The Large Scale Blast Score Ratio (LS-BSR) pipeline

Citation:

Jason W. Sahl, J. Gregory Caporaso, David A. Rasko, Paul S. Keim (2014). The large-scale blast score ratio (LS-BSR) pipeline: a method to rapidly compare genetic content between bacterial genomes. PeerJ PrePrints 2:e220v1.

Jason W. Sahl, J. Gregory Caporaso, David A. Rasko, Paul S. Keim (2014). The large-scale blast score ratio (LS-BSR) pipeline: a method to rapidly compare genetic content between bacterial genomes. PeerJ 2 (e332).

Contact:

Please address queries, concerns, improvements to jasonsahl at gmail dot com

What does it do?

The LS-BSR pipeline was designed as a way to quickly compare the genetic content between a large number of bacterial genomes. LS-BSR can calculate several pan-genome statistics in a population and the output can be easily visualized with a variety of third-party tools. Additionally, LS-BSR can be used to query a set of genes against a large set of genomes to identify gene distribution and conservation. LS-BSR was developed to be easy to run and interpret.

Installation

-The code is kept here:

https://github.com/jasonsahl/LS-BSR.git

-You can clone the repository to your own system with git:

$git clone https://github.com/jasonsahl/LS-BSR.git

-Enter the directory, then:

$python setup.py install

-if your install directory is /Users/jsahl/LS-BSR, run:

$export PYTHONPATH=/Users/jsahl/LS-BSR:$PYTHONPATH

-You can add this to your .bashrc or .profile

-You can test your installation by running the tests:

$python /Users/jsahl/LS-BSR/tests/test_all_functions.py

-If your installation is correct, all tests should pass

Dependencies

1. Python >2.7 and <3.0
2. USEARCH (tested version is 6.0.307), path is passed as command-line option - **only required if a set of gene sequences is not supplied**. 32-bit version should be sufficient for most applications, including the analysis of 1000 *E. coli* genomes. Tested successfully with versions v7.0.959 and 7.0.1090. Can be freely obtained for academics/non-profits from: http://www.drive5.com/usearch/
3. BioPython, must be in $PYTHONPATH environmental variable. Can be freely obtained from: http://biopython.org/wiki/Main_Page
4. blastall (tested version is 2.2.25), must be in path as 'blastall' **–only required if you are using BLASTN or TBLASTN, and not BLAT**. Errors have been observed with v2.2.26. v2.2.25 can be obtained from: ftp://ftp.ncbi.nlm.nih.gov/blast/executables/release/2.2.25/
5. BLAT (tested version is v. 35x1), must be in path as 'blat' – **only required if you use choose blat for your alignment method**. Can be obtained from: http://hgdownload.cse.ucsc.edu/admin/exe/
6. Prodigal (tested version is 2.60), must be in path as 'prodigal' - **only required if a set of gene sequences is not supplied**. Can be obtained from: https://code.google.com/p/prodigal/
7. Numpy, must be in PythonPath. **Numpy is only required for the post-process compare matrices tool**. If you don't want to install numpy, you should be fine, but won't be able to run the compare script. Can be obtained from: www.numpy.org

Command Line options

-d DIRECTORY, --directory=DIRECTORY : the directory to your fasta files, all must end in ".fasta". Can either be complete genomes or draft assemblies. Scaffolds are discouraged. [REQUIRED]

-i ID, de-replication clustering value for USEARCH, defaults to 0.9 (range from 0.0-1.0)

-f FILTER, whether to use BLAST filtering, default is "F" or filter, turn off with "T". Turning this to "T" should speed up the analysis, but may throw out highly repetitive sequences.

-p PROCESSORS, number of processors to use, defaults to 2.

-g GENES, if you have a list of genes to screen, supply a nucleotide fasta file. Each gene sequence must be in frame, or questionable results will be obtained (only true for TBLASTN). If this flag is not invoked, then the de novo gene prediction method is invoked

-**b BLAST**, which alignment method to use. Default is 'tblastn', can be changed to 'blastn' or 'blat'. Can be used with either a list of supplied genes, or with the *de novo* method.

-q PENALTY, blast mismatch penalty, default is -4, only works with blastn. Optimized to return longer matches. Only certain q/r ratios are allowed. See BLAST documentation for more details.

