Reducing noise in protein multialignments

Nima Behnam Makoi, nimabm@kth.se
Louis Joos, louisj@kth.se
Jan 7, 2019

Table of contents

Abstract

When using multiple sequence alignments, an issue is poorly aligned regions or noise. This project aims to implement a simple model of noise reduction and to examine whether phylogenetic reconstruction improves after alignment cleaning or not. Implementation was done in Python resulting in three programs and one script to run the noise reduction. The average distance when comparing trees was slightly lower after applying our denoising algorithm for 5 cases out of 6, albeit with very small variations. Also, most of the time the reference tree can not be recovered. We suspect that the model is too simplistic, and to obtain more distinguishable results, it has to be expanded.

Introduction

Multiple sequence alignments (MSA) are central in bioinformatics and are often the first step of any analysis. MSA is done by aligning the nucleotides or amino acids in columns based on their inferred homology by similarity, inserting gaps between positions in order to align the characters. Gaps represent insertion or deletion of portions of the sequence, or indels. Beside the algorithm used to align, another source of issue in MSA are the poorly aligned regions or noise. Homologous proteins can contain regions that are not inherited and should therefore not be aligned, and other regions may have evolved so fast that the correct multialignment is impossible to infer. Although it can be a loss of information those regions are often removed from the multi-alignments, using tools like Gblocks [1] or TrimAl [2]. Here we present a simple noise-reduction method for protein MSA. The aim was to examine whether phylogenetic reconstruction improves after alignment cleaning or not.

Material and Methods

Model

We used a simple model for noisiness. Looking at the MSA as a set of columns we define a column as noisy if:

- there are more than 50% indels
- at least 50% of amino acids are unique
- no amino acid appears more than twice

Implementation & validation

The programming language for this project was Python, version 3.7. This was complemented with a UNIX shell script to run the necessary commands. The directory structure of the project folder is as follows:

- Project folder
    - run.sh
    - bin/
        - denoising
        - difftab
        - treecompare
        - UserDefinedExceptions.py
    - data/
        - Provided folders with test files
    - results/
        - count_tables/
        - filtered_trees/
        - unfiltered_trees/

The project consists of three programs and one module, possible flags are also specified:

- denoising: used for denoising sequences
    - -o filename: output is written to filename (default: stdout)
- treecompare: used for comparing trees
    - o filename: output is written to filename (default: stdout)
    - --notext: only distance values are written, option is mostly for output not having redundant text when creating database
- difftab: used for creating a table of differences and their respective frequency
- UserDefinedExceptions.py: a Python module consisting of specifically defined exception classes

The project is run by running the run.sh script, which sets up the structure and runs all files through the programs above. For convenience reasons, the run script is placed in the project root folder and not in bin/ in accordance (ie. not in accordance with the guidelines in the Stafford-Noble paper), so that one can run the script directly from the project root folder. To make sure the programs were working properly, validation was made with the following files:

- denoising:
    - an empty file,
    - a text file,

        - a file with a mixture of columns consisting of 50% or more indels, 50% or more unique amino acids, and where no amino acids appear more than twice.
- treecompare:
    - an empty file,
    - a text file.
- difftab:
    - an empty file,
    - a text file.

all of which raised the desired exceptions. The exceptions defined for this specific project are:

- EmptyFileError: raised when input file is empty
- EmptySeqError: raised when a sequence is empty
- FastaFileError: raised if the input file format is not correct
- TreeError: raised if input is not a (properly formatted) tree

