The Making of a Phylo Puzzle

Rudolf Lam
Instructor: Professor Waldispuhl

***Abstract*— puzzleGen is a puzzle extraction tool developed for Phylo to find interesting puzzles embedded in biological sequences. We developed efficient methods for computing interestingness score for all sub-blocks of an MSA and empirically found a metric for predicting the difficulty of generated puzzles.**

I. INTRODUCTION

Phylo is a platform for citizen science. On one hand, it serves as a tool for biologists to perform comparative studies on genomic data. On the other hand, it provides a leisure time activity for the public. Optimizing alignment data is a computationally expensive task for computers. However, for humans, with the visual aids and queues, the solutions for these tasks can be augmented rather inexpensively.

To convert a computational task to a game enjoyable by the public is not a trivial task. We must ensure that, not only is the task accomplished, but also fulfill its role as a game. Therefore, we must construct puzzles that are interesting and challenging, so that the public will continue to play the game.

In this paper, we will describe the how the puzzle is selected from a set of sequences, the measure of difficulty, and usage of the python application to generate the puzzles we will have defined.

II. MODELS

A. Puzzle

First, we must define what constitutes a puzzle in our model. Given a multiple sequence alignment M with M_i being the i-th sequence in M and $M_{:,j}$ being the j-th column M, we can define $M_{:,j:k}$ to be the block comprised of all columns of M from its j-th column to its k-th column. A block which is *"good"* based on its level of interestingness and difficulty will be considered a puzzle.

B. Interestingness Score

To measure interestingness of a block, we will consider the number of mismatches, matches, and gaps in the block.

$$\frac{n_{mismatch}}{n_{mismatch} + n_{match} + n_{gap}} \tag{1}$$

The $n_{j_{mismatch}}$ and $n_{j_{match}}$ are values relative the the most common variant of nucleotide in $M_{:,j}$. This is an heuristic already used in previous version of Phylo.

C. Difficulty Score

To measure the difficulty of a particular block, we use a simple linear mapping of the total phylogenetic branch length derived from our initial multiple sequence alignment.

$$difficulty = c \cdot L + b \tag{2}$$

The values of c and b are found empirically to be 2.0327509373773545 and -0.59127446582908016.

III. PIPELINE

A. Input

We have developed an application to pipeline the construction of puzzles from a set of sequences provided in the form of one of many different file formats like json, fasta, embl, fastq, fastq-solexa, fastq-illumina, genbank, imgt, phd, sff, tsv, and qual. The wide range of different input formats allows easy integration with other systems as well as interface with many other softwares.

B. Multiple Sequence Alignment

The input sequences need not to be pre-aligned. Our pipeline will strip input sequences of extraneous gap characters and then align the sequence using one of 3 methods; ClustalW, Muscle, and TCoffee.

Once aligned, the sequences will be searched for potential puzzle candidates that fit user-defined parameters like minimum size, maximum size, and minimum interestingness score.

C. Puzzles Discovery

When puzzles are found they will be written in their individual files named $[outputname]\_offset[a]\_length[b].fasta$ where a is the offset index of the puzzle relative to the alignment and b is the length of the puzzle. By constructing the file name this way, we can locate the puzzles simply from the file name and is potentially useful in integration in databases, where these values are used.

D. Outputs

Along side the main output files (the puzzles) there are auxiliary files produced. The first set of files produced are feature files. These files are named the same way as the puzzle files except their file extension is $.txt$ instead of $.fasta$. For each puzzle file created a feature file is also created. The feature file contains 6 features each, all of which describe the corresponding puzzle. These files

are used for machine learning purposes used for analysing the puzzles that are being generated, so that they can be improved. The 6 features included in the feature files are mean square difference in sequence lengths, number of mismatches, number of sequences, number of gap stretches, total phylogenetic branch length and Shannon entropy. The mean square difference describes potential the number of ways a sequence can be positioned relative to another. Sequences whose lengths are very different will have more ways to position themselves relative to each other than if those sequences were the same length. Thus, this will be a good feature for predicting the difficulty of a puzzle. Likewise, the Shannon entropy provides an estimate for the average uncertainty in the alignment.

