Running STARRInIGHTS **19 August 2011**

B. Jesse Shapiro
jesse1@mit.edu
bshapiro@fas.harvard.edu

Overview.
Strain-based **T**ree **A**nalysis and **R**ecombinant **R**egion **In**ference **I**n **G**enomes from **H**igh-**T**hroughput **S**equencing-projects (STARRInIGHTS) is an algorithm for detecting homologous recombination among closely-related microbial genomes. The input to STARRInIGHTS is an aligned contig (or set of contigs) of genomic sequences from the strains of interest, from which we wish to infer recombination breakpoints and relative rates of mutation and recombination. If the strains evolved only clonally, nearly all mutations should fall along a single phylogeny, with only a few conflicting 'homoplasic' mutations. Assuming a random mutation process, these homoplasies should be scattered evenly across the genome. If, instead, homoplasies are clustered into regions of the genome supporting a different phylogeny of clonal descent, this is better explained by a small number of recombination events than a large number of homoplasic mutation events. STARRInIGHTS first infers maximum-likelihood (ML) phylogenies for every possible subsequence of the genome, including the no-recombination scenario where the whole genome constitutes a single block with a single tree. Alternatively, several subsequences can be combined, with recombination breakpoints in between, to yield a mosaic genome, with each subsequence supporting a different ML tree. The number and locations of breakpoints will depend on the probability of recombination (which can either be set to a conservative value, or estimated from the data), and the degree to which homoplasic sites are clustered in the genome or not.

The output of STARRInIGHTS is the optimal allocation of recombination breakpoints, if any, inferred by dynamic programming across the genome. Because each block (bounded by breakpoints) has its own evolutionary history, a tree is also output for each block, along with some population genetic statistics, and whether the block supports or rejects a phylogenetic partitioning of strains (ingroup/outgroup) of particular interest (*e.g.* strains with different ecological preferences, phenotypes, geographic distributions etc.).

The procedure described here consists of three major steps, each requiring running several scripts:
1. Apply a pre-filter for stretches of incompatible SNPs (not required, but reduces the number of trees to be built in step 2) (page 1)
2. Infer maximum-likelihood trees for subsequences *i,j* of the core genome. (page 2)
3. Simulate contigs using observed trees to compute model complexity penalties. (not required but strongly recommended). (page 5)
4. Dynamic programming to infer breakpoint locations. (page 6)

Before you begin:
- make sure you have a core genome alignment, with all gapped positions trimmed, in fasta format. This can be broken up into multiple contigs. (Note: the dynamic programming currently does not support circular genome alignments. The alignment should be split into at least one linear block, ideally at a known breakpoint such as a large indel). You may use an alignment program such as Mauve or Mugsy to generate Locally Colinear Blocks (LCBs) of DNA. You may then want to trim gaps, BLAST the LCBs against a reference genome and order them accordingly into a single concatenated contig. (STARRInIGHTS will always produce breakpoints between LCBs so the LCB ordering does not matter for breakpoint inference, but may facilitate downstream analysis and visualization). These steps can be done using a *modified* version of the current pipeline of scripts, provided for reference only (you will need to change some of the paths and filenames to point them at your own data, BLAST DBs etc.):

Start with aligned LCBs each in their own file.
1. `trimGaps_bulk.pl` to run `trimGaps.pl` on each LCB
2. `blastLCBs.pl` against reference contig(s)
3. `parseBLASTALL.pl`
4. `orderLCBs.pl` > order.txt
5. `concatenate.pl` order.txt

This will produce a concatenated fasta file, and a key file (concat.$order_file.key.out) mapping the order of LCBs.
- unless your alignment has very few polymorphic sites, you will most likely require access to a computer cluster with many parallel compute nodes, in order to build many millions of ML trees.
- you will need to install PhyML v. 2.4.5 and seq-gen (both freely available at http://atgc.lirmm.fr/phyml/ and http://tree.bio.ed.ac.uk/software/seqgen/, respectively).
- the dnacomp program from the phylip package is recommended (and freely available at http://evolution.genetics.washington.edu/phylip.html) , but not required, for building parsimony trees in downstream analysis.

