserved except for a change in one sequence (i.e. sites that have two type of nucleotides, one occur multiple times and the other one occur only one time in the sequence) are called **singleton sites** [7] (e.g., sites 3, 6 and 7 in Figure 7). All other sites provide some useful information for constructing a Phylogenetic tree, and therefore are called **parsimony-informative sites** (e.g., sites 2, 8 and 9 in Figure 7). We use these useful information to define our MRs for *dnapars* and *dnapenny*. The MRs for *dnapars* and *dnapenny* are discussed below.

MR1: If we generate a follow-up input X' by swapping two sites (the columns) in the original input X (see example below), then the set of original and follow-up output trees T and T' are identical and their corresponding total lengths t and t' are equal.

Example: The follow-up input X', by interchanging column 3 and 8 of original input X in Figure 7 looks like:

$$\mathbf{X}' = \begin{bmatrix} \texttt{A T C G A A G C A A} \\ \texttt{A G T G A T G C T G} \\ \texttt{A G T G A T A C T T} \\ \texttt{A T C G A T G T A C} \end{bmatrix}$$

this won't compile: you have to put in units for the column widths, not just "p0.009": 0.009 what? textwidth? sadi: added the unit

Output Comparison : $T = T'$ and $t = t'$.

MR2: If we add k (k>0) number of uninformative sites into the original input X to generate a follow-up input X' (see example below), then the set of original and follow-up output trees T and T' are identical and their corresponding total lengths t and t' are equal. Additions of uninformative sites are order independent and can be placed after any site of the original input X.

Example: We add five (k=5) uninformative sites (consisting of nucleotide character A) into the original input X in Figure 7 to generate follow-up input X':

$$\mathbf{X}' = \begin{bmatrix} \texttt{A T C G A A G C A A A A A A A} \\ \texttt{A G C G A T G T A A A A A T G} \\ \texttt{A G C G A T A T A A A A A T T} \\ \texttt{A T T G A T G C A A A A A A C} \end{bmatrix}$$

Output Comparison : $T = T'$ and $t = t'$.

MR3: If we remove some uninformative sites from the original input X to generate a follow-up input X' (see example below), then the set of original and follow-up output trees T and T' are identical and their corresponding total lengths t and t' are equal.

Example: We can see that there are three uninformative sites (sites 1, 4 and 5) in the original test input X in Figure 7. If we remove the two uninformative sites (sites 1 and 5) from the original input X in Figure 7 and generate X', X' looks like:

$$\mathbf{X'} = \begin{bmatrix} T & C & G & A & G & C & A & A \\ G & C & G & T & G & T & T & G \\ G & C & G & T & A & T & T & T \\ T & T & G & T & G & C & A & C \end{bmatrix}$$

Output Comparison : $T = T'$ and $t = t'$

MR4: If we double the length of DNA sequences in the original input X by the concatenation of each DNA sequence with itself to generate follow-up input X' (see example below), then the set of original and follow-up trees T and T' are identical and the follow- up total length t' is double of the original total length t. This MR is only true for input alignment with $n = 4z$ where z is a integer number.

Example: The follow-up input X' by concatenating the DNA sequence of with itself at the end in the original input X in Figure 7 is:

$$\mathbf{X'} = \begin{bmatrix} A & T & C & G & A & A & G & C & A & A & A & T & C & G & A & A & C & C & A & A \\ A & G & C & G & A & T & G & T & T & G & A & C & C & G & A & T & G & T & T & G \\ A & G & C & G & A & T & A & T & T & T & A & C & C & G & A & T & A & T & T & T \\ A & T & T & G & A & T & G & C & A & C & A & T & T & G & A & T & G & C & A & C \end{bmatrix}$$

Output Comparison : $T = T'$ and $2t = t'$

MR5: If we add some hypervariable sites into the original test input X to generate a follow-up input X' (see example below), then the set of original and follow-up output trees T and T' are identical. Hypersensitive site(s) can be placed after any site of the original input X.

