A Co-Evolution Simulator (aCES) Command Line Interface, User Manual

From "aCES: A Co-Evolution Simulator generates co-varying proteins and nucleic acids"

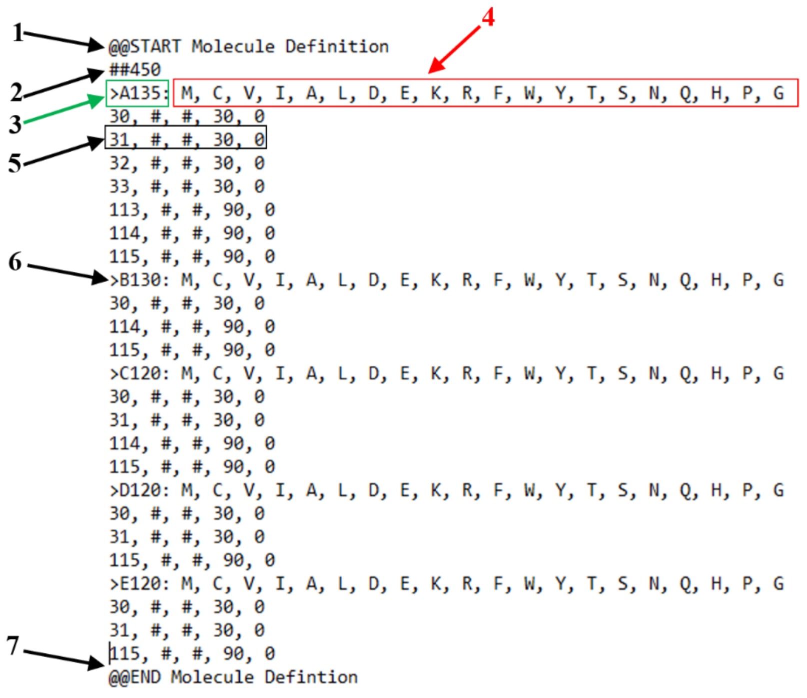
Authored by Devin Camenares
Assistant Professor, Department of Biochemistry
Alma College
614 W. Superior St, Alma MI 48801
camenaresd@alma.edu

Purpose:

This tool will take necessary parameters and generate up to five FASTA files that contain a simulated multiple sequence alignment. This alignment can be generated with a user-determined number of sequences, sequence and residue type, and the conservation of similarity and identity can (and must) be specified by the user. Most importantly, co-evolution constraints of both intra- and inter-molecule fashion can be applied to the generation of sequences.

Input:

This tool requires a text file with the parameters or constraints for the simulation. The input file provided, both the blank and the example, are annotated to help guide the user. **These annotations are provided before or after a block of code that is bracketed by "@@START .. " and "… @@END". Any text outside those brackets is ignored by the program; any text inside must conform to the expected input format.** For clarity, consider the beginning of the following example input file, which defines the nature of each molecule:



```
@@START Molecule Definition
##450
>A135: M, C, V, I, A, L, D, E, K, R, F, W, Y, T, S, N, Q, H, P, G
30, #, #, 30, 0
31, #, #, 30, 0
32, #, #, 30, 0
33, #, #, 30, 0
113, #, #, 90, 0
114, #, #, 90, 0
115, #, #, 90, 0
>B130: M, C, V, I, A, L, D, E, K, R, F, W, Y, T, S, N, Q, H, P, G
30, #, #, 30, 0
114, #, #, 90, 0
115, #, #, 90, 0
>C120: M, C, V, I, A, L, D, E, K, R, F, W, Y, T, S, N, Q, H, P, G
30, #, #, 30, 0
31, #, #, 30, 0
114, #, #, 90, 0
115, #, #, 90, 0
>D120: M, C, V, I, A, L, D, E, K, R, F, W, Y, T, S, N, Q, H, P, G
30, #, #, 30, 0
31, #, #, 30, 0
115, #, #, 90, 0
>E120: M, C, V, I, A, L, D, E, K, R, F, W, Y, T, S, N, Q, H, P, G
30, #, #, 30, 0
31, #, #, 30, 0
115, #, #, 90, 0
@@END Molecule Defintion
```

1. This line defines the start of the block defining all molecule characteristics. Anything before this line is ignored by the program.

