AlignmentProcessor1.4 Package

Shawn Rupp
Wilson Sayres Lab
Arizona State Univeristy
Shawn.M.Rupp@asu.edu

Updates in version 1.4:

If the kaks or codeml flag is given, alignments will automatically be converted into axt/phylip.
ConcatenateCodeML.p.y automatically determines input type

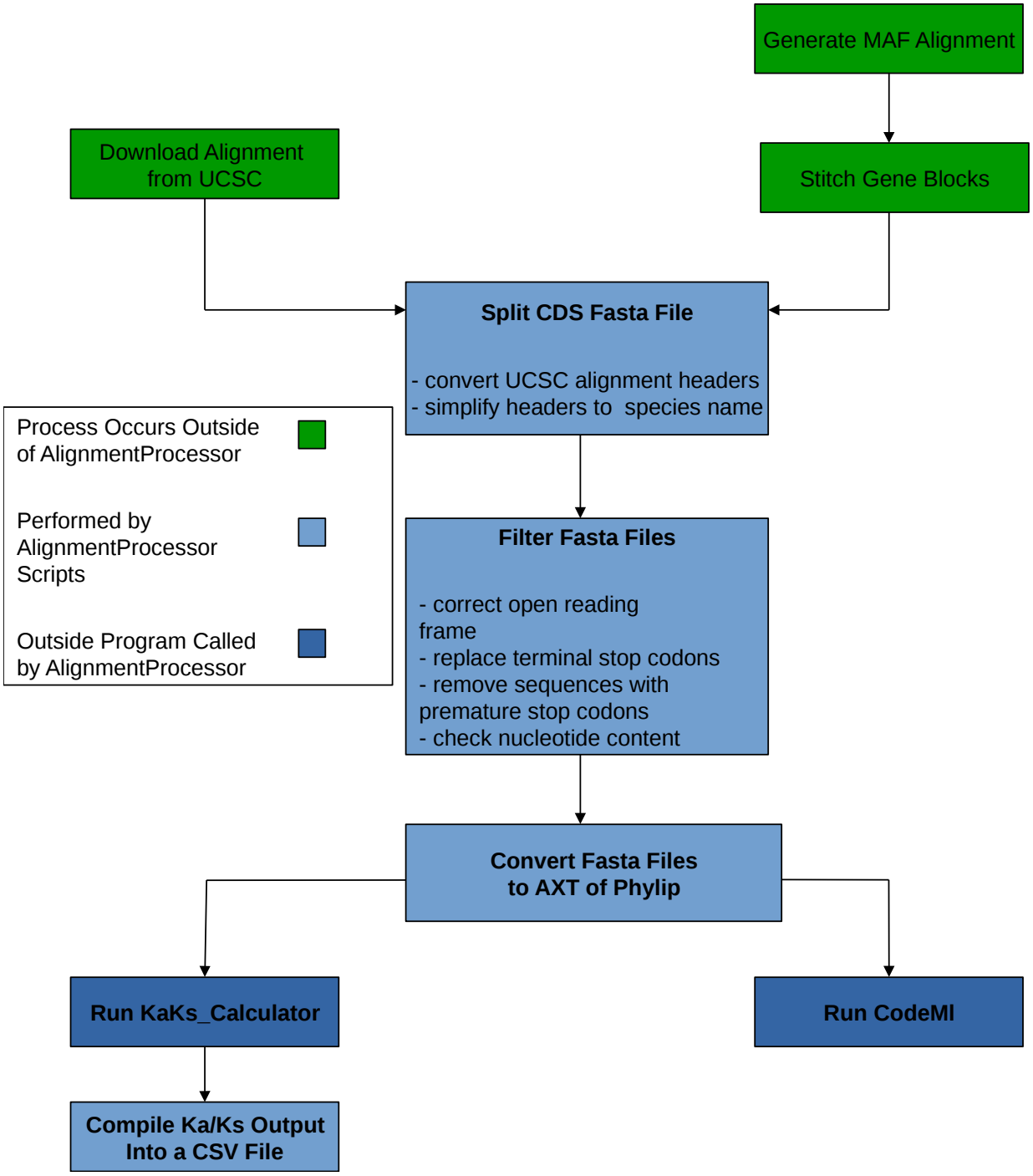
Contents

Introduction

AlignmentProcessor is a pipeline meant to quickly convert a multi-fasta alignment file into a format that can be read by KaKs_Calculator or PAML and optionally run those programs. When running CodeML, alignment processor willuse an input control file as a template to create a unique control for each gene. It will call PhyML to create a unique phylogenic tree for each gene based off of its sequences.