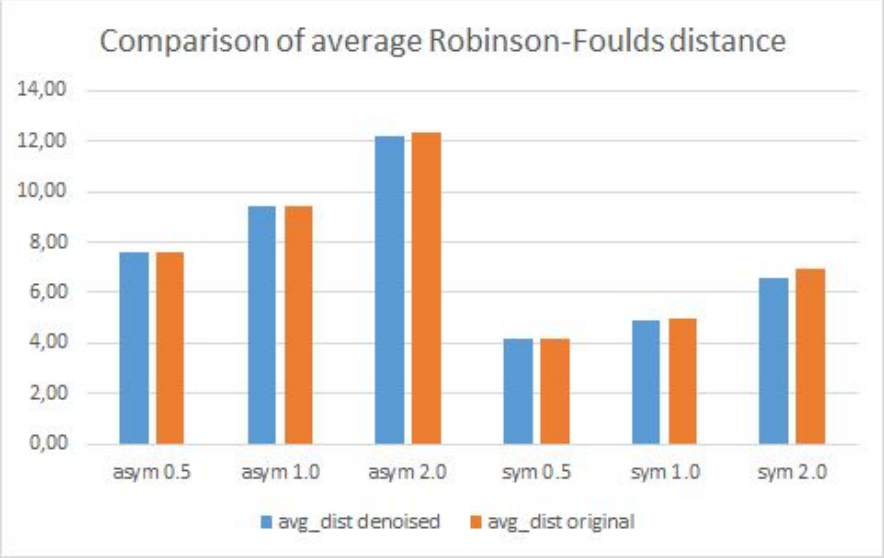
Evaluation

We evaluated our method on its ability to improve phylogenetic reconstruction. To do so we used a dataset derived from the one used to evaluate trimAl performance. It's composed of 6 subdirectories, each of them containing a reference tree and 300 MSA of 6 sequences derived from this reference tree by a computer program. There are 3 subdirectories with a symmetric tree and 3 with an asymmetric tree and for both categories 3 different average amounts of mutations per site (0.5, 1.0 and 2.0).
The evaluation process is divided in 3 parts. First we created the denoised MSA from the original MSA in our evaluation dataset. Then we used fastprot to compute the distance matrices for the original and denoised MSA, and fnj (Fast Neighbour Joining algorithm) to create new trees from them. Both fastprot and fnj are tools of the FastPhylo package [3]. Finally we computed Robinson-Foulds distance between the new trees and the reference trees in the dataset using Dendropy Python library [4], giving similarity scores before and after applying our denoising method. For each case, the similarity scores were put in a database and the occurence of each score counted. To assess whether denoising gives better results we considered looking at the average distance between the reference tree and the 300 trees created in each case.

Results and Discussion

Results

Our implementation took 2914 seconds (48.6 minutes) to compute each denoised MSA from our evaluation dataset and to also make trees from the sequence files (3,600 trees in total). The detailed tables of results are given as appendix along with bar charts showing the distribution of the Robinson-Foulds distance to the reference tree before and after denoising. In this section we only display the final results that could allow us to conclude about our problematic which is the average Robinson-Foulds distance to the reference trees for the 6 cases before and after denoising (see figures below). We observed that the average distance was slightly lower after applying our denoising algorithm for 5 cases out of 6. The only exception is for the asymmetric reference tree with 1.0 mutation per site where the Robinson-Foulds distance was slightly higher.



Category	Average distance (denoised)	Average distance (original)
Asymmetric 0.5	7.57	7.63
Asymmetric 1.0	9.45	9.43
Asymmetric 2.0	12.17	12.34
Symmetric 0.5	4.15	4.16
Symmetric 1.0	4.89	5.01
Symmetric 2.0	6.61	6.96

Discussion

As a general observation we can notice that we get way better reconstruction in cases of symmetric trees than of asymmetric trees, confirming that the latter are more difficult to work with. The same considerations apply for the number of mutations per site: the higher the larger is the Robinson-Foulds distance after reconstruction of trees from MSA. We can also notice that most of the time we cannot recover the exact same tree as the reference tree, meaning that the Robinson-Foulds distance is equal to zero. This is because the process of deriving sequences from the tree and then of applying our algorithm on the MSA introduce a loss of information making the whole pipeline non revertible.

One can question the relevance of using a time consuming denoising method considering that the average distance after denoising, even if lower, is still very close to the one we get from the original MSA. Our guess is that our noisiness model is a bit too simplistic to obtain more remarkable results. The choice of a single set of parameters (basically the 50% thresholds we used) is a problem especially when the dataset grows big as trimAl authors pointed out [2]. A more clever way could be to include some parameter tuning steps for the thresholds to depend on the sequences and doing so optimize the signal-to-noise ratio. In addition we don't have any conservation threshold, meaning an proportion of the original number of columns we don't want to go below. In our method we only trigger an error if the denoising process removes all the columns but it might be interesting to keep at least a certain amount of information.

