Pipeline documentation

The pipeline is a solution to the need to examine and compare various methods for the partitioning of gene sequences, and how well they perform with various phylogeny-inference methods. The current version is capable of executing five programs: PartitionFinder and MtPAN for partitioning, and RaxML, PhyloBayes and TreeCollection for phylogeny inference. However, there are several 'helper' programs and scripts used for creating the inputs necessary for the use of these main programs. This document will list these and briefly explain what they do. It's also important to note that the pipeline uses these programs with defined parameters, and there are ways of modifying these for future analyses.

Input

To run the pipeline, only the script Pipeline.py needs to be executed. The script itself takes no inputs, but it does assume a certain directory structure and the presence of certain files:

Directory structure:

Data: All files (OGs) containing the gene sequences should be kept in a single folder. At the moment this folder is assumed to have the name 'Data', but this can be changed by modifying the constants.py script. The pipeline is designed for running sets of analyses multiple times, containing each set of results within its own folder ('sample folder'). Pipeline.py creates the sample folders in Data, and the subfolders containing the results for separate analyses.

In the same folder as Pipeline.py must also be a folder called "external_tools", which should contain the scripts: "nexus_to_phy_sequential.py", "PartitionFinderProtein.py", "PartitionFinder.py", max_monophly.pl (assumed to be in a subfolder called 'monophyly') and "run_tree_collection.py". Of course which of these you need will depend on the analyses you wish to run.

Necessary scripts and modules

The pipeline itself has had its stages divided into several Python scripts. The main script is 'Pipeline.py', which is the one to be executed. The others are:

job_file_methods.py - contains the methods used to create and submit jobs to the cluster. These jobs handle separate programs for use on separate samples of data.

concatenate.py - converts all alignments into NEXUS format, and for each sample combines these into one NEXUS alignment.

constants.py - defines the values of the essential variables: the number of samples and their sizes, the directory in which the pipeline resides, and the list of FASTA alignments in each sample.

clusters_to_partitions.py - creates RaxML-style partition files from the partitions produced by MtPAN.

make_partition_file.py - creates RaxML-style partition files from the partitions produced by PartitionFinder

map_names_to_species.py - Changes the mapped names (seq1, seq2, etc…) in a tree to the original species names. The mapping is specified in the text file created by nexus_to_phy_sequential.py.

partition_by_genes.py – Takes a NEXUS alignment and produces a RaxML-style partition file, with each OG being a partition and using WAG to estimate model parameters.

monophyly_scores.py - Takes (in a text file) a list of the names of monophly results, and produces a ranking, in descending order, of their scores as a percentage.

to_nexus.py - Converts a tree into Newick format for use by the monophyly script max_monophly.pl

Scripts needed for running a TreeCollection analysis:

root_tree.py - Used for creating a bifurcating tree.

make_OG_lists.py - goes through each sample folder, and for each one produces a text file with a list of alignments in that folder. This list is required as input for create_treecoll_input_aligned.py.

create_treecoll_input_aligned.py – Takes the text file produced by make_OG_lists.py and creates the distance-variance matrix, mapping and labels from the alignments listed.

average_matrix_distvar_WEIGHTED.py – makes a distance-variance matrix combined from weighted distance variance matrix. The inputs for this script are the outputs from create_treecoll_input_aligned.py, plus the name you wish to give the resulting matrix.

run_tree_collection.py - Uses a wrapper to execute Tree Collection. Requires a folder name as input, this is where your outputs from create_treecoll_input_aligned.py and average_matrix_distvar_WEIGHTED.py will be stored, in addition to the resulting tree.

Running the pipeline:

Running the the pipeline is straightforward. To execute Pipeline.py, use the command "python Pipeline.py" in the same directory as Pipeline.py. You will then be given prompts asking you which analyses you wish to run:

```
dhcp-41-247:Pipeline Lovecraft$ python Pipeline.py
Run PartitionFinder? (y/n): y
Run TreeCollection? (y/n): y
Run RaxML on MtPAN partitions? (y/n): n
Run RaxML on PF partitions? (y/n): y
Run MtPAN? (y/n): n
You won't be able to run RaxML on MtPAN partitions since MtPAN won't run.
Run RaxML vanilla? (y/n): y
Run PhyloBayes? (y/n): y
```

Type in 'y' or 'Y' for yes and 'n' or 'N' for no. If you select no for MtPAN or PartitionFinder, you'll be given a message saying you can't run RaxML with MtPAN or RaxML with PartitionFinder, respectively.

The pipeline handles relatively little itself – its main job is to create and submit the job files which are used for executing programs by the cluster. Your connection to the cluster will only be maintained so long as alignments are being combined and converted into PHYLIP format (if you have chosen for this to be done).

To record the progress of your jobs, one text file for each sample will be created and updated during computation, to show which analyses have been completed for that sample. A list of trees that have been completed so far will be in each sample folder too.

Outputs

In each sample the output will be:

- OG_list.txt, a list of aligned OGs.
- sample.alignment.nex – concatenated and aligned sequences, in NEXUS format.
- sample_alignment.phy - combined alignment of sequences, in PHYLIP format. In this file the species names are mapped to substitute names since many exceed 10 characters long.
- sample_alignment.nexSpeciesMapping.txt – Contains the species-substitute name

mapping , e.g.:

```
seq0    Catenulida_fasta_cd.fa
seq1    Echinoplana_celerrimaclean_cd.fa
seq2    Maritigrella_cd.fa
```

The trees for each program will be in their corresponding folders, along with logs recording the run of those programs. There will also be a text file showing the results of the monophyly test used.

Post-analysis:

To compare the methods used in each sample, the pipeline automatically stores the full path of each text file holding the monophyly results for some tree. In each sample the file these paths are stored in is called “mon_results.txt”. After all analyses have finished for all samples, run the script “produce_all_scores.py”, which will put in each sample folder a text file called “mon_percentages.txt”, which lists the monophyly scores for all trees in descending order. These can then be examined to compare the relative accuracy of phylogeny methods.

There is also a script, mean_var.py, which uses the time files in the “times” folder to print to the screen the mean and variance of the times taken by each method, over all samples. The output times will be in minutes, whereas the raw scores recorded in the time files are in seconds.

Parameters used:

The parameters used for the programs can be found in the job_file_methods.py script.

In “constants.py”, you can alter constants like the sample size, number of samples, and directory in which Pipeline.py is stored.

To change the parameters used for programs like RaxML and PhyloBayes, “Pipeline.py” and “job_file_methods.py” are the scripts to alter. To alter the number of threads for their jobs, change “Pipeline.py”. For parameters such as substitution models, random seeds and tree names (as in RaxML), alter the relevant methods in “job_file_methods”. For Phylobayes, a ‘chain’ analysis is used, in which separate successions of trees (chains) are generated until they are similar enough to be sure an accurate phylogeny can be made. The parameters for this analysis are in the “phylobayes_job” method.