You can run the AlignmentProcessor wrapper which will call all of the python scripts in sequence, or you may call each script individually (See figure and page 3 for analysis pipeline).

Dependencies:
Python 3
Python 3 version of Biopython
PAML (if using CodeML)
PhyML (if using CodeML for multiple species alignments)


Generate MAF Alignment
Download Alignment from UCSC
Stitch Gene Blocks
Split CDS Fasta File
- convert UCSC alignment headers
- simplify headers to species name
Process Occurs Outside of AlignmentProcessor
Performed by AlignmentProcessor Scripts
Outside Program Called by AlignmentProcessor
Filter Fasta Files
- correct open reading frame
- replace terminal stop codons
- remove sequences with premature stop codons
- check nucleotide content
Convert Fasta Files to AXT of Phylip
Run KaKs_Calculator
Run CodeMl
Compile Ka/Ks Output Into a CSV File

1. Installation

GitHub
AlignmentProcessor is freely available on GitHub and can either be downloaded directly from the site, can cloned using the following command:

```
git clone https://github.com/WilsonSayresLab/AlignmentProcessor.git
```

Installing Biopython
Since Biopython offers the fastest and easiest ways to deal with alignments in a python script, it is used in this package. The easiest way to install Biopython is to download a Python 3 version Anaconda (Anaconda is incredibly useful in general, so it is a good program to have around). AlignmentProcessor was written in Python 3, but Python 3 will be installed alongside Anaconda, so you don't need to worry about that if you use Python 2.

Once Anaconda is installed, all you have to do is paste the following into a terminal and Anaconda will install Biopython for you:

```
conda install -c https://conda.anaconda.org/anaconda biopython
```

Alternatively, if you already have a Python 2 version of Anaconda installed, you can simply create and activate a Python 3 environment using the following:

```
conda create --name=py3 python=3
source activate py3
```
To deactivate, type:
```
source deactivate
```

KaKs_Calculator
AlignmentProcessor is packaged with KaKs_Calculator2.0 binaries for Linux and Windows, and a KaKs_Calculator1.2 binary for Mac (there is no 2.0 binary available for OSX). Before using, copy or move the appropriate binary for your system into the AlignmentProcessor bin which contains the python scripts.

PAML
If you plan to use CodeML, you must first download PAML and move the folder into the AlignmentProcessor directory. Make sure that the folder is titled "paml".

PhyML
If you plan to use CodeML, you must also download PhyML. Similar to PAML, you must move the folder into the AlignmentProcessor directory and change the name of both the folder and the binary for your operating system to "PhyML".

2. Obtaining a fasta alignment

UCSC Fasta Alignment
It is possible to download CDS fasta alignments from the UCSC Table browser. This does, unfortunately, limit you to currently available alignments. If they do have the alignment that you are interested in, however, it will be faster to download it rather than generate a new one. While previous versions of AlignmentProcessor have required the user to specify whether they were using an alignment from UCSC, AlignmentProcessor1.1 will automatically detect the source of the file and make any necessary conversions.

User Generated Alignments
Since most alignments are in maf format, you will have to convert your alignment from maf to fasta. There seem to be very few programs that can do this; fortunately Galaxy's Stitch Gene blocks not only converts a maf to fasta, but it also separates sequences by genes, which is something we need to do anyway.

If you did not use a UCSC genome for the reference species in your alignment, you may need to upload the reference genome that you used as a custom build. Make sure that the genome, maf file, and BED file are all set to the custom build, and that the reference species build name is identical in all three files.

Additionally, you may not be able to use the UCSC BED12 file if you did not use a UCSC genome. If that is the case, you can either upload your own, or, if you used an Ensembl genome, you can just remove the "chr_UN" and "chr" chromosome prefixes from the file, and resubmit the file to Galaxy. For NCBI genomes, you may download the gff from NCBI genome, use the UCSC utility "gff3ToGenePred" with the -useName and -honorStartStopCodons options, and use the UCSC utility "genePredToBed". This will return a BED12 file which may be submitted to Galaxy.

Upload your maf and BED files to Galaxy (usegalaxy.org) (or retrieve a BED file of the genes for your reference species using the UCSC Main link under the Get Data tab on the left of the screen). Select your species from the UCSC table browser, select Genes and Gene Predictions, Ensembl genes, and the whole genome. Select BED as the output format and check the send to Galaxy box.