One can also discuss the choice of the metrics. We chose the Robinson-Foulds distance, first because it was proposed to us and also because a quick look at other examples such as euclidean distance didn't give differently interpretable results, but Dendropy has a large set of metrics at disposal and another choice might have given a clearer separation between results before and after denoising. The choice of using the average of this metric over the 300 trees has been made considering there was no large outlier in the scores and median or mode couldn't discriminate results from before and after denoising.

Reflections on the Stafford-Noble paper

After reading the paper, it became apparent that, at least when it comes to the actual code and file/directory structure, the recommendations from the author were similar to methods taught in earlier courses for our part; methods which have been used in earlier projects by both of us. Keeping a lab notebook was fairly new but not at all an issue. Principles about using version control were also being followed, seeing as we used GitHub for version control and sharing purposes. Like the author,

we also used a mixture of Python- and shell scripts, and hopefully commented the code well enough so that a third party could possibly understand the meaning of the different parts. Error-preventing measures have also been taken, and the error messages should be easy enough for a user to interpret.

References

[1] Talavera, G., and Castresana, J. (2007). Improvement of phylogenies after removing divergent and ambiguously aligned blocks from protein sequence alignments. Systematic Biology 56, 564-577.

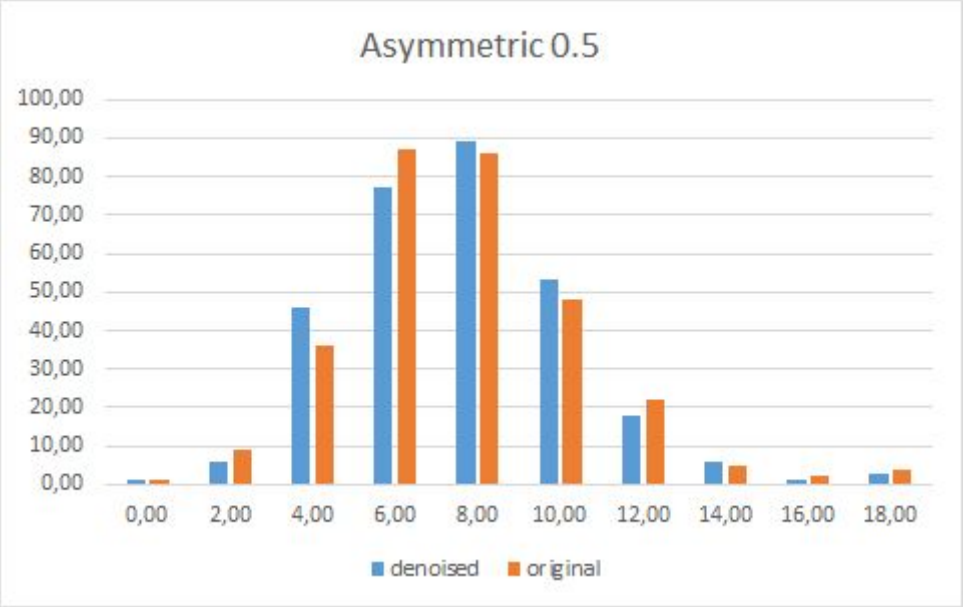
[2] trimAl: a tool for automated alignment trimming in large-scale phylogenetic analyses Salvador Capella-Gutierrez; Jose M. Silla-Martinez; Toni Gabaldon. Bioinformatics 2009 25: 1972-1973.

[3] Khan et al, FastPhylo: Fast tools for phylogenetics, BMC Bioinformatics, 2013.

[4] DendroPy: A Python Library for Phylogenetic Computing, Bioinformatics 26(12):1569-71, June 2010

