Evaluating a Noise-reducing Algorithm for Multiple Sequence Alignments

Robin Andeer [randeer@kth.se](mailto:randeer@kth.se)  
Erik Fasterius [erik.fasterius@hotmail.com](mailto:erik.fasterius@hotmail.com)

# Introduction

Project 2 is about phylogeny and the possible merit of using noise-reduction algorithms on multiple sequence alignments for making phylogenetic trees. The test data contains several alignments over six combinations of different mutation rates along with being either symmetric or asymmetric, as well as six reference trees.

The multialignment is made up of columns of the different individual sequences aligned in the best possible way. A column is considered to be “noisy” if it either contains an indel (“-“) or more than 50% unique amino acids. An amino acid is unique if it occurs only once in the column.

# Materials and Methods

We started out writing a Python program that would read an alignment file and converts it to the Phylip file format: *convert2phy.py*. It utilizes BioPython to read and write the alignment to a new file, which we then can use in the other parts of the project.

We then moved on to the main file, the one that actually performs the noise reduction: *playwmsa.py*. This program has undergone several different iterations, especially in the noise-reducing algorithm itself, as we thought about different possible ways to do it. We also tried to keep in mind that the data set, although apparently shortened, is still very large and would take quite some time to go through, should our program prove too slow. Thus, we strove for the most efficient and fast program we could muster.

What the program has always done is to simply read the file that was created in *convert2phy.py* and manipulate it in various ways. It is comprised of different functions doing different things: *main* reads the file, *findIndels* finds and reports any indels in the current column and *checkAas* calculates and reports if the number of unique amino acids exceeds the allowed number. These two functions have not changed all that much since the start of the programming.

In the first variant of the program, our thinking went along the lines of *“check all columns with* findIndels *and* checkAas*, then build up a new variable from scratch using only the columns that passed the tests.”* For this, we used *for*-loops and built up a variable step-by-step inside them. One of the problems was that the data didn’t really handle as columns, but more like rows. The code looked something like this, but was never really finished:

In the next version, we though more along the lines of *“Check each column with* findIndels *and* checkAas *and remove any columns that don’t pass,”* hoping that this would be a more efficient way to do it. The problem of having the data as rows is now solved by a new function, *bildSeqCols*, which transposes them into more “true” columns, which greatly facilitated manipulating the data in this line of thinking. To circumvent the problem of changing indices after removing a column we go backwards, always removing the noisy column furthest back.

Then we had some problems in getting the structure correct for the program. We initially thought that it would be best to find some way to pipe all the different Python scripts together with *fastprot* and *fnj*, but that turned out to be very hard to do. In the ideal world we could avoid having to write every result to a file all the time, saving time. But, alas, that failed. So, the current structure of the program is a *shell*-script that loops over the different multialignments, performing the analysis on each (together with the appropriate reference tree) in turn (*convert2phy* 🡺 *playwmsa* 🡺 *fastprot* | *fnj* 🡺 *cmpTrees*) and finally writing the resulting data to the result file.

*convert2phy* writes to *outfile.phy*  
 *playwmsa* writes to *infile.fa*   
 *fastprot* | *fnj* writes to *treeout.txt* *cmpTrees* writes to *distances.txt*

We also perform the tree comparison with the reference and non-noise reduced alignments, which are the second column in the *distances.txt*, so that it is easily read alongside with the alignments that have been through the noise reduction.

The script will go through one folder per run, and we have to manually run it six times to get all six folders. We change the name of *distances.txt* each time, so that we keep the data that we already have. We name the new files to *distances\_a\_0.5.txt* and similar.

As for controls of our program, we started out simply by only using a very small dataset, a part of the asymmetric\_0.5 folder. We saw that it worked for that data, and slowly worked our way up towards the full data sets. We also check each record for being empty (less than 30), which would give an error. Comparing the reference tree to itself gives a result of zero difference, just as expected.

As a fun side-note, we did actually follow the course centerpiece in regards to the organization of our folders and data, which did help quite a bit.

# Results

These are the statistical results we obtained from analysing the data from our program using Excel.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Folder | Reduced average | Normal average | Reduced STDEV | Normal STDEV | Reduced recoveries | Normal Recoveries |
| Asymmetric\_0,5 | 7.25 | 7.63 | 2.92 | 2.91 | 1 | 1 |
| Asymmetric\_1,0 | 9.07 | 9.43 | 3.06 | 2.95 | 0 | 0 |
| Asymmetric\_2,0 | 11.21 | 12.34 | 3.30 | 3.48 | 0 | 0 |
| Symmetric\_0,5 | 4.21 | 4.16 | 2.33 | 2.35 | 24 | 23 |
| Symmetric\_1,0 | 4.89 | 5.01 | 2.49 | 2.54 | 18 | 17 |
| Symmetric\_2,0 | 6.15 | 6.96 | 2.94 | 3.19 | 12 | 4 |

# Discussion

So, we can more or less conclude that our noise-reducing algorithm is more or less useless. It seems to make more of a difference for the alignments with higher frequency of mutations, but it’s still very small. It recovered very few trees for the asymmetric alignments, which should make sense, since they were supposed to be the “more difficult” trees. There is no difference at all between the asymmetric trees, while the symmetric ones do show some small sliver of difference.

Our algorithm is very simplistic and brutal in its function: it removes every column that either has an indel or more than 50% unique amino acid residues. One could imagine a more efficient and sensitive algorithm that would perhaps function better than our does, but that would require some more expertise and knowledge within the field.

One thing to note is that our program does very little on terms of removing columns based on unique residues for the trees with lower mutation frequencies, since they won’t actually have “time” to get any unique residues. Therefore, the algorithm will more or less be totally based on the indels, for the lower mutation frequencies.

The only place where we noticed a difference in the number of recovered trees was the last symmetrical alignment (2.0), which had a difference of 8. We can’t really perform any kind of statistics on a data amount this low, though, but it’s still noteworthy.

Because of the brute force of our algorithm one could assume that the accuracy (the amount of columns that are not under selective pressure are removed) is pretty good whereas the precision (not removing columns that are under selective pressure) is very bad. This will likely affect the outcome of the comparison between the reduced trees with the reference tree so that the difference increases.