2. After the ## the number of sequences for each molecule is specified. In this example, the program will 'evolve' 450 sequences each for molecules A, B, C, D, and E.

3. This begins the block defining molecule A – the number indicates the total length of the sequence/molecule.

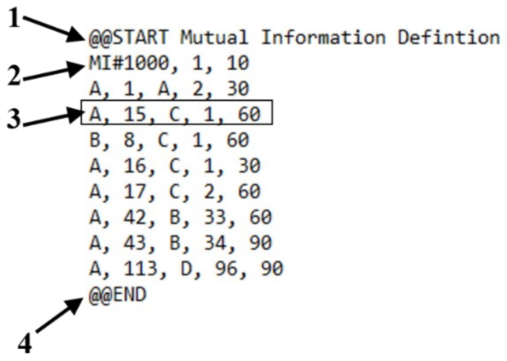
4. Following the colon in the molecule definition line is the grouping of residues that can be chosen from when creating the sequence. In this example, the simulation will create a protein, and treat each of the twenty amino acids (initially) as equal choices. For DNA or RNA, the input on this line should read as **"A, G, T, C"** or "**A, G, U, C"** respectively, in any order. It is also possible to group the amino acids or any other characters together. For example, the line "**MC, VIAL, DE, KR, FWY, TSNQ, H, P, G"** will create a two-dimensional array of choices. The first dimension of choice, in which there are 9 possibilities, relates to the chemical nature of the amino acid. The second dimension of choice governs the selection of the exact amino acid identity. For the purposes of simulating co-evolution, the program will only consider the first dimension – in this respect, D (Asp) and E (Glu) are treated as identical with that particular grouping.

5. This line indicates, separated by commas 1) the number of a given residue 2) the default identity choice, in the first dimension 3) the default identity choice in the second dimension 4) the percentage conservation, 0-100 in the first dimension, and 5) the percentage conservation, 0-100, in the second dimension. If a # symbol is present, the program will select a value at random from the available space.

6. The beginning of a block that defines molecule B – this follows the same format as was outlined for molecule A in items 3-5 above. This same pattern also applies for molecules C, D, and E.

7. This line defines the end of the entire block for defining each molecule. Any code after this line will be ignored by the program.

This is followed by a second block that defines the co-evolution constraints within and between the molecules simulated. Again, for the example input this appears as follows:



```
@@START Mutual Information Defintion
MI#1000, 1, 10
A, 1, A, 2, 30
A, 15, C, 1, 60
B, 8, C, 1, 60
A, 16, C, 1, 30
A, 17, C, 2, 60
A, 42, B, 33, 60
A, 43, B, 34, 90
A, 113, D, 96, 90
@@END
```

1. This line defines the start of the block for defining all co-evolution. Anything before this line is ignored by the program.

2. This line defines the overall parameters for how co-evolution will be simulated. The numbers after the MI# indicate the following, separated by commas: 1) The maximum number of iterations of adjustments to the distribution of a pair of residue identities, 2) the % threshold for a MI score to deviate from the target and be selected, and 3) The multiplier that is used to the random adjustment variable in each iteration. A larger number creates more volatile swings possible in each iteration (i.e. the multiplier for how many standard deviations from the mean value of a normal distribution will be sampled).

3. This line defines each specific co-evolution constraint. It indicates the following, separated by commas: 1) the first molecule in the co-evolving pair and 2) the residue in this molecule, 3) the second molecule in the pair and 4) the co-evolving residue in this molecule, and 5) the strength of the constraint, as a percentage of possible co-evolution from 0-100. Note that this is modulated by the overall conservation of the residues in question – a constraint of 100% co-evolution will actually produce very little co-evolution if each residue is already invariant (100% conservation).