APPENDIX A: Statistics

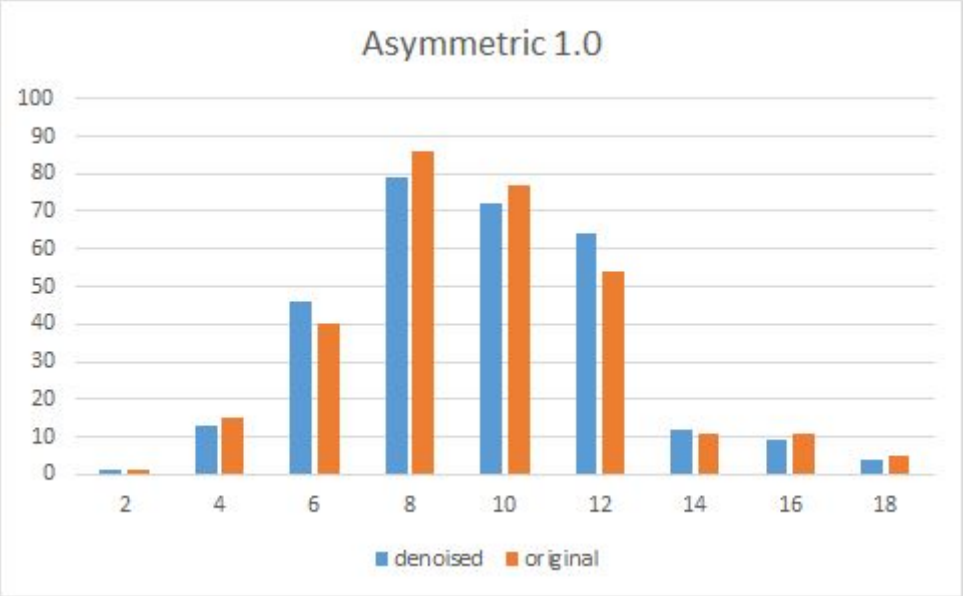
Asymmetric 0.5



Distance	Denoised	Original
0	1	1
2	6	9
4	46	36
6	77	87
8	89	86
10	53	48
12	18	22
14	6	5
16	1	2
18	3	4

	Denoised	Original
Mean	7.57	7.63
Median	8.00	6.00
Mode	8.00	8.00

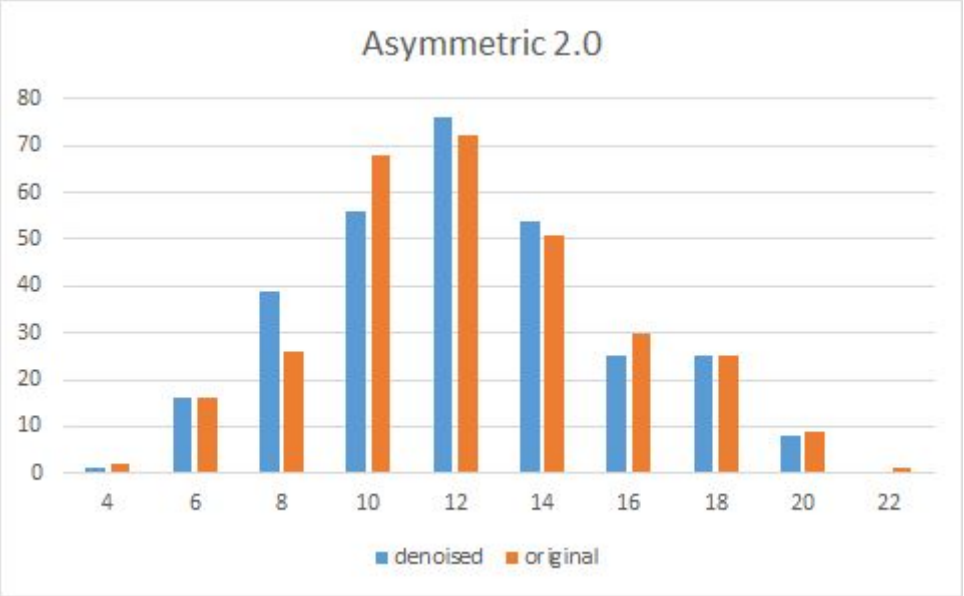
Asymmetric 1.0



Distance	Denoised	Original
2	1	1
4	13	15
6	46	40
8	79	86
10	72	77
12	64	54
14	12	11
16	9	11
18	4	5