1. Apply a pre-filter for stretches of incompatible SNPs.
The input for this step is an aligned contig (or multiple contigs) of the core genome, named `contig.XX.fa` (where XX is some string of numbers), produced by Mauve or another alignment program. We assume that gapped positions have been removed. A pre-filtering step for phylogenetically incongruent stretches of SNPs is not required, but is recommended to reduce the amount of time required to build ML trees.

The script runs two metrics: one that calculates the minimum percentage of incongruent SNPs, by recording all observed topologies and their compatibilities in a window of informative SNPs. The second metric splits the window of informative SNPs in two and performs a test of independence for the distribution of SNPs on the left and right side of the window. The program combines this two metrics to find gross scale breakpoints, across which the optimal (but more computer intensive) dynamic programming method should not bother to evaluate.

To run this script execute:

```
sh runtopoblocks.sh full-path/contig.XX..fa [optional: window size] >
contigs.XX.w150.dat
```

This produces an output file named `contig.XX.w150.dat` (or named whatever you like), where 'w150' indicates the default sliding window size of 150 informative SNPs. The format of this file looks like:

```
pos posrea mean_t sd_t mean_chi signf 12F01 ZF270 1F289 1F273 ZF14 ZF264
ZF170 13B01 1F53 12B01 ZS63 1F111
...
1543 15811 0.10107685764056659 0.024185825910146612 3.9587104156588355E-4 **
0 0 0 0 0 1 0 0 1 0 0 0
1544 15823 0.10053755441836973 0.024553166953244856 4.0062290316924393E-4 **
1 0 0 0 0 1 0 1 1 1 1 0
1545 15839 0.09998416102274547 0.02490962572105084 4.066689161962067E-4 - 0 1
1 1 1 0 1 0 0 0 0 1
1546 15852 0.09942451013153038 0.02526305598799705 4.145920612458228E-4 - 0 1
1 1 1 0 1 0 0 0 0 1
...
```

Where the columns are:
pos: the position in the informative SNP alignment,
posrea: position in the real alignment,
meat_t: the mean percentage of incongruent topologies over overlapping windows on this position.
sd_t: the standard deviation of the last measure.
mean_chi: the mean chi-square value of the independence test over overlapping windows on this position.
signif:
if mean_t > 0.1 (10% incongruent SNPs), this column reports the probability that both sides of the window come from the same distribution:

0.000001 *****
0.00001 ****
0.0001 ***
0.001 **

0.01 *
>0.01 –

The remaining columns represent the SNPs with 0s and 1s. This is used by the R script `plotlandscape.r` to create a visual representation of the output file. To use this script, edit the name of the file in `plotlandscape.r` and call it from R (e.g. `source("plotlandscape.r")`). The script produces a plot like the one shown in Figure S10.

2. Infer maximum-likelihood trees for subsequences *i,j* of the core genome

Next, you must produce a list of core genome subsequences for which to build trees (excluding any subsequences that span significantly incongruous regions identified in step 1, above). This is done by running:

```
perl make_subSeqList.pl –msa –prebreaks –strains –pcut –lenlim –nodelineup –
startsite –endsite –skippers –LCB_order
```

