Reads2seqs pipeline. v1.0, 1/1/15

Please see Heyduk et al 2015 (in review) and Stephens et al 2015 for example of data produced by workflow.

This "pipeline" consists of a number of wrappers and batch submission scripts to use already existing programs. Please remember to cite those individually! If you use this start to finish, you should have no need to rename files, etc, between scripts. It is not even necessary to first rename read files, as the first step – read cleaning – will rename files with something a bit more comprehensible.

Please note: this pipeline assumes you have access to a multi-node computing cluster. You will have to modify most scripts so that the submission to the cluster is appropriate for your setup.

Clean reads

Illumina reads have a significant quality drop off at the end of the read. Luckily it's straightforward to clean the reads. **batchCleanReads.pl** is run like so in the directory where your raw reads are:

perl batchCleanReads.pl adapters.fasta library_index.txt

where adapters.fasta is a fasta file of adapter sequences that you would like removed from your reads (an example is available from the github page for this pipelie), and library_index.txt is a two column file (tab delimited) that matches a shorthand ID to each read file. For example:

libA	AACTGAGC_illumina_1.fastq
libA	AACTGAGC_illumina_2.fastq
libB	TTAGCTAA_illumina_1.fastq
…etc

Every ID should have two files.

Trimmomatic options default to: ILLUMINACLIP:$adapters:2:30:10 LEADING:10 TRAILING:10 MINLEN:40." To see other options see the Trimmomatic manual. You will have to manually modify the script to change options. This will output four files per library: libID_paired_R1.fastq, libID_unpaired_R1.fastq, libID_paired_R2.fastq, libID_unpaired_R2.fastq, where libID is the first column of the library_index.txt file.

Assembly

Trinity – We're going to treat our assembly as *de novo* assembly using Trinity. Trinity is originally developed for RNA-sequencing data – meaning it assumes the assembled contigs will be wildly variable in coverage, and no large scaffolds should be created. For this reason, Trinity works well for gene capture data, because the data will be variable in coverage due to hybridization efficiency/copy number, and because the pieces will be in fragments consisting of exons surrounded by 100-200bp of intron.

This script will create a separate folder for each library based on libIDs.

Step by step to run Trinity:

1) Make a folder called “trinity” somewhere in your working directories
2) When you run batchTrinity.pl, it needs the following arguments (provide them without the < >):

    perl batchTrinity.pl </full/path/to/trinity/folder> </full/path/to/reads> <CPUs> <libIDs.txt>

CPU refers to how many cores you want Trinity to run on – if you have many available, use 4-8.

The libIDS.txt file is just a list of all the libIDs (the first column of the library_index.txt file, but without each name duplicated):

libA
libB
libC…

The script as it is currently set up to use the output of trimmomatic. It looks for the libID_paired and libID_unpaired files. If you have deviated from this, realize the script either needs to be modified, or your read names need to be modified.

Trinity manual is here: http://trinityrnaseq.sourceforge.net/

CAP3 Trinity

Trinity likes to make isoforms, which is great when you’re worried about alternative splicing or expression variants in RNA seq. It’s not so great, and doesn’t necessarily make sense, with gene capture data. In this data, isoforms represent split graphs in Trinity, likely due to paralogy or allelic variants. How Trinity determines what is to be split is somewhat unknown, so here we use CAP3 to collapse isoforms that are at least 95% identical, assuming that this much identity is going to indicate allelic forms rather than paralogs. You can collapse at any level you want, however (see below).

Run batchCap3_Trinity.pl from anywhere:

    perl batchCap3_Trinity.pl %match libIDs.txt /full/path/to/trinity
    where %match = percent identity you want to collapse in CAP3
    lbIDs.txt is a text list of all your library IDs, one per line, same one you used for Trinity.

This script will generate a number of files in each Trinity/libID/trinity_out_dir that end in libID.contigs (or singlets, etc). The following script (**batchBlast_bigdata.pl**) will merge the .contigs and .singlets file, as you want the sequences in both of these.

