Quoll Transcriptome Alignment Analysis Package

Version 0.2

Contents

Introduction

Quoll is a series of scripts for filtering and analyzing transcriptome alignments. It is meant to facilitate the analysis of transcriptomes assembled with Trinity and aligned with ProgressiveCactus (although Lastz/MultiZ alignments can also be used). Fasta transcriptomes with NCBI refseq IDs may also be used.

Dependencies:
Python3
Cython
TransDecoder

(By the way, the quoll is the second largest marsupial carnivore after the Tasmanian devil.)

Installation

Cython
Several of Quoll's core scripts utilize Cython, which compiles python code into C and drastically improves performance. Cython can be installed from the pypi repository or via Miniconda (it is installed by default with the full Anaconda package).

To install with Miniconda:
```
conda install cython
```

Quoll
Download the repository:

```
git clone https://github.com/icwells/Quoll.git
```

Most of the scripts are written in python3, but several contain Cython modules which must be compiled. Additionally, Quoll requires TransDecoder to identify open reading frames within transcripts. Running the install script will compile the Cython scripts and install TransDecoder from GitHub:

Change into the Quoll directory and run the install script:

```
cd Quoll/
./install.sh
```

Getting Started

Aligning Transcriptomes
Quoll was written to process alignments in which at least one of the input fasta files is a *de novo* transcriptome assembly (it is better to do a whole genome alignment if possible). If you have a *de novo* transcriptome assembly, additional transcriptomes can be *de novo* assembled or can be subset from genomes. A fasta genome and a gff/gtf annotation is required to extract a fasta transcriptome from a genome. If you have both, this can be done with gffread from the Cufflinks package. Once you have all of your desired transcriptomes, the easiest way to align them is with ProgressiveCactus. Fortunately, this takes far less time than aligning the equivalent genomes would. Cactus produces alignments in hal format, but it does include a script for converting hal files to maf format.

Filering Alignments for Gene Content
After creating an alignment in maf format, first run mafTransToFasta.py which will extract all DNA sequences over 60 bp from the maf file.

```
cd bin/
python mafTransToFasta.py {path to input file}
```

If you are only interested in the terminal nodes of the alignment (i.e. the transcriptomes you supplied to your aligner), removeAncestor.py can be used to remove ancestral sequences from the alignment. If you are using a MultiZ alignment, you may script this step since it does not infer these sequences. Likewise, if you are interested in ancestral nodes you will want to keep these sequences.

```
python removeAncestor.py {path to input file}
```

Lastly, filterCDS.py can be used to extract ORFs from the aligned transcripts. First, TransDecoder must be called on the output of the previous script to identify open reading frames. Export the path to TransDecoder and change into the output directory (TransDecoder writes output to the working directory).

```
export PATH=$PATH:{path to TransDecoder directory}
cd {output directory}

TransDecoder.LongOrfs –t {input file}
TransDecoder.LongOrfs –t {input file}
```

filterCDS.py can now be called to convert the TransDecoder output back into a fasta alignment. The input file for this script (and most of the remaining scripts) is the same input used for TransDecoder (its output will be inferred from the input file name). This script only needs the transdecoder.cds file, so all other TransDecoder output can be deleted if desired.

```
cd ~/Quoll/bin/
python filterCDS.py {path to input file}
```

The remaining scripts can then be called in any order for your analysis.

Scripts

mafTransToFasta.py
This script will pull DNA sequences from aligned blocks within a maf file and writes them in fasta format. Any sequences under 60 nucleotides in length will not be written since the shortest known protein is 20 amino acids long. The output file will be printed in the same directory as the input.

```
python mafTransToFasta.py {path to input maf}
```

removeAncestor.py
This script will remove ancestral sequences which were inferred by alignment programs such as ProgressiveCactus.

```
python removeAncestor.py {path to input fasta alignment}
```

filterCDS.py
This script converts TransDecoder output into fasta alignment format for each aligned block. This is a fairly simple technique for extracting ORFs and will be replaced with a more sophisticated algorithm in the future.

```
python filterCDS.py -p {minimum nucleotide content} {path to input fasta alignment}
```

alignedGenes.py
This script creates a csv table of equivalent gene IDs and any associated transcipt IDs for each alignment block. It also summarizes the total number of genes and transcripts found in each block.

```
python alignedGenes.py {path to input fasta alignment}
```

sampleMatrix.py
This script produces a csv matrix summarizing the number of genes aligned between each species/sample in the alignment. If "--combinations" is specified, it will return a matrix of the number of genes uniquely aligned between eavery possible combination of samples.

```
python sampleMatrix.py --combinations {path to input fasta alignment}
```

subsetAlignment.py
This script will subset aligned blocks if they have the specified number of samples aligned or if they contain genes from specified samples. The "--unique" flag indicates that blocks with only the given samples will be selected. Otherwise, blocks that contain at least those samples will be selected.

```
python subsetAlignment.py -n {# of samples to select} -i {path to input fasta alignment}
python subsetAlignment.py --unique -s {comma seperated list of samples to select} -i {path to input fasta alignment}
```

extractSequences.py
This script will extract one sequence from each aligned block. If no sample ID is given, the sequence will be chosen at random.

```
python extractSequences.py –t {sample to extract} –i {path to input fasta alignment}
```

blastToCSV.py
This script will combine blastn, blastx, and blastp results (in outfmt6 format) into a csv file.
Input files must be titled blastn.outfmt6, blastx.outfmt6, and blastp.outfmt6.

```
python blastToCSV.py –i {path to directory containing blast results} –o {path to output file}
```

annotateFasta.py
This script will insert protein and gene IDs identified by blast and add them to corresponding headers in a fasta transcriptome.

```
python annotateFasta.py –i {path to input fasta transcriptome} –a {path to blastToCSV annotation}
```