The ten command line arguments are as follows:

```
-msa: the aligned fasta file input (e.g. contig.XX.fa)
-prebreaks: phylogenetic incongruence output file (e.g. contig.XX.w150.dat).
If none, just type anything to fill this field, but do not leave it blank.
-strains: subset of strains in the fasta file to include. List each strain
name on a separate line in a text file.
-pcut: the p-value cutoff (in log scale) for considering a stretch of
phylogenetic incongruence in 'prebreaks' file. A cutoff of ~1/(# informative
sites) is recommended. For example if there are 1e4 informative sites, set
pcut = 4. Higher values of pcut are more conservative, but will result in
building more trees and longer runtimes. Do not leave this field blank even
if you skipped the pre-filtering step – just type any number.
-lenlim: the total amount of sequence (in bp) to be run on a single compute
node.
-nodelineup: the number of jobs (trees) lined up on each compute node.
-startsite: the polymorphic site to start at (0 is the first site)
-endsite: the site to end at.
-skippars: takes a value of 1 or 0. If = 1, skip stretches of SNPs that fit
the same tree (meaning that no breaks can be introduced within them); if 0,
do not skip them. Setting 1 reduces the number of trees and the runtime. If
set to 1, an additional file contig.XX.lk.txt will be produced listing the
'perfectly parsimonious' subsequences. NOTE: THIS OPTION IS NOT YET
SUPPORTED. PLEASE JUST SET IT TO 0.
-LCB_order: a tab delimited file listing the order of Locally Colinear Blocks
(LCBs) that have been concatenated to make up the aligned msa. Subsequences
that span LCB boundaries will not be considered. The 7 columns in this file
are: (1) the LCB ID, (2) the contig ID, (3) LCB's start position in the
concatenated contig, (4) it's end position, (5) LCB's start position in
another contig/reference genome of origin, (6) it's end position, (7) + or –
strand. The *.key.txt output of concatenate.pl, described above, is in the
appropriate format, or you can create your own order, or omit this file if
you have just one contig, or want to run the rest of the pipeline separately
on each contig.
```

The resulting list of subsequences *i,j* will be broken up into *n* separate files named contig.XX.concat3.YY.subseqs.txt, where YY = `endsite` + *n*. Each file contains `nodelineup`*100 subsequences, each printed on a separate line. Once this number of subsequences is achieved, a new file *n* is started.

Many thousands or millions of subsequences will likely be required, necessitating building an equally large number of ML trees. It is assumed that this will be done on a computer cluster with many parallel nodes. You can set `lenlim`, `nodelineup`, `startsite` and `endsite` to split your tree building jobs into an appropriate number of batches. We provide an example script for submitting SGE jobs, but it is likely that your computer cluster may require a different submission strategy depending on the number and speediness of your nodes.

An example submission script to be run on each of the subsequence files produced by `make_subSeqList.pl`:

```
perl writePhy+submit.pl -subseqfile -msa -strains -alpha -Ncats –queue -
maxTrees
```

The seven command line arguments are as follows:

```
-subseqfile: one of the contig.XX.concat3.YY.subseqs.txt described above
-msa: the aligned fasta file descrbed above (e.g. contig.XX.fa)
-strains: subset of strains in the fasta file to include. List each strain
name on a separate line in a text file.
-alpha: the gamma distribution shape parameter to model rate heterogeneity
among sites. This should be estimated ahead of time by running PhyML on the
entire aligned core genome, or if this is not possible then on windows of a
few thousand bp.
-Ncats: the number of discrete gamma-distributed rate categores. 2 or 4 is
recommended. 2 will run faster.
-queue: specify if you want to submit to a specific queue for speedy, short
or long jobs
-maxTrees: the maximum number of jobs (trees) lined up on each compute node.
You may set to the same value as –nodelineup used in make_subSeqList.pl, or
choose a smaller number (e.g. 1000-3000 is recommended for a job lasting ~1-3
hours, but this will depend on the sequence length and number of strains in
your alignment).
```

Remember this script is just an example. You will need to install PhyML on your system and specify the path in the script. You will also need to run PhyML first on either the whole aligned core genomes (or if that is too many bp for PhyML to handle, break it up into 1000-5000 bp windows) to estimate `alpha`. You can do this using a command like:

```
./phyml your_aligned_genome_file.phy 0 I [Ndatasets] 0 JC69 e 0 [Ncats] e
BIONJ y y
```

where `[Ndatasets]` is 1 for the whole genome or the number of windows you broke it into, and `[Ncats]` is 2 or 4.