	Denoised	Original
Mean	9.45	9.43
Median	10.00	10.00
Mode	8.00	8.00

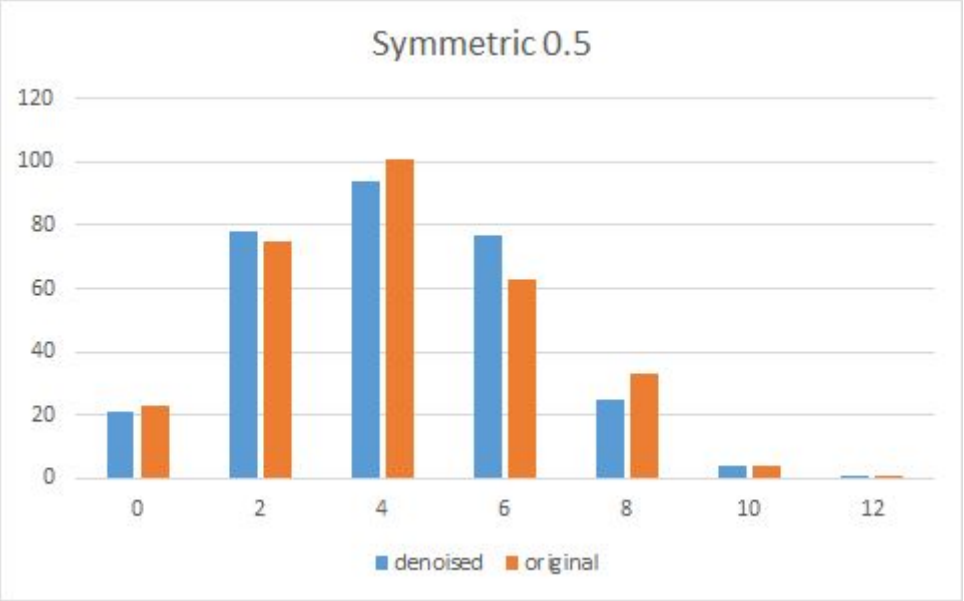
Asymmetric 2.0



Distance	Denoised	Original
4	1	2
6	16	16
8	39	26
10	56	68
12	76	72
14	54	51
16	25	30
18	25	25
20	8	9
22	0	1

	Denoised	Original
Mean	12.18	12.34
Median	12.00	12.00
Mode	12.00	12.00

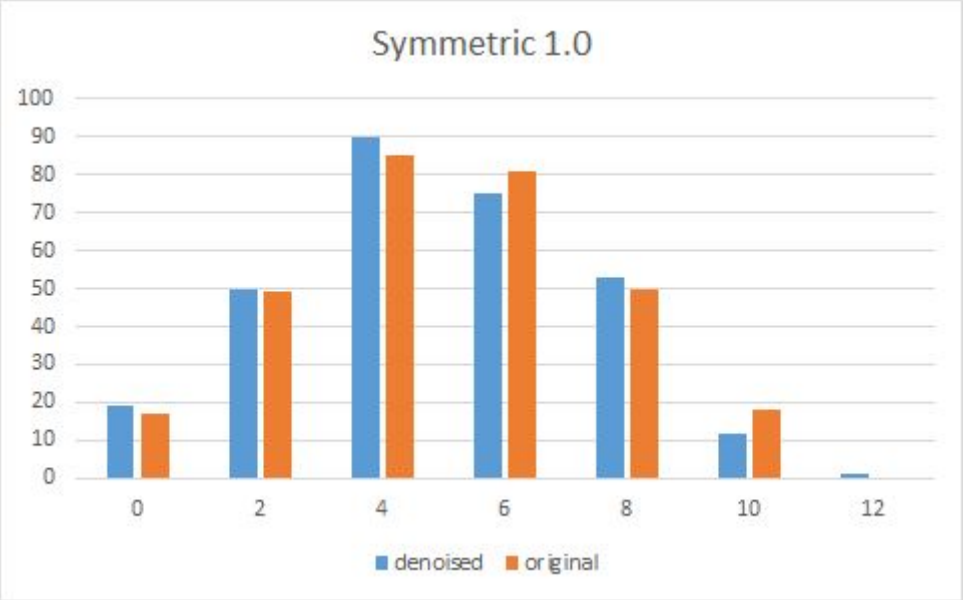
Symmetric 0.5



Distance	Denoised	Original
0	21	23
2	78	75
4	94	101
6	77	63
8	25	33
10	4	4
12	1	1