Example: We add two hypersensitive sites to the original input X in Figure 7 to generate follow-up input X':

$$\mathbf{X'} = \begin{bmatrix} A & T & C & G & A & A & G & C & A & T & A & A \\ A & G & C & G & A & T & G & T & T & C & T & G \\ A & G & C & G & A & T & A & T & C & G & T & T \\ A & T & T & G & A & T & G & C & G & A & A & C \end{bmatrix}$$

Output Comparison : $T = T'$

MR6: If we apply the same transformation to change every character in every DNA sequence, for example (A→T, T→G, G→C, C→A), in the original input X to generate follow-up input X' (see example below), then the set of original and follow-up trees T and T' are identical and their corresponding total lengths t and t' are equal.

Example: We create a follow-up input X' from original input X in Figure 7 by changing (A→G, T→C, G→A, C→T):

$$\mathbf{X'} = \begin{bmatrix} G & C & T & A & G & G & A & T & G & G \\ G & A & T & A & G & C & A & C & C & A \\ G & A & T & A & G & C & G & C & C & C \\ G & C & C & A & G & C & A & T & G & T \end{bmatrix}$$

Output Comparison : $T = T'$ and $t = t'$.

MR7: If we add a duplicate DNA sequences of any taxon in the original input X to create the follow-up input X' (see example below). Then- 1. the trees of original and follow-up output sets, T and T' respectively, should differ only for the duplicate taxa such that in the follow-up output tree, the duplicates are grouped together in a sub tree, and 2. The total lengths of original and follow-up trees t and t' should be same and the output is independent on where the duplicate DNA sequence is placed.

Example: Follow-up input X' is given below by adding the duplicate DNA sequence of first taxon (first row) before the third row of original input X in Figure 7.

$$\mathbf{X'} = \begin{bmatrix} A & T & C & G & A & A & G & C & A & A \\ A & G & C & G & A & T & G & T & T & G \\ A & T & C & G & A & A & G & C & A & A \\ A & G & C & G & A & T & A & T & T & T \\ A & T & T & G & A & T & G & C & A & C \end{bmatrix}$$

Output Comparison : In T', taxon having same DNA sequence will be grouped together in a sub tree. Except the sub tree of the duplicate taxon having same DNA sequence, T and T' are the same, and the total lengths of original and follow-up outputs t and t' are the same. For the X' of MR7 we can say that taxon of the first and third row will group together in a sub tree and the tree structure of the taxon of second, fourth and fifth row is same in T and T'. **what does this mean?.Sadi: simplified it**

4.2 Metamorphic Relation for *dnaml*

For running the original and follow-up test inputs on *dnaml* we have used two input files, called "infile" (DNA sequence file) and "intree" (containing Phylogenetic trees of the DNA sequences). The "outtree" file generated by the *dnaml* program using the default option and only infile, is used as original intree file by renaming as "intree" for these two MRs. For the first MR infile is same for original and follow-up test inputs but intrees are different. For the second MR intree is same in both original and follow-up test inputs and infiles are different.

For ease of discussion we will use the original input X in Figure 7. The intree file of X is denoted as original intree S and is given below:

```
(seq4:0.06250(seq3:0.16250,seq2:0.06250):0.25000,seq1:0.26250);
```

We will also use S' for denoting follow-up intree.

MR ML1: If we generate a tree by swapping two taxa of any sub tree in the bottom layer of the original intree file S where the two taxa must share one immediate ancestor and append the newly generated tree to the original intree file to generate follow-up intree file S', the set of original and follow-up output trees T and T' and their corresponding likelihoods l and l' are identical.

Example: We swap seq2 and seq3 in the original intree S to generate a modified tree and append the modified tree with S to generate follow-up intree S':

```
(seq4:0.06250,(seq3:0.16250,seq2:0.06250):0.25000,seq1:0.26250);
(seq4:0.06250,(seq2:0.06250,seq3:0.16250):0.25000,seq1:0.26250);
```

Output Comaprison: $T = T'$ and $l = l'$.