Pull out targets of interest

Once assembled, it’s fairly straightforward to move from all 50k+ contigs to the small subset you want to focus your analyses on. The first step is to simply BLAST your assembled contigs to your reference (the sequences used to design the probes). This can be done all at once using **batchBlast_bigdata.pl**. Step by step:

1) Make a directory called “blast” or something similar. Move into that directory – you will run the batchBlast.pl script from here.
2) Make sure to first format your reference file as a blastable database:

   formatdb –p F –i references.fa

   This makes a number of files ending in .n*. When you want to give this reference to a blast submission script, just give the basename: references.fa. Make sure these files are in the directory called “blast” **or** make sure you give the relative path to the files in the submission line below.
3) Run batchBlast:

   perl batchBlast_bigdata.pl </path/to/trinity/directory> <./references.fa> <libIDs.txt>

This will create a number of .out files, per library, in the /blast directory. The libIDs.txt is the same file as you have been using all along.

Get only best hits

BLAST is not highly specific in its output – it outputs a number of alignments for the same contig. We just want the best. Run batchGetTop.pl to create .top files that contain only the best hit for a given contig.

Usage:

perl batchGetHits.pl <count>

count = number of hits you want to keep. Just use 1.

Rename target contigs

Now you have .top files where each contig has only one hit. This will now become its new identity. In other words, we’re replacing the meaningless header that Trinity gave the sequence with something we can identify as a locus. Run the **batchRenamecontigs.pl** script. This script looks at the blast file, finds the associated contig in the trinity file, but renames it with the locus header ID. This script in essence does two things: 1) renames headers and 2) pulls only the contigs that we’re interested in, those that had a blast hit. To run:

perl batchRenamecontigs.pl /path/to/trinity/directory

This will create a Libid.targets.fasta file. **Note** this script has a number of file names hard coded. That is, as long as you used the previous scripts in this pipeline, it will find the correct files (specifically, the Trinity.fasta.Libid.cap3 file in the trinity folders and the libID_targets.out.top blast and gethits output. If you deviated at all! Make sure you alter appropriate places in the script (lines 28 and 37) but with caution.

Remove duplicates

Although we collapsed a number of isoforms with CAP3, we only collapsed things at some percentage identity that you specified. Paralogs likely still remain. If you look at the libID_targest.out.top file, you’ll

still have instances where more than one of your assembled contigs matches the same target. This poses a problem, because we don't have a good way of phasing paralogs – if we have a gene with three exons, and each exon is triplicated in the genome, we don't know which combination of exons represents an entire homolog. So for now we remove them, though I'm open to other suggestions!

Usage:

perl batchRemovedups.pl

This will generate two files for each library: one ending in .norep, which is a list of the targets excluding duplicated ones, and the other is.reps, which includes all targets which had more than one hit (if you wanted to look at these later, you could).

Condense genes

We don't want to make exon trees, we want gene trees. We also need to identify each of our species in each gene so we can make sense of this data as we progress from here. The next script, **batchMakegenes.pl,** first pushes exons together in the order they were originally annotated in for the baits, and also adds a sample identifier after each header.

Usage: perl batchMakegenes.pl

Align

Now we have multifasta files for each species. This does us no good – we need to make multifasta files per each gene. The script **batchAlign.pl** does this, as well as align those multifasta files for us. It uses the aligner prank, though it's fairly easy to swap it out in favor of another alignment program within the script (you would need to change line 39, where the script writes the shell script for the alignment program, and point it to whichever program you'd like to use with the corresponding parameters).

This pipeline uses prank because it's forgiving of length variation and indel polymorphism. Remember, we took exon fragments that have adjacent introns (which will vary in length by sample) and pushed them together into "gene" fragments. Because introns vary, the aligner we use must be sensitive enough to not force the issue – prank is very good at inserting gaps into the alignment, which is perfect for us. It's also in general a very good aligner and relatively quick. We are using the *de novo* version, not the version where it uses a guide tree.

Usage:

1) make a new directory to house your alignments (six or so files per gene will be created, so creating a new folder to keep this in is generally a good idea).

2) move your .genes files into this newly created directory (mv or cp *.genes ./newdirectory)

3) run batchAlign.pl: perl batchAlign.pl

Alignment cleaning (optional)

While prank is great because it inserts gaps, we don't really want to consider alignment regions where only a few samples have sequence information. Gblocks is a program that can selectively cut out areas where sample numbers are low, for any given basepair.