Next, cleanup the output from all the PhyML runs and save the important information (likelihood scores and trees):

```
perl retrieveLike.pl –subseqfile -maxTrees
```

(NB: Use the same value of `maxTrees` as you did for `writePhy+submit.pl`. Depending on how much storage space you have available, it may be important to do this after every `writePhy+submit.pl` run because the .phy sequence files produced can be very large, on the order of a few Gb each, potentially amount to a few hundred Gb in total)

Running `retrieveLike.pl` will produce a directory called /lk, if it does not already exist, in your current working directory, and created a file called `contig.XX.concat3.YY.lk.txt` containing the ML trees and likelihoods for each subsequence. If `skippars` was set to 1, you must also copy any *.lk.txt files ('perfect parsimony' subseqs') created by `make_subSeqList.pl` to /lk.

Once you have done this for all the required subsequences, you must run on more script to check that all your trees completed, and then you are ready to proceed to step 2. You may also proceed directly to step 3 without correcting for model complexity, but this is not recommended.

Before proceeding to step 2 or 3, you must run the script:

```
perl printVar.pl -msa –strains
```

This will print a summary of polymorphism in your msa to the screen, for example:

```
***informative sites****
dimorph  inform: 22486
trimorph inform: 202
4-morph  inform: 1
total    infrom: 22689
***non-informative sites***
singletons in 1 leafs: 7348
singletons in 2 leafs: 7
singletons total: 7355
```

And will produce warnings if likelihood files for some subsequences are missing:

```
/lk/contig.73.concat3.10018.lk.txt does not exist!
```

And also produces files

```
[msa_name].poly.key.txt
[msa_name].poly.variants.key.txt
[msa_name].singletons.key.txt
key.txt
```

If your core genome alignment consists of multiple contigs, make sure either run each contig separately in a different directory or rename the above files, for example to key.contig1.txt between each run of `printVar.pl`. Once all contigs are done, catenate all the key.txt files into a file called `subseqs.list.txt` or `key.txt`, for example by doing:

```
cat key.contig1.txt key.contig2.txt key.contig3.txt > subseqs.list.txt
```

3. Simulate contigs using observed trees to compute model complexity penalties.

First, you need to divide your subsequences into percentiles of length (bp) and polymorphism levels (SNPs per site). You can do this in percentiles, quartiles, or deciles. The example here uses deciles. Run:

```
perl getTiles.pl [run_dir] [num_breaks]
```

where `[run_dir]` is the name of the directory containing the file `subseqs.list.txt` or `key.txt`, and `[num_breaks]` defines how finely you want to divide up your subsequences (for example, num_breaks = 100 would divide it into 100 bins, = 10 would be 'deciles', = 4 would be quartiles, etc.). This will produce the files

```
poly.tiles.txt
length.tiles.txt
```

These list the cutoffs (upper limit) for each bin, in terms of SNPs/site and bp, respectively. You may also add additional cutoff values on new lines in these files if would like to customize the binning.

Second, you will need to sample different tree topologies at random from those observed. For each bin of polymorphism/length, you will sample 100 different trees. You should first make a new directory called `/sim` in `[run_dir]`, and then run the following command in `/sim`:

```
perl sampleTrees_bulk.pl [run_dir] poly.tiles.txt length.tiles.txt
```

This may take a while (~1-2 hours) to run, depending on the number of bins. Many files will be produced, one for each observed bin of polymorphism/length, called `sampleTrees.len_[bp_bin].poly_[poly_bin].txt` where `[bp_bin]` is the upper limit in bp of the length bin and `[poly_bin]` is the upper limit in SNPs/site of the polymorphism bin. Each file contains a list of the observed tree topologies (up to 100) within the bin.