MR ML2 : This MR is similar to MR6 defined to test *dnapars* and *dnapenny* in the last subsection. If we generate the follow-up input as mentioned in MR6 and also use intree, the sets of follow-up and original output tree are identical and their corresponding likelihoods are equal.

Example: Same as MR6.

Output Comparison : $T = T'$ and $l = l'$.

5 Experiment Setup

In the experiment, the original and follow-up inputs generation as well as original and follow-up outputs verification were conducted automatically. The automated process was implemented using C#.

5.1 Input Selection

As mentioned in Section 3 the original test input of MT can be generated using existing test case selection methods. For testing Phylogenetic inference programs, we have considered two types of test inputs namely real and random. The real DNA sequences of different taxon are obtained from TreeFam [16]. On the other hand, some DNA sequences were randomly generated which we denote as random input. To select one type of input to be used as original input of MT, we have used 500 test inputs of each type and measured different types of code coverage of the chosen three programs of this study. Code coverage is a measurement of the portion of the code that is executed by a test input and one of the strong criterion to measure the effectiveness of a test input.

We have used LCOV, a very renowned tool, to measure the coverage of real and random test inputs for the chosen Phylogenetic inference programs. The coverage result of the three programs for both type of test inputs are given in the Table 2.

Table 2: Coverage Results of Real and Random test suit

Program	Coverage	Real(%)	Random(%)
dnapars	Lines	38.2	38.1
	Functions	33.5	33.5
	Branches	25.7	25.6
dnapenny	Lines	38.1	20.3
	Functions	33.5	16.9
	Branches	25.6	11.3
dnaml	Lines	24.0	23.9
	Functions	20.1	20.1
	Branches	12.3	12.2

From the coverage results we found that both type of inputs cover almost same proportion of statements, functions as well as branches for *dnapars* and *dnaml* programs. Although *dnapenny* shows exception to this, we have decided to use both type of inputs for testing to measure the effectiveness of MT. Same test inputs were used for testing all the three programs.

5.2 Output Comparison

The linear representations of the resulting trees of original and follow-up outputs are compared using string matching. At first we discarded the branch lengths to match the structure of original and follow-up output trees. Each tree of the original output is matched against all the trees of the follow-up output. We have focused on the tree structure and the total length in this paper for testing.

5.3 Mutation Analysis

To measure the effectiveness of metamorphic testing to test *dnapars*, *dnapenny* and *dnaml*, mutation analysis was conducted [17]. Mutants are faulty program versions generated by seeding faults in the original version of a program. In mutation analysis, if the output of original and mutant programs for same test input differs, then the fault in the mutant program is revealed. On the other hand, if same output is produced by original and mutant program on same test input, the mutant is said to be equivalent mutant of the original program. Some very simple mutation operators are used to generate the mutants for our experiment. Mutants were generated randomly by using an automated Perl script. The script can generate mutants by mutating one statement at a time.

The PHYLIP package uses many source code files to make the executable files. As *dnapars*, *dnapenny* and *dnaml* all use seq.c and phylip.c file, we decided to generate the mutants by modifying these two files. However, most part of the code in phylip.c is related to exception handling. So we left the file unchanged and used seq.c file for generating mutants. At first a number of mutants were generated by mutating seq.c. We have randomly selected one program among the three to start our experiment with. The randomly selected first program was *dnapars* and we ran the program with the mutated seq.c files. We have analysed and excluded those mutants which are equivalent to original programs. The mutant having same result in original version with ten thousand test inputs, is considered as equivalent mutant. We have also exclude those mutants that have obvious errors include giving exception during execution, not generating any output tree, falling in infinite loops, having printing mistake in the output, printing negative length and so on. At last ten mutants listed in Table 3 were selected for our analysis.