Once you have your data uploaded to Galaxy, select the Stitch Gene Blocks tool under the Fetch Alignments/Sequences tab. Select your reference species' genome BED file in the Gene BED file drop down menu. Change MAF Source to the maf file you uploaded (you may also use a locally catched alignment if it is available for your species of interest). Select the desired species IDs, leave Split into Gapless MAF Blocks to "no" (AlignmentProcessor will deal with gaps later on), and hit "Execute." When the tool has finished running, download the output file. This process will take a few hours, so plan accordingly.

3. Running AlignmentProcessor

AlignmentProcessor is designed to convert the file into a usable format and run the substitutions quickly, so everything can be run with one command. Each script can be run individually if necessary (each script's function and options will be discussed later).

To execute the AlignmentProcessor pipeline, you must first change into the package directory. Otherwise it will not be able to locate the scripts in the bin/ directory.

If you are running CodeML and the program is interrupted, you may call the 07_CodeMLonDir.py script and it will continue where CodeML left off. This will save the time of having to run those genes through CodeML again (It will do the same thing if you call the entire pipeline again, but there is no need to re-run the previous steps). It will not, however, do the same for KaKs_Calculator since KaKs_Calculator is much faster, so it should not be a problem to invoke KaKs_Calculator on the whole directory again.

Example Usage

```
python AlignmentProcessor.py --axt/phylip --kaks/codeml -r <reference species> -i <input fasta file> -o <path to output directory>
```

Required Arguments:

-i the path to your input fasta alignment.
-o the path to your working/output directory.
-r the build or common name of your reference species (more below).

Optional Arguments:

-h/--help will print the program's help dialogue

--retainStops This will tell AlignmentProcessor to retain sequences that contain internal stop codons. By default, sequences with internal stop codons will be removed from the analysis as they may bias the results.

-p A decimal value specifying the minimum percentage of reads that must remain after replacing unknown codons with gaps (Default = 0.5). You may wish to use a lower threshold for highly diverged species or for low quality genomes.

--axt/phylip Specifies which format to convert the files into (axt format for KaKs_Calculator; phylip for CodeML, HyPhy, etc.). This is only required if CodeML and Ka/Ks_Calculator is not being run.

-t Number of CPUs to run CodeML, PhyML, and/or Ka/Ks_Calculator. AlignmentProcessor will call multiple instances of the target program to shorten overall run time. (Default = 1)

KaKs_Calculator

--kaks Will convert files to axt and run KaKs_Calculator. Otherwise the program will quit after converting the files.

-m Indicates the method for KaKs_Calculator to use to calculate substitution rates (see below).

CodeML

--codeml Will convert files to phylip and run codeml on all of the files in the 03_phylipFiles directory. You must also supply a control file for CodeML (see below).

-f The first ten characters (standard phylip format truncates the species names to ten characters) of the build name of the species on the forward branch of the phylogneic tree supplied by PhyML. This species does not have to be the same as the reference species.

-n The path to a phylogenic tree in Newick format. If provided, AlignmentProcessor will use this tree for CodeML instead of calling PhyML.

--cleanUp Tells the program to remove temporary CodeML control files, tree files, and other PhyML and CodeML output/temporary files. These are are retained by default since they may be useful and AlignmentProcessor can reuse the tree file generated by PhyML.

KaKs_Calculator Method

KaKs_Calculator can calculate substitution rates in a number of different ways which can be specified to AlignmentProcessor using the "-m" flag. These methods include estimations, that are generally much faster, and maximum liklihood models, that should be more accurate. See the KaKs_Calculator documentation in the KaKs_Calculotor folder for more information.

Estimations
NG (default in AlignmentProcessor)
LWL
LPB
MLWL
YN
MYN

Maximum Liklihoo
GY
MS (recommended)

The CodeML control file

Codeml requires that all of its parameters be specified in one control file. Provide a control file with your desired parameters and AlignmentProcessor will use it as template.

The control file must have a ".ctl" extension, and it must be located in the output directory. Only provide one control file in this directory. Examples are included with PAML and the controlFiles directory contains example control files for branch site, branch specific, and pairwise analyses. These can simply be copied into your output directory or you may supply one of your own.

Invoking the Ka/Ks pipeline with a UCSC alignment

```
python AlignmentProcessor0.21.py --axt --kaks -r anoCar2 -i
anolis_gallus.fa -o pairwiseKaKs/
```

Invoking the CodeML pipeline

```
python AlignmentProcessor0.21.py --phylip --codeml -p 0.4 -r anoCar2
-f anoCar2 -i anolis_gallus.fa -o codemlOutput/
```

4. Individual Scripts

Each script performs a core set of functions on the input file or files and saves the output to a new subdirectory.