Third, you need to simulate sequence along all of your sampled trees. This is done with seq-gen (freely available at http://tree.bio.ed.ac.uk/software/seqgen/), which you need to install. Then run seq-gen using this command in the directory containing your `sampleTrees.len_[bp_bin].poly_[poly_bin].txt` files:

```
perl runSeqGen_modcorrect.pl
```

You must first modify the script to specify the correct path to seq-gen. You may also choose to make subdirectories (e.g. for different length bins, /len10, /len100, etc.) and run break up your seq-gen runs within them. This will yield files named `contig.[bp_bin]_[poly_bin]_XX.concat3.fa`, where XX is a number between 1 and 100, corresponding to a replicate simulation, each done on a different observed tree.

Fourth, once your seq-gen runs are complete, you must build ML trees for each subsequence within the simulated sequences. This is done by running:

```
perl make_subSeqList_modcorrect.pl [simulated_contig] [strains]
```

on each of your `simulated_contig` files (`contig.[bp_bin]_[poly_bin]_XX.concat3.fa`), of which there are 100 replicates per bin. This is best done in bulk using a modified version of a script like this:

```
perl submitAllSims_modcorrect.pl [queue] [poly_bin] [alpha] [strains]
```

where `[queue]`, `[alpha]`, `[poly_bins]` and `[strains]` are all arguments described above. This script calls `make_subSeqList_modcorrect.pl` and `writePhy+submit.pl`, so make sure to specify their correct paths in the script `submitAllSims_modcorrect.pl`.

Once all the simulated trees have completed, it remains to calculate the L1/L0 ratio (the likelihood ratio of a model with 1 breakpoint in the subsequence to a model with 0 breakpoints) for each bin. Make a new directory called `/L1L0` and copy all the relevant PhyML output there:

```
cp *subseqs.txt L1L0/
cp *stat.txt L1L0/
cp *tree.txt L1L0/
```

In the `L1L0/` directory, run:

```
perl retrieveLike_parseL1L0max.pl
```

which will produce the file `L1L0.max.txt`, containing the maximum value of the L1/L0 ratio across 100 simulated contigs for each bin. This file is basically a 'heatmap' (a 2D matrix, with each combination of length and polymorphism bin having an L1/L0 ratio, which be used later as a model complexity penalty), which you can choose to 'smooth' manually or using a smoothing algorithm of you choice. This ensures that imperfect sampling (only 100 trees per 2D bin) does not result in 'bumps' in the heatmap that would lead to undesirable irregularities in the model correction procedure.

4. Dynamic programming to infer breakpoint locations.

The final stage of the STARRInIGHTS pipeline is to find the optimal allocation of breakpoints between subsequences of the core genome using dynamic programming. You can either set the breakpoint probability manually (generally to a low value, *e.g.* log *p*(break) = -100, to yield a conservative estimate of breakpoints), by running `dp.pl` or `dp+brkCorrect.pl`, or infer the breakpoint probability by Expectation-Maximization (E-M) by running `dp+em.pl` or `dp+em+brkCorrect.pl`. It is recommended that you generate empirical model complexity correction factors to correct for spurious breakpoints, as described in (2) above. To apply this correction to the dp algorithm, you should run `dp+em+brkCorrect.pl` or `dp+brkCorrect.pl` (respectively with or without E-M to estimate the breakpoint probability).

Before running dp, you must first make a directory named whatever you want, but for the purposes of this manual let's call it `[dp_dir]`. You must copy your list of subseqs, named either `subseqs.list.txt` or `key.txt` (see the end of (1) above), to `[dp_dir]`. If your aligned genome consists of a series of concatenated LCBs, you should place in this directory a file named `LCB.boundaries.key.txt` (the same tab-delimited file generated by `concatenate.pl`, used as the LCB_order argument to `make_subSeqList.pl`. You must also make a text file `[pB_init]`, listing a range of different initial breakpoint probabilities. Depending on your initial *p(*break), E-M may converge to different final values of *p*(break) so it is prudent to try a range. The file should look something like this, with each starting value of log *p*(break) listed on a new line:

```
-100
-50
-15
-10
-8
-6
-4
-2
```

If you are running dp without E-M, dp will simply be run with each listed log *p*(break) as a fixed probability.