Table 3: Mutants for *dnapars*

Faulty Program	File Name	Line No.	Original Statement	Faulty Statement
M1	seq.c	738	ns = 1 <<G;	ns = 1 <<C;
M2	seq.c	2807	if (i == j)	if (i != j)
M3	seq.c	565	if (ally[alias[i - 1] - 1] != alias[i - 1])	if (ally[alias[i - 1] - 1] >= alias[i - 1])
M4	seq.c	1115	for (i = a; i <b; i++)	for (i = a; i <= b; i++)
M5	seq.c	567	j = i + 1;	j = i - 1;
M6	seq.c	992	for (i = (long)A; i <= (long)O; i++)	for (i = (long)A; i >(long)O; i++)
M7	seq.c	575	itemp = alias[i - 1];	itemp = alias[i + 1];
M8	seq.c	1137	for (j = (long)A; j <= (long)O; j++)	for (j = (long)A; j >= (long)O; j++)
M9	seq.c	1077	else p->numsteps[i] += weight[i];	else p->numsteps[i] -= weight[i];
M10	seq.c	1182	for (j = (long)A; j <= (long)O; j++)	for (j = (long)A; j >(long)O; j++)

We have randomly selected *dnapenny* as our second program for analysis. We noticed that the execution of *dnapenny* program does not follow the faulty path of M2, M4, M6, M8, M9 and M10 in Table 3. So along with having M1, M3, M5 and M7, we generated six more mutants for *dnapenny* by mutating seq.c file. The six new mutants (along with M1, M3, M5 and M7) used for testing *dnapenny* program are given in Table 4.

Since seq.c file is commonly used by all the three programs, the mutants generated for *dnapars* and *dnapenny* by mutating seq.c file, was executed for *dnaml* as well. A new set of five mutants were generated for *dnaml*, as the faulty path of any mutant was not followed executing the program. As the two defined metamorphic relations for testing *dnaml* are related to the permutations of the taxon in a sub tree as well as permutation of character (represented as characters i.e A, C, G, T) in input DNA sequences, for generating mutants for this program we have targeted the methods that work on the characters of DNA sequences. We have randomly selected one such method for generating mutants. The mutants listed in Table 8 are used for testing *dnaml* program.

6 Experiment Result

We have generated 1000 original test inputs (500 with real DNA sequences and 500 with randomly generated DNA sequences) for testing the programs. For seven MRs of *dnapars* and *dnapenny* (7*1000)= 7000 follow-up test inputs were generated respectively for each program. To test 10 mutants of each of the two

Table 4: New mutants for *dnapenny*

Faulty Program	File Name	Line#	Original Statement	Faulty Statement
M11	seq.c	959	if (p->base[i] == 0) {	if (p->base[i] != 0) {
M12	seq.c	566	if (j <= i)	if (j >i)
M13	seq.c	1277	if (p->back == p1)	if (p->back != p1)
M14	seq.c	726	for (i = 0; i <spp; i++) {	for (i = 0; i >= spp; i++) {
M15	seq.c	1278	else if (p->back == p2)	else if (p->back != p2)
M16	seq.c	1479	if (other == *root)	if (other != *root)

Table 5: Mutants for *dnaml*

Faulty Program	File Name	Line#	Original Statement	Faulty Statement
M17	seq.c	464	sumg += w * (*freqg) * treenode[i] ->x [j][0][(long)G - (long) A] / sum;	sumg -= w * (*freqg) * treenode[i] ->x [j][0][(long) G - (long) A] / sum;
M18	seq.c	460	sum += (*freqg) * treenode[i] ->x [j][0][(long)G - (long)A];	sum -= (*freqg) * treenode[i] ->x [j][0][(long) G - (long) A];
M19	seq.c	465	sumt += w * (*freqt) * treenode[i] ->x [j][0][(long) T - (long) A] / sum;	sumt -= w * (*freqt) * treenode [i] ->x [j][0][(long) T - (long) A] / sum;
M20	seq.c	461	sum += (*freqt) * treenode[i] ->x [j][0][(long) T - (long) A];	sum -= (*freqt) * treenode [i] ->x [j][0][(long) T - (long) A];
M21	seq.c	468	sum = suma + sumc + sumg + sumt;	sum = suma -sumc + sumg + sumt;