	Denoised	Original
Mean	4.15	4.16
Median	4.00	4.00
Mode	4.00	4.00

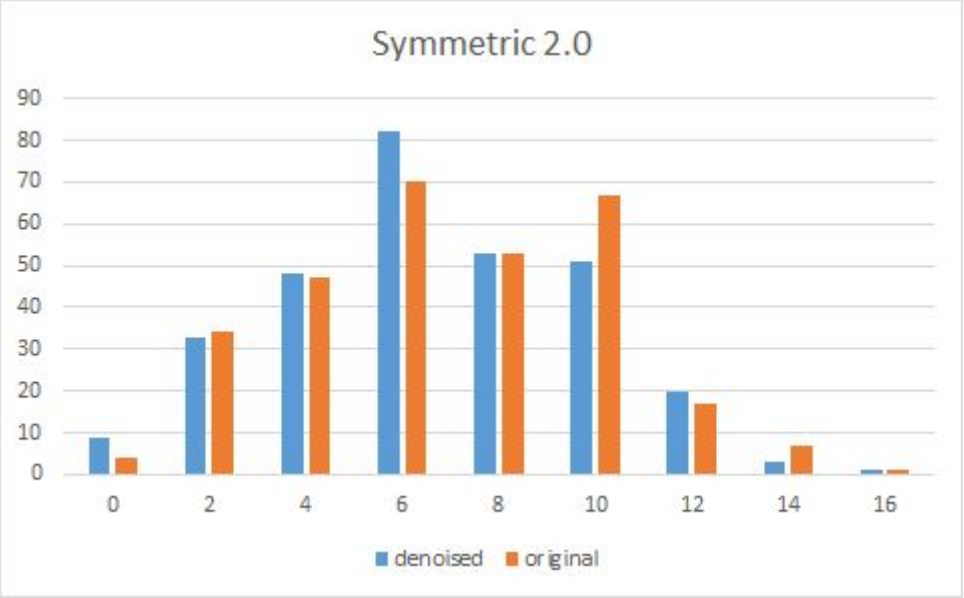
Symmetric 1.0



Distance	Denoised	Original
0	19	17
2	50	49
4	90	85
6	75	81
8	53	50
10	12	18
12	1	0

	Denoised	Original
Mean	4.89	5.01
Median	4.00	4.00
Mode	4.00	4.00

Symmetric 2.0



Distance	Denoised	Original
0	9	4
2	33	34
4	48	47
6	82	70
8	53	53
10	51	67
12	20	17
14	3	7
16	1	1

	Denoised	Original
Mean	6.60	6.96
Median	6.00	6.00
Mode	6.00	6.00

APPENDIX B: Code

denoising

```
#! /usr/bin/env python
import argparse
import sys
import re
import UserDefinedExceptions as ude
from Bio import SeqIO
from Bio.Alphabet import SingleLetterAlphabet
from Bio.Seq import Seq

# Functions
def transpose(lst):
    '''Returns the transpose of a list'''
    transposed_elements = list(map(list, zip(*lst)))
    transposedlst = []
    for e in transposed_elements:
        transposedlst.append(''.join(e))
    return transposedlst

def is_empty(s):
    '''Checks if a provided string is empty'''
    return not s

def is_space(s):
    '''Checks if a provided string is only whitespace, does not require any characters to
be present in string'''
    stripped = s.strip()
    return not stripped

# Execution
# Initialize parser and arguments
parser = argparse.ArgumentParser(description='Filters sequences in a given input file')
parser.add_argument('input_filename', metavar='fileref', type=str, help='input file
reference')
parser.add_argument('-o', dest='output', action='store', metavar='filename', type=str,
default=sys.stdout,
                    help='output filename. If not specified, output is written to stdout.')
args = parser.parse_args()

# Open file and store sequences in a list, then transpose to get columns
try:
    f = open(args.input_filename, "r")
    s = f.read()
    if is_empty(s) or is_space(s):
        raise ude.EmptyFileError
except FileNotFoundError:
    print('Error: File not found. Please try again.', file=sys.stderr)
    sys.exit()
```

```
except ude.EmptyFileError:
    print('Error: Provided file is empty. Try again with another file.', file=sys.stderr)
    sys.exit()
f.seek(0)
records = list(SeqIO.parse(f, "fasta"))
sequences = []
try:
    for record in records:
        if is_empty(record.seq) or is_space(record.seq):
            raise ude.EmptySeqError
        else:
            sequences.append(record.seq)
    if not sequences:
        raise ude.FastaFileError
except ude.EmptySeqError:
    print('Error: Provided sequence file has one or more empty sequences. Please provide
another file.', file=sys.stderr)
    sys.exit()
except ude.FastaFileError:
    print('Error: Provided file is malformed and/or not in Fasta format. Please check or
provide another file.', file=sys.stderr)
columns = transpose(sequences)

# Check if columns fulfill requirements
output = []
for column in columns:
    approved = True
    if column.count('-') / len(column) > 0.5:
        approved = False
    else:
        approved = False
        bases = list(set(column.replace('-', '')))
        cnt = 0
        for i in range(0, len(bases)):
            basecount = column.count(bases[i])
            if basecount > 2:
                approved = True
            if basecount == 1:
                cnt += 1
        if cnt / len(column.replace('-', '')) <= 0.5:
            approved = True
    if approved:
        output.append(column)

# Transpose approved columns to get the sequences, write to stdout/specified output file
filtered_sequences = transpose(output)
try:
    for i in range(0, len(records)):
        records[i].seq = Seq(filtered_sequences[i], SingleLetterAlphabet())
except IndexError:
    print('Error: Sequence file consists of only noisy columns. Please provide another
one.', file=sys.stderr)
    sys.exit()
```

```
SeqIO.write(records, args.output, 'fasta')
f.close()
```