4. This line defines the end of the entire block for defining co-evolution. Any code after this line will be ignored by the program.

** NOTICE: The program has not been exhaustively tested with incorrect input formats. It is likely to freeze or abort if incorrect inputs are provided. You are encouraged to use the debugging option to help identify the problem – please email your input, debugging logs and results of failed runs to camenaresd@alma.edu for additional assistance.

Once the input file contains your desired constraints in the correct format, you can proceed to load the program. The interface for this program is described in the following section.

Command Line Interface:

Each time the program is run, the user will be prompted to enter a filename. Simply entering the address of an input file (relative to the file of the compiled program – recommended to include those files in this same folder) will load the input, process the constraints, and save the files. The user will then be prompted to enter another filename.

It is also possible to specify parameters (shown below) by placing them after the filename, separated by a comma and space ", ". Note: your filenames should not contain this pair of characters! The parameters can be specified as follows:

Filename, parameter, parameter, etc

If one or more of the additional parameters is not specified, the default value will be used. Each parameter is defined as the type of parameter (id, output, phylogeny, or debug) followed by the value. For example:

> **output:common, phylogeny:50, debug:t**

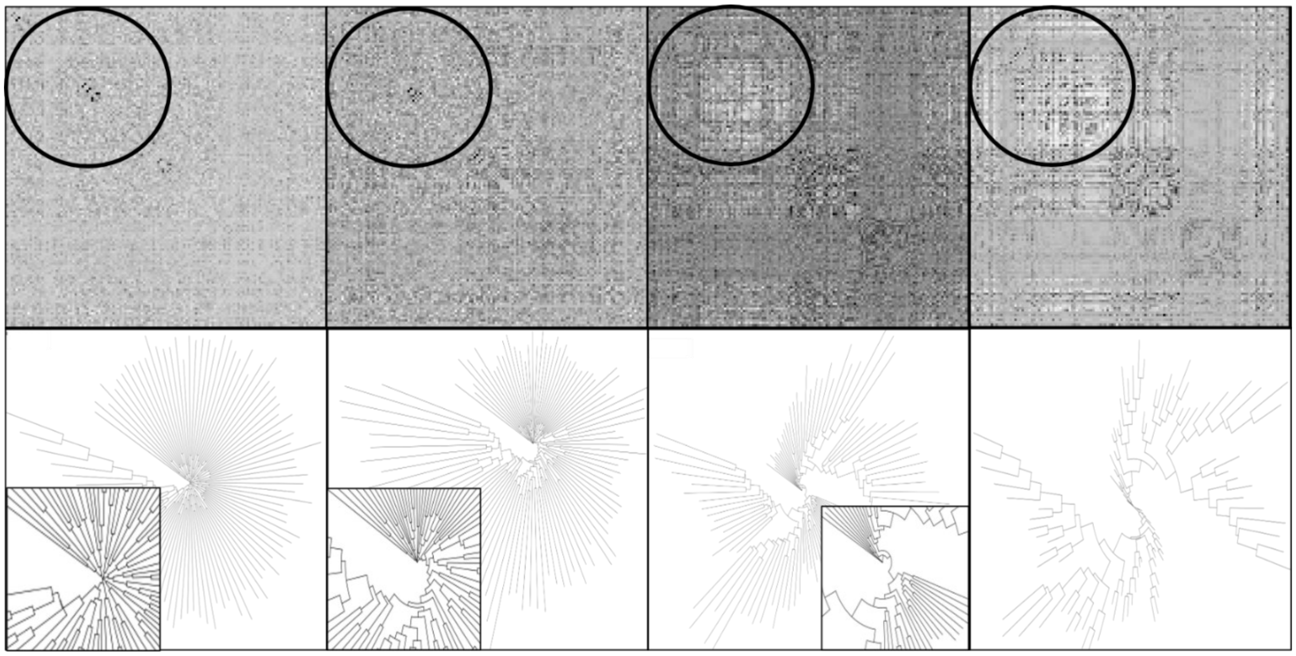
Not all of these parameters need to be defined (in the above example, **id** is not defined). They also do not need to be specified in a particular order. The options and effect of these are as follows:

- **(id) Job ID:** for this parameter, you can specify a particular name or number to be appended to the filename and directory. The default will be the timestamp generated by the program.