programs a total of (10*7*1000) 70000 original and follow-up test input pairs were executed. Same original test inputs were used for testing *dnaml*. For the five mutants and two MRs of *dnaml* a total (5*2*1000) 10000 original and follow-up test input pairs were executed.

6.1 *dnapars* Result:

The result of applying MT on the ten mutants of *dnapars* is summarized in Table 6. From the result we have found that most of the mutants are detected by MR7. We have also found that MR1 could only reveal failure in M3.

Among the 70000 test input pairs 18232 (26.05%) violated the MRs and hence revealed failure. As we have used both real DNA sequences and randomly generated DNA sequences, we measured the effectiveness of test inputs separately. With real DNA sequences 8753 (25.01%) test inputs revealed failure and with randomly generated DNA sequences 9479 (27.08%) test inputs revealed the failure.

A close inspection of the results of applying MT on this program reveals two opposite scenario- while M7 is detected by MR2 with random input only, M9, on the other hand, was detected by MR5 with real input only. However, analysing the overall result we have found that all MRs detected more failures with random inputs with an exception of MR4 which performed better with real inputs.

Since M11, M12, M13, M14, M15 and M16 in Table 4 and all the mutants of *dnaml* in Table 8 are generated by altering seq.c file and seq.c is used by *dnapars* as well, we have executed these mutants for *dnapars*. We have found that M12 was detected by MR1-MR6 and M16 was detected by MR7 only. M11, M13, M14 and M15 produced obvious errors with original test case for *dnapars*.

Table 6: Metamorphic testing result of mutants of *dnapars* program

MRs	types	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10
MR1	Real	0	0	127	0	0	0	0	0	0	0
	Random	0	0	172	0	0	0	0	0	0	0
MR2	Real	0	0	455	6	17	500	0	0	0	500
	Random	0	0	490	39	158	500	32	0	0	500
MR3	Real	0	475	478	43	263	496	122	0	0	0
	Random	0	500	481	65	225	500	131	0	0	0
MR4	Real	0	0	465	2	374	0	66	0	0	0
	Random	0	0	493	3	500	0	76	0	0	0
MR5	Real	0	0	40	243	12	0	0	0	7	0
	Random	0	0	66	102	13	0	0	0	0	0
MR6	Real	489	0	492	0	410	0	66	0	0	496
	Random	497	0	499	0	492	0	113	0	0	500
MR7	Real	0	500	227	232	168	267	39	192	205	279
	Random	0	500	236	336	213	252	41	301	189	264

6.2 *dnapenny* Result:

The testing result of *dnapenny* using MT on ten mutants is given in Table 7. From the result we have found that MR1 could only reveal failure in M3. Among the total original and follow-up test input pair 17603 (25.15%) revealed failure. 8273 (23.64%) real test input pair and 9330 (26.66%) random test input pair revealed failure.

Analysing the results of testing *dnapenny*, we have found that All MRs perform better in revealing failure with random test inputs.

Since seq.c file is common, the mutants of *dnaml* were executed for *dnapenny*. The faulty path of any of those mutants was not executed by the test inputs and as such none of those five mutants of *dnaml* were detected by the MRs of *dnapenny*.

6.3 *dnaml* Result:

Testing result of *dnaml* using MT on five mutants is given in Table 8. From the result we have found that MR b is effective for all the mutants. No mutant was detected by MR ML1. Among the total test inputs 2500 (50%) real test inputs and 2496 (49.92%) random test inputs revealed failure.

For this program we have seen that almost same numbers of mutants were detected by both real and random test inputs.