Parameters defined in gblocks for this workflow:

-b1 = minimum number of sequences for a conserved position (set to half of the total number of sequences in the file +1; this is the program's minimum)
-b2 = minimum number of sequences for a flank position (likewise, set to half the total number of sequences +1)
-b3 = set the minimum length of non conserved positions before gblocks throws it out (default, 25)
-b4 = minimum length of block to keep after cleaning, set to default of 10bp
-b5 = allowed gap positions, set to "all," the least conservative of the options. Allows all gaps in the alignment to be kept.

Usage: perl batchGblocks.pl

For more:
http://molevol.cmima.csic.es/castresana/Gblocks/Gblocks_documentation.html#command_line

RAxML

This workflow runs RAxML with the outgroup option. To make this work, the user needs to provide a list of outgroups, with the putatively most distant the first in the list, and becoming progressively less distant. RAxML will not run, with the –o (outgroup) flag, if an outgroup is not present. We give it a list so that if the furthest outgroup did not get captured, it will use the next furthest.

This script will take the –gb files generated by gblocks, filter them for taxa where all sequence information is undetermined (----), then run RAxML with 500 bootstraps. You may specify the number of bootstraps on the command line. Usage:

perl batchRAxml.pl outgroups.txt 500 .fasta

-where "outgroups.txt" is the outgroup list, ordered from furthest "out" to most closely sister to your target group."500" is the number of bootstrap replicates. ".fasta" is the ending of the files you want to use as your input. If you used the previous gblocks script, it will be –gb. Do not include the wildcard.

Concatenation

If you want to compare multilocus approaches to a concatenated sequence tree, the following script will concatenate all your genes together. If a taxon is missing data for a gene, it will insert blanks. Genes MUST be aligned already. You also need to provide the program a list of taxa (and this list must match your headers).

Usage: perl concat3.pl ending species.txt

Where “ending” is the way your files are named (ie, .fasta…do not include wildcard), and species.txt is your list of taxa, one taxon per line, names matching the headers of your. If you ran PRANK as your aligner, followed by gblocks, your ending will be “-gb.”

Output is concat.fasta.

MP-EST

MP-EST is a coalescent species tree estimator (see Liu et al 2010). It takes as input bootstrap trees for all genes of interest. These directions are for running the program with only single accessions per taxon (ie, without assigning multiple individuals to a given species).

The first step is to reformat your bootstrap trees (RAxML_bootstraps.* files, if you’ve run RAxML as above). MP-EST cannot handle taxa whose names start with a number, it does not take underscores or other odd characters – these all must be changed or removed.

Next, we need to reshuffle the bootstrap trees. Currently you have one file per gene, but we need one file per bootstrap replicate. So if you ran 500 bootstrap replicates, this script is going to generate 500 files, where each line is one bootstrap tree from each gene. So if you have 700 genes, each file will have 700 lines.

Usage: perl batchParseBS.pl .out

Need to tell the script what the ending of your files are (the bootstrap trees) – that’s the .out. Don’t include the asterisk. This script can take a while, so run accordingly (by writing a submission script or running on an interactive node if available).

Next, you’ll have to edit the batchMPESTcontrol.pl file. This script prints a control file for MPEST, which needs such a file to run. The way we’re running MPEST is that each of the files you made with the batchParseBS script above will have its own run in MPEST, and then we summarize those results. In other words each file we made above is a replicated run on MPEST that will generate a tree, and we summarize those final trees to get support values.

Look here to find what the control file lines mean (under manual): https://code.google.com/p/mp-est/downloads/list

In the script, the control file is the portion that starts after “print OUT” on line 12. Leave the $file, 0, 6438 as they are (the last one is a random seed number, you can change it if you really want).

Change the next line so it reflects the number of gene trees (how many trees in each of those bootstrap files?) and the number of taxa.

Then you have to change and list all your species. The order of columns on each of those lines is species name, number of alleles (probably 1), and then name of allele in gene tree. This allows you to assign

multiple alleles/individuals to the same species if that's how you did your sampling, but if you just have one individual per species, you'll just have:

SpeciesA 1 SpeciesA

Note that it's a space separating those, not a tab.

Keep the final number after the species list set to 0.

This script submits X number of mpest runs (X = bootstrap reps you did in RAxML). If it's a lot, considering doing this in interactive or a qsub.