-r REWARD, blast reward value, default is 5, only works with blastn. Optimized to return longer matches. Only certain q/r ratios are allowed. See BLAST documentation for more details.

-l LENGTH, minimum BSR value to be called a duplicate, defaults to 0.7. The BSR of the "duplicate" divided by the reference bit score must be greater than this value to be called a duplicate

-m MAX_PLOG, maximum value to be called a paralog, defaults to 0.85. If the BSR value is greater than this value, then it is considered to be an ortholog

-n MIN_HLOG, minimum BLAST ID to be called a homolog, defaults to 75. If the BLAST ID is below this value, it is considered a remote homolog

-t F_PLOG, filter ORFs with a paralog from BSR matrix? Default is F (do not filter), values can be T (filter paralogs) or F

-k KEEP, keep or remove temp files, choose from T or F, defaults to False (F)

Test data – give LS-BSR a whirl on small datasets

-Test data is present in the test_data directory. This data consists of:

- Genomes (4 *E.coli* genomes from 4 different pathogenic variants). Genomes are:

- H10407 - enterotoxigenic *E. coli* (ETEC)
- E2348/69 - enteropathogenic *E. coli* (EPEC)
- O157:H7 sakai - shiga toxin *E. coli* (STEC)
- SSON046 - Shigella sonnei

-Genes (5 different markers that delineate between the variants). These include:
- IpaH3 - Shigella invasion antigen. Mostly present in *Shigella* spp.
- LT - heat-labile toxin. Only present in ETEC (not all)
- ST2 - heat-stable toxin. Only present in ETEC (not all)
- bfpB - bundle forming pilus. Only present on plasmid in EPEC
- stx2a - shiga toxin. Present in STEC

-You can test out the LS-BSR functionality in 4 different ways:

1. Test the gene screen method with tblastn:

-enter test_data directory, run LS-BSR

$python /Users/jsahl/LS-BSR/ls_bsr.py -d genomes -g genes/ecoli_markers.fasta

-the output should show how each gene is only present in the correct pathovar

2. Test the gene screen method with blastn:

-enter test_data directory, run LS-BSR

$python /Users/jsahl/LS-BSR/ls_bsr.py -d genomes -g genes/ecoli_markers.fasta -b blastn

3. Test the de novo gene prediction method:

-enter test_data directory, run LS-BSR

$python /Users/jsahl/LS-BSR/ls_bsr.py -d genomes -u /usr/local/bin/usearch6

-To inspect the output, you can look up the following entries in the BSR matrix. They should correspond with the results obtained with the gene screen methods (TBLASTN only):

-IpaH3 -> centroid_1724 LT -> centroid_11953 ST2 -> centroid_19265 bfpB -> centroid_1922 stx2a -> centroid_7471

-Sample output for each method is shown in the test_data directory

Visualization of output:

1. The output of LS-BSR can be visualized in many different ways. One popular method to visualize the output as a heatmap is through integration with R. Many beginners find R to be intimidating, but this link by Kat Holt provides some excellent workflows on how to build heatmaps, and correlate the output with phylogenies:

http://bacpathgenomics.wordpress.com/2012/05/25/displaying-data-associated-with-phylogenetic-trees/

2. The Interactive Tree of Life (iTOL) project has an interface that is very user-friendly and can directly take LS-BSR output and a phylogeny and create publication ready figures. iTOL can be found here:

http://itol.embl.de/

3. Finally, MeV is designed as a way to visualize expression data, but can just as easily create heatmaps from LS-BSR output. MeV can also be used to cluster LS-BSR data. MeV is platform independent and can be found here:

http://www.tm4.org/

Post-matrix scripts

1. compare_bsr.py

-what does it do? Looks for CDS differences between two user-defined populations. Differences can be set by user-defined thresholds for presence and absence. The "names.txt" file contains the names as they should be listed in your separate groups file

-what do you need for the script to run? Requirements include:

- BSR matrix
- Two new-line delimited group files, taken from "names.txt"
- FASTA file of all CDS sequences

-what does output look like? Perhaps the most interesting output includes two separate files with any FASTA entries unique to each defined group

```
$python compare_BSR.py -1 group1.txt -2 group2.txt –f consensus.fasta –b
 bsr_matrix_values.txt
```

2. filter_BSR_variome.py

-what does it do? Filters out the conserved regions of the pan-genome, if you are only interested in looking at the "variome" or accessory genome

-what do you need for the script to run?