difftab

```
#! /usr/bin/env python
import argparse
import sys
import sqlite3
import UserDefinedExceptions as ude

parser = argparse.ArgumentParser(description='Returns the count of differences in a result
file from running the script compare.sh')
parser.add_argument('infile', action='store', type=str, default=sys.stdin, help='input
filename. If not specified, input is read from stdin.')
args = parser.parse_args()

try:
    f = open(args.infile, 'r')
except FileNotFoundError:
    print('Error: File not found. Please try again.', file=sys.stderr)
values = f.read().split('\n')
f.close()
try:
    values = [val for val in values if val.isnumeric()]
    if not values:
        raise ude.EmptyError
except ude.EmptyError:
    print('Error: Input file is empty or malformed. Please check input file and try again
or provide another file.', file=sys.stderr)
    sys.exit()
conn = sqlite3.connect('differences.db')
c = conn.cursor()
c.execute('CREATE TABLE differences (difference INT)')
for val in values:
    cmd = 'INSERT INTO differences VALUES (' + val + ')'
    c.execute(cmd)
c.execute('SELECT difference, COUNT(difference) FROM differences GROUP BY difference')
print(c.fetchall(), file=sys.stdout)
```

treecompare

```
#! /usr/bin/env python
import argparse
import dendropy
import sys
import UserDefinedExceptions as ude
from dendropy.calculate import treecompare

parser = argparse.ArgumentParser(description='Compare phylogenic trees and get their
symmetric difference')
parser.add_argument('-o', dest='output', action='store', metavar='filename', type=str,
default=sys.stdout,
                    help='output filename. If not specified, output is written to stdout.')
parser.add_argument('--notext', dest='notext', action='store_false',
                    help='only the difference is written. Useful if value is to be used
elsewhere')
parser.add_argument('tree1', type=str, default=sys.stdin,
                     help='file reference for the first tree. If not specified, input will
be read from stdin')
parser.add_argument('tree2', type=str, help='file reference for the second tree. Must be
specified')
args = parser.parse_args()

try:
    tns = dendropy.TaxonNamespace()
    try:
        tree1 = dendropy.Tree.get_from_path(args.tree1, 'newick', taxon_namespace=tns)
        tree2 = dendropy.Tree.get_from_path(args.tree2, 'newick', taxon_namespace=tns)
    except:
        raise ude.TreeError
    tree1.encode_bipartitions()
    tree2.encode_bipartitions()
    if args.notext == True:
        print("Symmetric difference between " + args.tree1 + " and " + args.tree2 + ": ",
file=args.output)
    print(treecompare.symmetric_difference(tree1, tree2), file=args.output)
except FileNotFoundError:
    print('Error: File not found. Please try again.', file=sys.stderr)
except ude.TreeError:
    print('Error: Provided file does not contain a tree. Please try again or provide
another file.', file=sys.stderr)
```