Table 7: Metamorphic testing result of mutants of *dnapenny* program

MRs	types	M1	M3	M5	M7	M11	M12	M13	M14	M15	M16
MR1	Real	0	89	0	0	0	0	0	0	0	0
	Random	0	178	0	0	0	0	0	0	0	0
MR2	Real	0	450	17	0	500	0	0	500	0	0
	Random	0	469	131	21	500	32	0	500	0	0
MR3	Real	0	466	245	121	496	21	0	496	0	0
	Random	0	468	223	128	500	87	0	500	0	0
MR4	Real	0	465	374	66	0	79	0	0	0	0
	Random	0	493	500	76	0	74	0	0	0	0
MR5	Real	12	84	12	0	0	0	0	0	0	0
	Random	6	184	42	2	0	14	0	0	0	0
MR6	Real	490	477	388	64	0	67	0	0	0	0
	Random	488	488	479	108	0	87	0	0	0	0
MR7	Real	0	210	161	26	500	30	465	500	400	2
	Random	0	236	234	39	500	18	500	500	500	25

Table 8: Metamorphic testing result of mutants of *dnaml* program

MRs	types	M17	M18	M19	M20	M21
MR a	Real	0	0	0	0	0
	Random	0	0	0	0	0
MR b	Real	500	500	500	500	500
	Random	500	500	500	496	500

7 Discussion and Conclusion

As mentioned earlier, Phylogenetic inference programs such as *dnapars*, *dnapenny* and *dnaml* suffer from oracle problem. To our knowledge, there is no effective way of testing the correctness of such programs. Our approach of applying MT to test mutant versions of three example Phylogenetic inference programs showed promising fault detection rate. The results illustrate that this innovative testing method can be useful in detecting faults and hence can alleviate the drawback of oracle problem in testing this kind of programs. Thus MT can be an effective method for testing Phylogenetic inference programs.

From the results of testing all the three programs with MT, we found that different MRs detect faults in different mutants. This phenomenon suggests that, defining more MRs is helpful to detect different type of faults. While executing the programs with one MR, the faulty statement may not be executed and hence the fault may remain unnoticed. Employing a variety of MRs to test a program will thus be beneficial to detect different faults.

In our experiment we have used both real DNA sequences and randomly generated DNA sequences. A coverage analysis of the programs with both types of inputs showed that both real and random inputs cover very close percentages of lines, functions and branches of three programs. In the case of *dnapars* and *dnapenny*, fault detection rate was high for random test inputs than real test inputs. However, The fault detection rate in both types of test inputs was almost same for *dnaml*. In our analysis neither type of test input outperforms the other in detecting faults. Furthermore, in some cases some mutants were detected

by MRs with one type of input only. Based on the analysis we can suggest the Phylogenetic inference scientists to test Phylogenetic inference programs with both types of test inputs.

In process of MT, defining the MRs require some kind of background knowledge on the theory of the program. As scientist put overwhelming focus on the theory, designing MRs make the task easier for them [18]. As such it is most likely that MT will be accepted more enthusiastically by the Phylogenetic inference scientist community.

With MT, we can reveal failures using a number of MRs but we can not detect the faulty statement. As a future works, we will isolate the bug by getting the execution trace of the source and follow-up test inputs of MT.

A number of Phylogenetic inference software has been implemented since the last three decades. Unfortunately no established test tool has been developed to test these kinds of software. For the purpose of this paper we have used only three programs. In future we plan to make an automated general MT tool for testing Phylogenetic inference programs.

Acknowledgement

We would like to acknowledge the support given to this project by an Australian Research Council Discovery Grant (ARC DP*******).

References

[1] B. M. E. Moret, D. A. Bader, and T. Warnow, *High-performance algorithm engineering for computational phylogenetics*, ser. Lecture Notes in Computer Science, 2001, vol. 2074, pp. 1012–1021.