- **(output) Output:** for this parameter, you can specify the name of the directory for the output files. The default for this will be the jobID, but can also be the string/name specified here.

- **(debug) Debugging Mode:** here, users will indicate if they want the debug mode activated. If the value "t" is present, debugging mode will lead the program to output a file that details the steps being taken – this will report information in several stages:
    a. **New Molecule Created:** All the inputs will be read and values that are captured are reported line by line. This includes parameters that may not have been specified, in which case a random number will be chosen and displayed. [Category #1 is the first dimension of choice, Category #2 is the 2nd dimension. Thus, if you are using grouped amino acids, first dimension is the chemical nature of the amino acid and the 2nd dimension is the actual identity. The co-evolution algorithm ignores the 2nd dimension.]
    b. **Fill in unspecified positions:** similar to the preceding lines, but for any positions that were not explicitly defined.
    c. **MI pair:** for any specified co-evolution constraint, the target level of MI, as well as the final MI score and the number of iterations used to reach this score is displayed.
    d. **MI Needed!** Every time the program encounters a residue that participates in a direct co-evolution interaction, it will call upon the already established distribution to determine the probability of a particular residue being selected. It reports on the first line of this block the residues in question and the residue choice for the partner residue. The next three lines report the counts of residues it has available to choose from, with the total listed after the asterisk. The PbS-T line reports a probability cutoff array – the program then selects a random number from 0-100 and compares this against the array to select the residue, which is then displayed on the following line.

- **(phylogeny) Phylogenetic Weight:** Here, the user can specify, with a number from 0 to 100, the degree or percentage of the time they want a bifurcating phylogenetic structure to be mimicked. When the user provides residue conservation and co-evolution constraints, the program pre-loads (randomly generates within the constraints) a distribution for these residues across the entire set of 'organisms' to be simulated. By default, the program then randomly assigns these residues to each sequence, leading to a star phylogeny, in which each sequence is mostly independent of the others. In an effort to mimic the more realistic bifurcating phylogeny, the program can also be instructed to choose the pre-loaded residues in a particular order, which creates a greater degree of similarity between 'organisms'. The phylogenetic weight, specified here (and by default 0), is the percentage of the time in which the bifurcating instructions are followed. A demonstration of this effect is shown below:



In the above figure, four different settings for phylogeny are shown, indicating the % of time a bifurcating process was used. From left to right, each vertical panel is 0%, 50%, 80%, and 100%. The top panels show a heatmap for co-evolution, with a peak signal expected in the circled region (and very prominent in the 0% bifurcating setting). The bottom panels shown a phylogenetic tree for the sequences, with an inset zoomed onto the root or center of the tree.

The difference shown here is a drastic one – this is intentional, as a small sequence set makes the tree structure easier to visualize. However, a consequence of this is a strong obfuscation of the co-evolution signal.

The heatmaps were generated using my own mutual information program, and the bottom trees were generated using a combination of Clustal Omega (https://www.ebi.ac.uk/Tools/msa/clustalo/) and iTOL (https://itol.embl.de/).

Automating across many files:

Each time the program is run, it will continue to prompt for a filename or filename with parameters until a blank or null line is reached. This can be automated using some command line interface tricks. For example, if you have a large set of input files that you would like analyzed, you can generate a text file that has the address of all of these files, each on a separate line. Such a file might be named **directory.txt**, and point to all other files.

The program (once compiled) can then be run as follows: **java aces2 <directory.txt**

This will then take each line from **directory.txt** and use it to answer the prompt for filenames each time the program runs.