- BSR matrix
- Sometimes if a single genome is missing a value, it is still of interest. You can change the number of missing values that can still be considered as core, and therefore filtered

-what does output look like? A new BSR matrix, with only variable positions included

```
$python filter_BSR_variome.py –b bsr_matrix_values.txt
```

3. filter_column_BSR.py

-what does it do? Can remove a column from a BSR matrix, in the case where a genome doesn't belong, or is of poor quality

-what do you need for the script to run?

- BSR matrix
- Prefix for output file
- New line delimited file of genome(s) to remove

-what does output look like? A new BSR matrix, with genome columns removed

```
$python filter_column_BSR.py –b bsr_matrix_values.txt –p pruned –g to_remove.txt
```

4. isolate_uniques_BSR.py

-what does it do? Isolates CDSs only present in a single genome, using a user defined threshold for the definition of absence

-what do you need for the script to run?

- BSR matrix
- Threshold for absence, defaults to 0.4

-what does the output look like? A new BSR matrix, with only CDSs present in a single genome

$`python isolate_uniques_BSR.py –b bsr_matrix_values.txt`

5. pan_genome_stats.py

-what does it do? Calculates several popular pan-genome stats, based on the BSR matrix

-what do you need for the script to run?

- BSR matrix
- Upper and lower threshold for BSR values

-what does output look like? Several stats are printed to screen. The script also creates two files for the IDs of core and unique CDSs. The frequency_data.txt file can be graphed in order to see the distribution of CDSs across the pan-genome

$`python pan_genome_stats.py –b bsr_matrix_values.txt`

6. BSR_to_PANGP.py

-what does it do? Converts a BSR matrix to something that can be easily visualized with PanGP (http://PanGP.big.ac.cn). CDSs that are above a given threshold are converted to a "1" and below that threshold are converted to a "-".

-What do you need for the script to run?

- BSR matrix
- Threshold for presence/absence

-what does output look like? A new matrix compatible with PanGP

$`python BSR_to_PANGP.py –b bsr_matrix_values.txt`

7. BSR_to_gene_accumulation_scatter.py

-what does it do? For a given number of iterations, the script randomly samples genomes at various depths, and reports back the number of core and unique CDSs in the pan-genome. The script also determines the gene accumulation in the pan-genome. The output can be easily graphed in Excel.

-what do you need for script to run?

- BSR matrix
- Upper and lower bounds for presence/absence

-what does output look like? The mean for each sampling depth is printed to screen. The script can be run to output accumulation, core, uniques, or all (default).

```
$python BSR_to_gene_accumulation_scatter.py –b bsr_matrix_values –i 100
```

8. quantify_BSR_uniques.py

-what does it do? Prints out the number of unique CDSs, sorted by a given tree. Nice way of annotating a tree with where unique CDSs are present

-what do you need for script to run?

- BSR matrix
- Newick tree

-what does output look like? The script prints to a file: uniques_sorted_by_tree.txt, which shows the taxa name and the number of unique CDSs

```
$python quantify_BSR_uniques.py –b bsr_matrix_values.txt –r test.tree
```

9. reorder_BSR_matrix_by_tree.py

-what does it do? Transposes a BSR matrix and re-orders the matrix based on the order of the taxa in a newick-formatted tree

-what do you need for script to run?

- BSR matrix
- Newick tree

-what does output look like? The script prints a reordered BSR matrix to file. Note that since the matrix is transposed, the number of columns can be significant. Best for smaller analyses.

```
$python reorder_BSR_matrix_by_tree.py –b bsr_matrix_values.txt –t test.tree
```

10. invert_select_group.py

-what does it do? Can be used in conjunction with the compare_BSR.py script. If you are comparing groups, you copy the IDs from group1 into a file, then uses the names.txt file to automatically create the group2 IDs.

-what do you need for script to run?