The commands to run the various flavours of dp are as follows:

(i) no E-M, no complexity correction:

```
perl dp.pl [dp_dir] [pB_init]
```

(ii) with E-M, no complexity correction:

```
perl dp+em.pl [dp_dir] [pB_init]
```

(iii) no E-M, with complexity correction:

```
perl dp+brkCorrect.pl [dp_dir] [pB_init] [correction_file]
```

(iv) with E-M, with complexity correction:

```
perl dp+em+brkCorrect.pl [dp_dir] [pB_init] [correction_file]
```

If you are running dp with the correction factors, you will need to list a third command line argument, `[correction_file]`, referring to the output file produced in (2) above (`L1L0.max.txt`, or your smoothed version of this). Remember to specify complete paths for each of the command line arguments in (i) – (iv) above.

Running (i) will produce output files:

a) `starri.init.pB_[pB_init].out` (one file for every different starting value of *p*(break), containing likelihood scores and indicators of run progress)

b) `STARRI.2.0.noEM.[contigID].init.pB_[pB_init].out` (one file for every starting value of *p*(break) and if multiple contigs make up the core genome, for every different contig (named [contigID]), listing the optimal breakpoint locations on a single line)

c) `STARRI.2.0.noEM.[contigID].init.pB_[pB_init].blocks.out` (same as (b) but with each block listed on a separate line, with 5 tab-delimited columns: (1) the block's start position (in units of polymorphic sites), (2) end position (units of polymorphic sites), (3) start position (actual position in trimmed contig alignment), (4) end position (actual position in trimmed contig alignment), (5) length of the block in trimmed bp (equal to (4)-(3)+1).

Running (ii) will produce the equivalent files:
a) `starriEM.init.pB_[pB_init].out`
b) `STARRI.2.0.EM.[contigID].init.pB_[pB_init].out`
c) `STARRI.2.0.EM.[contigID].init.pB_[pB_init].blocks.out`

Running (iii) will produce the equivalent files:
a) `starri.corr.init.pB_[pB_init].out`
b) `STARRI.correct.noEM.[contigID].init.pB_[pB_init].out`
c) `STARRI.correct.noEM.[contigID].init.pB_[pB_init].blocks.out`

Running (iv) will produce the equivalent files:
a) `starriEM.corr.init.pB_[pB_init].out`
b) `STARRI.correct.EM.[contigID].init.pB_[pB_init].out`
c) `STARRI.correct.EM.[contigID].init.pB_[pB_init].blocks.out`

To print a summary of the likelihoods achieved using different starting conditions, run the following as appropriate:

(i) `perl findML_noEM.pl starri.init.pB_`
(ii) `perl findML.pl starriEM.init.pB_`
(iii) `perl findML_noEM.pl starri.corr.init.pB_`
(iv) `perl findML.pl starriEM.corr.init.pB_`

This will print tab-delimited text to the screen, with 4 columns: "pBinit", "pB", "like", "nb", corresponding respectively to the starting value of log *p*(break), the final value of log *p*(break) after E-M, the final log likelihood achieved by the model, and the number of breakpoints in that model. You should inspect this file to find the maximum likelihood model (the one with the largest value of "like"). Take note also of the value of "pBinit" corresponding to the ML model (or if all values of "pBinit" yield the same final likelihood, just choose one arbitrarily) and call this `[pB_init_ML]`.

You can now run scripts to parse out the relevant info about your ML blocks and build parsimony trees for each block (remember, the ML tree for a given subsequence can be found in /lk/ contig.XX.concat3.YY.lk.txt).