Lastly, we have the difficulty file. This is an experimental value which we have yet to test in practice its actual validity. But results, show its performance should be promising.

IV. Interestingness Computation Method

An augmentation to the old system is the ability to generate and select puzzles using variable window sizes. Implemented in the pipeline is a dynamic algorithm to compute all interestingness scores of blocks of size s to t.

$$\begin{gathered} \forall s < k < t \\ \forall 0 < i < n - k + 1 \\ \forall 0 < j < n - k + 1 \end{gathered} \tag{3}$$

if $i \neq j$:

$$\begin{gathered} mismatches_{i,j} = mismatches_{i,j-1} + mismatches_{j,j} \\ matches_{i,j} = matches_{i,j-1} + matches_{j,j} \\ gaps_{i,j} = gaps_{i,j-1} + gaps_{j,j} \end{gathered} \tag{4}$$

otherwise:

$$\begin{gathered} mismatches_{i,j} = mismatches_{j} \\ matches_{i,j} = mismatches_{j} \\ gaps_{i,j} = gaps_{j} \end{gathered} \tag{5}$$

The $mismatches_j$, $mismatches_j$, and $gaps_j$ in Eq. (5) are found the in the manner described in previous section describing the interestingness score. First computing all values of mismatches, matches, and gaps and then aggregating the values into interestingness scores allow efficient computation of all values in $O((t-s)n^2)$. This is useful for searching through long sequences for puzzles and can provide puzzles of various lengths.

V. DATA

Aside from extracting features from generated puzzles. The pipeline also allow extraction of features from existing puzzles and alignments. For analyses of these data see Faizy Ahsan's project report.

VI. ANALYSES

It is found that out of the 6 features extracted from alignments in the database mean square difference, mismatches, and branch distances are the most predictive of difficulty level of an alignment. The difficulty measure defined in Ahsan's research is as follows:

$$difficulty = \frac{\textit{fail-count}}{\textit{play-count}} \tag{6}$$

In Eq. 6, the *fail-count* denote the number of unsuccessful attempt at augmenting a puzzle. Whereas the *play-count* is the total number of attempts made by all players.

From Ahsan's research, although mean square difference, mismatches, and branch distances are the most predictive their $R^2 \approx 0.23$ are still very low.

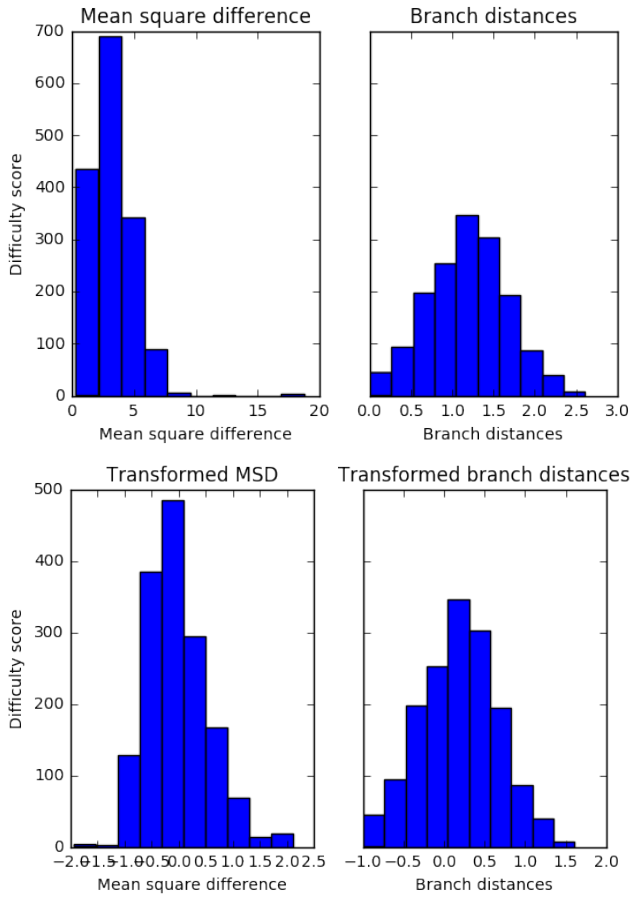

Fig. 1. Difficulty values distributed over the feature domains. The top pair of plots describe the raw distribution without modification. The bottom pair of plots described transformed values.