- New line delimited list of IDs in group1
- “names.txt” file which is standard output of LS-BSR

-what does output look like? The IDs from group2 are printed to screen, but can also be re-directed into an output file

```
$ python invert_select_group.py group1.txt names.txt > group2.txt
```

11. select_seqs_by_IDs.py

-what does it do? Sub-selects a FASTA file based on a list of IDs. This can be used in conjunction with the pangenome_stats script to select the core or unique genes from the consensus.fasta file

-what do you need for script to run?

- New line delimited list of FASTA headers
- FASTA file

-what does output look like? A new FASTA file is generated that contains only the sub-selected sequences of interest

```
$ python select_seqs_by_IDs.py -i in.fasta -d fasta_IDs.txt -o out.fasta
```

12. slice_ref_genome.py

-what does it do? This script splits a provided reference into fragments using a sliding window approach. This could be useful for the case where you are interested in intergenic regions, deletion junctions, etc.

-what do you need for script to run?

- Reference genome in FASTA format (currently must be finished assembly, only 1 chromosome)
- Fragment length (default is 1000bp)
- Step size (default is 100bp, set to 1000bp for non-overlapping windows)

-what does output look like? A FASTA file named by your input genome, containing all of the genomic fragments

```
$ python slice_ref_genome.py -r reference.fasta -f 500 -s 200
```

-Let's say you want to run this script on a set of genomes. You could run a simple for loop. Enter the directory of your FASTA files, then:

```
$ for file in `find . -maxdepth 1 -name "*.fasta"`; do name=$(echo $file | cut -f 2 -d "/"); slice_ref_genome.py -r $file; done
```

-you could then cluster with USEARCH and use the resulting centroids in your LS-BSR analysis

***Disclaimer**

TGen and ITS Affiliates, representatives and employees make no representations, warranties, guarantees, or claims, of any kind or nature, as to the accuracy of, reliability, utility, performance, effectiveness, or otherwise, of the Software or the results obtained therefrom, nor do they assume any responsibility for the results, quality of results, or lack of results. THE SOFTWARE IS PROVIDED "AS IS" AND WITHOUT WARRANTY OF ANY KIND, WRITTEN OR ORAL, EXPRESS OR IMPLIED, DIRECT, INDIRECT, BY ESTOPPEL, OR OTHERWISE, AND SPECIFICALLY EXCLUDING ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT OF ANY THIRD PARTY PATENT OR OTHER INTELLECTUAL PROPERTY RIGHTS, AND ANY OTHER WARRANTIES PROVIDED FOR BY THE UNIFORM COMMERCIAL CODE, USAGE IN THE INDUSTRY, OR COURSE OF DEALING BETWEEN THE PARTIES. TGEN AND IBBL DO NOT WARRANT THAT USE OF THE SOFTWARE WILL BE ERROR FREE, UNINTERRUPTED, SECURE, OR VIRUS-FREE. TO THE EXTENT TGEN AND IBBL MAY NOT AS A MATTER OF LAW DISCLAIM ANY WARRANTY, THE PARTIES AGREE THAT THE SCOPE AND DURATION OF ANY SUCH WARRANTY SHALL BE THE MINIMUM PERMITTED UNDER APPLICABLE LAW.

LIABILITY LIMITATION

TGEN SHALL NOT BE LIABLE FOR ANY DAMAGES WHATSOEVER, INCLUDING ACTUAL OR DIRECT DAMAGES, AND SHALL NOT IN ANY EVENT BE LIABLE FOR ANY INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF INAPPROPRIATE, SUBSTITUTE, OR ALTERNATIVE, SOFTWARE, LOSS OF USE, LOSS OF DATA, LOSS OF PROFITS, LOSS OF REVENUES, OR BUSINESS INTERRUPTION) HOWEVER CAUSED, RESULTING FROM, RELATED TO, OR ARISING OUT OF THE USE OF THE SOFTWARE, REGARDLESS OF FORESEEABILITY AND NOTWITHSTANDING WHETHER RECIPIENT HAS BEEN ADVISED OF THE POSSIBILITY OF ANY SUCH DAMAGES, AND REGARDLESS OF THE FORM OF THE CAUSE OF ACTION, WHETHER IN CONTRACT, BREACH OF WARRANTY, OR TORT (INCLUDING NEGLIGENCE, STRICT LIABILITY, OR OTHERWISE), OR OTHERWISE.