First, modify `prepPARS.pl` to contain the right path to dnacomp and make sure the file `subseqs.list.txt` or `key.txt` is in the `dp_dir` directory. Then run the following:

```
perl prepPARS.pl [dp_dir]/subseqs.list.txt [dp_dir]/ [pB_init_ML] [mode]
```

where `[mode]` corresponds to one of runmodes (i) – (iv) above, specified by typing one of the following as appropriate:

```
STARRI.2.0.noEM
```

```
STARRI.2.0.EM
STARRI.correct.noEM
STARRI.correct.EM
```

This will make a new directory within `[dp_dir]` called `dnapars.[pB_init_ML]`. In this directory, an SGE job submission file, `runDNApenny.sh`, will be produced. Go to this directory and submit the script by doing something like (depending on your job queuing system):

```
qsub runDNApenny.sh
```

Once the job has completed, you will have produced a parsimony tree for each of your ML blocks. You will now parse all this information by running:

```
perl parseBlockParsTrees.pl [dp_dir]/subseqs.list.txt [dp_dir]/ [pB_init_ML]
[mode]
```

The command line arguments are the same as `prepPARS.pl` above. This script will produce the following files:

"tree_file" = `trees.pB_[pB_init_ML].txt`
"parsimonious_snp_file" = `pars.snp.pB_[pB_init_ML].txt`
"homoplasic_snp_file" = `homoplasic.snp.pB_[pB_init_ML].txt`

The "tree_file" contains a tab-delimited list of ML blocks, with columns as follows:
1. the contig ID
2. block number
3. block start position (trimmed bp position in core genome)
4. block end position
5. number of polymorphic sites supporting the parsimony tree for this block
6. number of polymorphic sites rejecting the parsimony tree for this block (homoplasies)
7. phylogenetic partitions in the parsimony tree. Each partition is a list of strains separated by dots that group together on the tree, to the exclusion of other strains (*e.g.* strain1.strain2.strain3.). If multiple partitions exist, they are separated by an underscore.
8. the number of polymorphic sites supporting each partition (in the same order as column (7), separated by underscores)
9. phylogenetic partitions that reject the parsimony tree. Formatted as (7) above.
10. the number of polymorphic sites supporting each homoplasic partition in column (9).

The "parsimonious_snp_file" file simply contains a ranked list of the most commonly supported partitions across all blocks in the core genome. It contains two columns, the first containing the number of polymorphic sites supporting a particular tree partition, and the second containing the identity of the partition, in the format described in column (7) of the "tree_file."

The "homoplasic_snp_file" is equivalent to "parsimonious_snp_file" except it lists partitions commonly seen as homoplasies across the genome.

Alternatively, you can run `parseBlockParsTrees+splitstree.pl` to produce the same output, and also `splits*.fa` files which can be used as input to the SplitsTree program to produce a Neighbour-net graph of the data. The `+splitstree.pl` option takes one more command line argument, 1 or 0, to specify whether homoplasic SNPs (those not part of a block's parsimony tree) should be included (1) or not (0).

Finally, you can optionally run one more script to estimate divergence (*d*; see Nei and Li, 1979; Arbogast et al. 2002), Tajima's *D* (Tajima 1989), and population differentiation (Fst) within each block for a specified partition of your strains into 'ingroup' and 'outgroup'. The in/outgroup designation may be based on an ecological factor of interest, or a common phylogenetic partition, that divides the strains into 2 groups. You must first create 2 text files,

listing the names of your in/outgroup strains, each on a separate line. These files will be referred to as `[ingr_file]` and `[outgr_file]`. Then run the script:

```
perl parseBlockIngroupStats.pl [dp_dir]/subseqs.list.txt [dp_dir]/
[pB_init_ML] [mode] [common_pars_snps] [ingr_file] [outgr_file]
```

where all the arguments are the same as specified for `parseBlockParsTrees.pl` above, and `[common_pars_snps]` is the "parsimonious_snp_file" generated as described in the previous step.