UserDefinedExceptions.py

```
# User-defined exception classes

class Error(Exception):
    '''Base class for other exceptions'''
    pass

class EmptyFileError(Error):
    '''Raised when input file is empty or whitespace only'''
    pass

class EmptySeqError(Error):
    '''Raised when sequence is empty or whitespace only'''
    pass

class FastaFileError(Error):
    '''Raised when file is not a Fasta file'''
    pass

class TreeError(Error):
    '''Raised when input is not a tree'''
    pass
```

run.sh

```
# Variables
node=$(pwd)

# Stop execution if error
set -e

# Create file structure
mkdir -p results && cd results
mkdir -p filtered_trees
mkdir -p unfiltered_trees
mkdir -p count_tables

cd $node/data/
start_trees=`date +%s`
for folder in */
do
    echo "Entering dir "$folder"..."
    cd $folder
    echo "Preparatory cleanup..."
    rm -rf __* *.py denoising *_trees/
    echo "Creating subfolders..."
    mkdir ${folder%/}"_trees"
    mkdir ${folder%/}"_u_trees"
    echo "Copying files..."
    cp -r ../../bin/denoising .
    cp -r ../../bin/UserDefinedExceptions.py .
    for file in *.msl
    do
        # Get unfiltered tree
        echo "Running fastprot on "$file"..."
        fastprot $file > $file"_fp"
        echo "Running fnj on fastprot outfile..."
        fnj -O newick $file"_fp" > $file"_tree"
        rm -rf $file"_fp"
        echo "Moving tree to unfiltered trees dir..."
        mv -f $file"_tree" ${folder%/}"_u_trees"
        # Get filtered trees
        echo "Running denoising on "$file"..."
        ./denoising $file > $file"_out"
        echo "Running fastprot on "$file"_out..."
        fastprot $file"_out" > $file"_fp"
        rm -rf $file"_out"
        echo "Running fnj on fastprot outfile..."
        fnj -O newick $file"_fp" > $file"_tree"
        rm -rf $file"_fp"
        echo "Moving filtered tree to filtered trees dir..."
        mv -f $file"_tree" ${folder%/}"_trees"
    done
    echo "Copying reference tree to subfolders..."
    cp -r *.tree ${folder%/}"_trees"
    cp -r *.tree ${folder%/}"_u_trees"
```

```
    echo "Moving subfolders to results directory..."
    mv -f ${folder%/}"_trees" ../../results/filtered_trees/
    mv -f ${folder%/}"_u_trees" ../../results/unfiltered_trees/
    # Cleanup
    echo "Cleaning up..."
    rm -rf denoising *.py __*
    echo "Exiting "$folder"..."
    cd ..
done
end_trees=`date +%s`
start_tables=`date +%s`
echo "Entering results directory..."
cd $node/results/
for folder in *_trees/
do
    echo "Creating subfolders in count_tables..."
    cd count_tables && mkdir -p ${folder%/} && cd ..
    cd $folder
    for subfolder in */
    do
        echo "Entering "$subfolder"..."
        cd $subfolder
        echo "Preparatory cleanup..."
        rm -rf differences* treecompare difftab *.py __*
        echo "Copying files..."
        cp -r $node/bin/treecompare .
        cp -r $node/bin/difftab .
        cp -r $node/bin/UserDefinedExceptions.py .
        for file in *_tree
        do
            # Compare acquired trees against reference tree, accumulate distances in a file
            echo "Comparing "$file" to reference tree and appending to differences..."
            ./treecompare --notext *.tree $file >> differences
        done
        echo "Creating database and running queries..."
        ./difftab differences > ${subfolder%/}"_differences.table"
        echo "Moving distance count table to count_tables..."
        mv -f ${subfolder%/}"_differences.table"
$node/results/count_tables/${folder%/}/${subfolder%/}"_differences.table"
        # Cleanup
        echo "Cleaning up..."
        rm -rf differences* treecompare difftab *.py __*
        echo "Exiting "$subfolder"..."
        cd ..
    done
    echo "Exiting "$folder"..."
    cd ..
echo "Done..."
done
end_tables=`date +%s`
runtime_trees=$((end_trees-start_trees))
runtime_tables=$((end_tables-start_tables))
echo "Total time spent denoising files: "$runtime_trees" seconds"
echo "Total time spent creating tables: "$runtime_tables" seconds"
```