01_SplitFasta.py
This script will split the input multi-fasta alignment into one file per gene. Each new file will be saved with the transcript ID of the reference species as the file name and the headers in the file will only contain the species' genome build names. It will produce an output file for a gene if it has at least two sequences.

-h, --help	show this help message and exit
-i	path to input file.
-o	path to output directory.

02_FilterFasta.py
This script removes gaps introduced in the reference sequence by the alignment and removes corresponding sites in other species. It assumes that the reference sequence was in frame before any gaps were inserted, and it returns the reference sequence to its original open reading frame. It will then replace codons with missing nucleotides with gaps to remove unknown amino acids from the sequence. Next, will remove the internal stop codons (TAA, TAG, TGA) from the nucleotide alignment and replace them with gaps (---). Some programs will not run properly if they encounter a premature stop codon. Terminal stop codons will be replaced, while sequences with internal stop codons will be removed and have their gene id and sequence name recorded in the filterLog.txt file by default. If --retainStops is specified, these sequences will be retained. Lastly, it will check a multiple FASTA file to see that each species retains a certain percentage (50% by default) of its nucleotide sequence. If not, it will remove that sequence.

-h, --help	show this help message and exit

-i	path to input file.
-o	path to output directory.
-r	the build or common name of your reference species.

03_covertFasta.py
This script will convert all fasta files in a directory to axt or phylip format.

-h, --help	show this help message and exit
-i	path to input file.
-o	path to output directory.
--axt	converts fasta files into axt format
-- phylip	converts fasta files to phylip format

04_CallKaKs.py
This program executes KaKs_Calculator on every file in a directory and concatenates the output from KaKs_Calculator into a csv file. It adds a column for gene (or sequence) IDs, and prints the ID from the filename.

-h, --help	show this help message and exit
-i	path to input file.
-o	path to output directory.
-m	method for calculating Ka/Ks.

04_CallCodeML.py
This script will run codeml on every file in a directory. It requires a codeml.ctl file. It will overwrite the "seqfile", "treefile", "outfile" lines include the paths to the input phylip file, the output file, and the tree file. It will also run PhyML to create the tree file. If you are running CodeML and the program is interrupted, you may invoke this script to pick up where you left off.

-h, --help	show this help message and exit
-i	path to input file.
-o	path to output directory.
-t	number of threads (and number of parallel instances of CodeML).
-f	name of forward branch of phylogenic tree.
-n	The path to a phylogenic tree in Newick format (optional).
--cleanUp	removes temporary files.

ConcatenateCodeML.py
This script is not called by the AlignmentProcessor wrapper, but has been provided to provided a basic summary of CodeML's output. Since CodeML performs several analyses, it is difficult to provide a summary specific to every user's needs. This script will output a csv file containing each sequence's transcript ID, the tree-wide dN, dS, dN/dS, tree length, and the number of sequences remaining after filtering (for multiple alignments).

-i	path to input directory.
-o	path to output file.

5. Outputs

A directory is created for each step, and, prior to the file conversion step, each directory will contain a series of single-gene fasta alignments. These are retained after the programs finishes in case you need to examine them at any point.

If --axt is selected, the 03_axtFiles directory will be populated with axt files for use with KaKs_Calculator. If --kaks is specified, KaKs_Calculator will run and its individual output files will be placed in the KaKsOutput directory. Finally, a single csv file containing all of the output of KaKs_Calculator will be printed to the output directory that you specified with the -o option.

If --phylip is specified, the program will convert the fasta files to phylip after the stop codons have been removed. If you also specified --codeml, AlignmentProcessor will edit and submit the control file to CodeML and the output files will be saved in 04_CodemlOutput. Since there are many different things that can be done with the codeml output files, AlignmentProcessor does not attempt to concatenate specific parts from the output files, although ConcatenateCodeML.py has been provided to provide a basic summary of the output.

6. Run AlignmentProcessor on test data

To test KaKs_Calculator:
Change directory into the AlignmentProcessor folder. Paste the following into a terminal:

```
python AlignmentProcessor.py --axt --kaks -r anoCar2 -i
test/kaksTest.fa -o test/
```

This should return a csv file with 11 lines.

To test CodeML:
The test directory already contains a sample CodeML control file, so all you need to do is change into the AlignmentProcessor directory and paste the following:

```
python AlignmentProcessor.py --phylip --codeml -t 2 -r anoCar2 -f
anoCar2 -i test/codemlTest.fa -o test/
```

There should be 6 .mlc files in the 04_CodemlOutput directory.