This will generate a file called `trees.pB_[pB_init_ML]+stats.ingr.[ingr_file].txt`, containing the following columns, one for each block:

1. "contig" = the contig ID
2. "blk" = block number
3. "start" = block start position (trimmed bp position in core genome)
4. "end" = block end position
5. "Nsnp_pars" = number of polymorphic sites supporting the parsimony tree for this block
6. "Nsnp_incons" = number of polymorphic sites rejecting the parsimony tree for this block (homoplasies)
7. "splitSup" = number of polymorphic sites supporting the ingroup/outgroup partition in this block
8. "splitRej" = number of polymorphic sites rejecting the ingroup/outgroup partition in this block
9. "domGrp" = the identity of the most commonly supported partition in this block
10. "domFrac" = the fraction of polymorphic sites that support the dominant group identified in column (9)
11. "parsTree" = phylogenetic partitions in the parsimony tree. Each partition is a list of strains separated by dots that group together on the tree, to the exclusion of other strains (*e.g.* strain1.strain2.strain3.). If multiple partitions exist, they are separated by an underscore.
12. "Nintree" = the number of polymorphic sites supporting each partition (in the same order as "parsTree", separated by underscores)
13. "inconsTree" = phylogenetic partitions that reject the parsimony tree. Formatted the same as "parsTree".
14. "Nincons" = the number of polymorphic sites supporting each homoplasic partition in "inconsTree".
15. "NsegSite" = the number of segregating polymorphic sites (informative or not) in the block
16. "kHat" = the average nucleotide identity per site in the block (see Tajima 1989)
17. "tajD" = Tajima's *D* statistic for the block (see Tajima 1989_
18. "div" = estimate of divergence, $d = 2\mu t$, where μ is the mutation rate and *t* is the number of generations since divergence of ingroup and outgroup (Nei and Li, 1979; Arbogast et al. 2002).
19. "Fst" = population differentiation between ingroup and outgroup.

Columns listed in bold represent data not reported in the "tree_file" output by `parseBlockParsTrees.pl`.

Files for creating 'Starry nights' plots in R are also printed to a directory called `[dp_dir]/output.pB_[pB_init_ML]`. These files allow you to generate a heatmap of support for different phylogenetic bipartitions (tree partitions splitting the strains into 2 distinct groups) along the genome: the core genome is plotted along the x-axis (with tick marks indicating inferred breakpoints), and blocks supporting or rejecting each observed bipartition (on the y-axis, ranked according to their frequency across the whole core genome, with more frequent bipartitions occupying proportionally more space). The files are as follows, one per contig of the core genome:

`[contig].support.txt`: fraction of informative SNPs supporting a partition on the y-axis (strong support = white; no support = black)
`[contig].Nsup.txt`: the number of SNPs supporting a partition on the y-axis (strong support = white; no support = black)
`[contig].reject.txt`: fraction of SNPs rejecting a partition on the y-axis (strongly reject = white; none rejecting = black)
`[contig].Ykey.txt`: a tab-delimited file describing the partitions plotted along the y-axis, from most common to least common. Column 1 = total number of SNPs across the core genome supporting the partition; 2 = start position on the y-axis; 3 = end position on the y-axis; 4 = group 1 of the partition; 5 = group 2 of the partition.

To produce plots from the first 3 files, simply call them as commands in R (*e.g.* source("58.support.txt")). This will produces images like the examples provided as pdf files in the `example_plot` directory. These images show data from one contig of the *V. cyclotrophicus* core genome (designated contig 58). This contig includes the 'chitin operon' which is recognizable as a dense cluster of support for the second most-common bipartition (the ecological split, grouping 1F111, 1F273 and 1F53 separately from the other strains) around the 400 Kb position of the contig.