Looking at the raw data, see Fig 1., we can see that the difficulty values are somewhat gaussian in its distribution. Thus, it is possible to apply multilayer perceptron regressor (MPR) after normalization and transform. Likewise, we can transform our difficulty values into log difficulty values to make them more gaussian and centred around 0. But to no avail, the regression resulting from the MPR has very low R^2 as well.

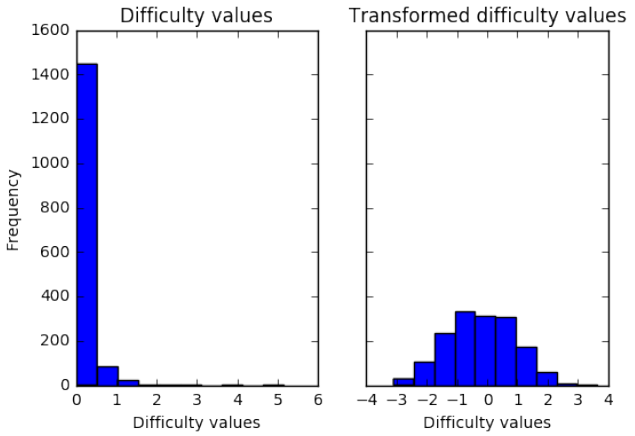

Fig. 2. Histogram for difficulty values and its transformed values.

Next, we can examine the correlation of each of this guassian distributions with that of the log difficulty score. In fact, this provides us with a decent correlation from which we can draw a predictor.

We can see, from Fig. 3, that the plot of transformed phylogenetic distances versus the log difficulty score provides a decent correlation. Using principal component analysis, we can find the eigenvector on which we can project all the data points. In this new coordinate system we see that we can describe the difficulty using a mapping from the transformed phylogenetic score to this new space. In the right image of Fig. 3, we see that the difficulty likes between a score of -4 to 4 in this new space.

A. Transformations

The transformations that we have performed to arrive at the values in the projected space is as follows:

Phylogenetic distances X:

$$X_T = X - \vec{1} \tag{7}$$

Difficulty scores Y:

$$Y_T = log(Y) + \vec{2} \tag{8}$$

Let A be the aggregate data matrix of X_T and Y_T concatenated horizontally. Let Q be the eigenvectors of A. Then the projected values are :

$$P = AQ \tag{9}$$

The points P are what we observe in Fig. 3. To derive the Eq. 2, we use Q_0, the eigenvector corresponding to the largest eigenvalue, and $\mu = ||A||_1/n$ where n is the number of rows in A. Values that lie on the principal component lies on the line coincident to the vector $Q_0 + \mu$, which is the line defined in Eq. 2.

VII. USAGE

The python application is called puzzleGen.py. Before it can be used, some dependencies need to be resolved:

- biopython

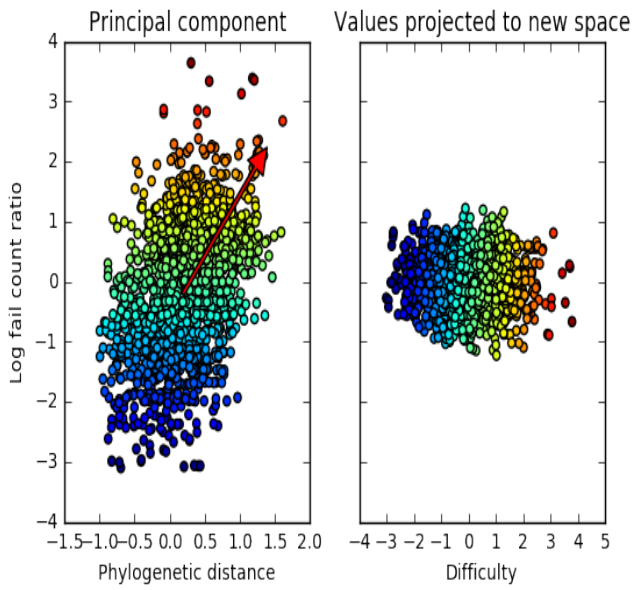

Fig. 3. Principal component analysis on the phylogenic distances and log difficulty score. The red arrow in the left figure shows the direction of the principal component. The right figure shows the data points projected on to the principal component with the points described in new coordinate system. The colours of the two set of points describe the difficulty score.

- numpy

Optional:

- The binaries for MUSCLE (included 3.8.31)
- T-Coffee installation (not included)

It was mainly tested on python 3, but it also works in python 2. However, because biopython does not work exactly the same in python 2 and 3 there are discrepancies in the results generated when performing the initial alignments.

The application takes as input two parameters; the input file name and the output file base name. All files outputted from the application will carry the file base name.

A. File Formats

There are many options that can be invoked. The first is the $-f$ option. This allows using, as input, file formats that are not fasta.

B. Alignment Algorithms

There are currently 3 algorithm implemented. By default, puzzleGen uses ClustalW, as this algorithm only requires biopython. With optional packages installed, puzzleGen can use MUSCLE and T-Coffee for its initial MSA. To use MUSCLE or T-Coffee, one must use the $-a$ option.

C. Threshold

The user can supply the application with a minimum interestingness score, under which puzzles will be rejected and not be part of the output. Interestingness score is between 0 and 1, where 1 is most interesting and 0 is least. By default, a threshold of 0.3 is used. At this interestingness score level, around 3 to 5 puzzles are generated for the sample alignments

of length 100. To apply an alternate threshold, one must use the $-t$ option.

D. Window Sizes

One can threshold for interestingness as well as the size of the puzzle. There are 2 parameters users can control; the minimum window size and the maximum. Window sizes outside the range will not be computed. The default sizes are 10 and 20 respectively for minimum and maximum. To customize the window sizes use the $--min\_size$ and $--max\_size$ options.

E. Difficulty

The $-d$ toggle will allow outputting a file containing the difficulty score described in Eq. 2.

F. Feature Extraction for Analyses

The application allows submitting sequences for feature extraction only. To use this functionality, one must use the $--feature$ option.

G. Miscellaneous

The are utility options as well like $--lsformats$ and $--lsalgorithms$ for listing all available file formats and MSA algorithms. For debugging, there is also the $-v$, $--verbose$, to increase the verbosity of the application.

H. Example

An example execution of the application is:

cat input.fasta | python puzzleGen.py output -d

The script is intended to be integrated as part of the Unix environment. Using pipe operator allows the forwarding of outputs from other applications directly into this application. However, one can provide the input as the first parameter as input for the program as well.

python puzzleGen.py input.fasta output -d

VIII. DISCUSSION

This application serve as a simple tool to process input data to Phylo and perform feature extraction for existing data. The difficulty measure for puzzles have yet to be tested and finalized. Thus, the feature extraction feature of the application may still serve to be useful in the future.

Due to time limitations, the application have yet to be linked with the database. However, due to how the output files are organized it is not difficult to be integrated with other modules in the system. Relevant information is stored in the name in each file to facilitate entry into database system.

IX. CONCLUSION

This paper described the inner working of the application puzzleGen. It included the design, implementation, and usage of the program.

In terms of design, we have described the workflow of the pipeline as well as the derivations for the equations that we have implemented . With regards to implementation, we looked at how the interestingness score is computed using dynamic algorithm. Finally, we have described in detail how to use the application.

Despite not having been able to find a good difficulty prediction model using machine learning techniques with the features extracted from existing data, we have an alternate means for computing the difficulty. Hopefully this serves as a starting point on which we can collect more data from players. And from the feed backs, we can modify our model to become more accurate at predicting the difficulty level.