[2] N. Friedman, M. Ninio, I. Pe'er, and T. Pupko, "A structural em algorithm for phylogenetic inference," *Journal of Computational Biology*, vol. 9, pp. 331–353, 2002.

[3] T. Y. Chen, S. C. Cheung, and S. M. Yiu, "Metamorphic testing: a new approach for generating next test cases," Department of Computer Science, Hong Kong University of Science and Technology, Hong Kong, Tech. Rep. HKUST-CS98-01, 1998.

[4] T. Y. Chen, J. W. K. Ho, H. Liu, and X. Xie, "An innovative approach for testing bioinformatics programs using metamorphic testing," *BMC Bioinformatics*, vol. 10, p. 24, 2009.

[5] *PHYLIP*. [Online]. Available: http://evolution.genetics.washington.edu/phylip/doc/main.html

[6] D. L. Swofford, *PAUP*. Phylogenetic Analysis Using Parsimony (*and Other Methods). Version 4.* Sunderland, Massachusetts: Sinauer Associates, 2003.

[7] S. Kumar, M. Nei, J. Dudley, and K. Tamura, "MEGA: a biologist-centric software for evolutionary analysis of DNA and protein sequences." *Briefings in bioinformatics*, vol. 9, no. 4, pp. 299–306, Jul. 2008. [Online]. Available: http://dx.doi.org/10.1093/bib/bbn017

[8] J. P. Huelsenbeck and F. Ronquist, "MRBAYES: Bayesian inference of phylogenetic trees." *Bioinformatics (Oxford, England)*, vol. 17, no. 8, pp. 754–755, Aug. 2001. [Online]. Available: http://dx.doi.org/10.1093/bioinformatics/17.8.754

[9] A. Stamatakis, T. Ludwig, and H. Meier, "RAxML-III: a fast program for maximum likelihood-based inference of large phylogenetic trees," *Bioinformatics*, vol. 21, no. 4, pp. 456–463, Feb. 2005. [Online]. Available: http://dx.doi.org/10.1093/bioinformatics/bti191

[10] Nomenclature committee of the international union of biochemistry (nc-iub). [Online]. Available: http://www.chem.qmul.ac.uk/iubmb/misc/naseq.html

[11] [Online]. Available: http://evolution.genetics.washington.edu/phylip/newicktree.html

[12] M. D. Hendy and D. Penny, "Branch and bound algorithms to determine minimal evolutionary trees," *Mathematical Biosciences*, vol. 59, pp. 277–290, 1982.

[13] J. Felsenstein and G. A. Churchill, "A hidden markov model approach to variation among sites in rate of evolution molecular biology and evolution," *Molecular Biology and Evolution*, vol. 13, pp. 93–104, 1996.

[14] C. Murphy, K. Shen, and G. E. Kaiser, "Automatic system testing of programs without test oracles," in *ISSTA*, 2009, pp. 189–200.

[15] W. J. Cody, *Software Manual for the Elementary Functions (Prentice-Hall series in computational mathematics)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1980.

[16] H. Li, A. Coghlan, J. Ruan, L. J. J. Coin, J.-K. K. Hériché, L. Osmotherly, R. Li, T. Liu, Z. Zhang, L. Bolund, G. K.-S. K. Wong, W. Zheng, P. Dehal, J. Wang, and R. Durbin, "TreeFam: a curated database of phylogenetic trees of animal gene families." *Nucleic acids research*, vol. 34, no. Database issue, Jan. 2006. [Online]. Available: http://dx.doi.org/10.1093/nar/gkj118

[17] M. R. Woodward and K. Halewood, "From weak to strong, dead or alive? an analysis of some mutationtesting issues," in *Proceedings of the 2nd Workshop on Software Testing, Verification, and Analysis (TVA'88)*, Banff Albert, Canada, July 1988, pp. 152–158.

[18] R. Sanders and D. Kelly, "Dealing with risk in scientific software development," *IEEE Software*, vol. 25, pp. 21–28, 2008.