The whole genome focused array SNP typing (*WG-FAST*) pipeline

Citation: Jason Sahl, James Schupp, David Rasko, Rebecca Colman, Jeffrey Foster, Paul Keim. Phylogenetically typing bacterial strains from partial SNP genotypes observed from direct sequencing of clinical specimen metagenomic data. Genome Medicine. Accepted and in press.

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

What does *WG-FAST* do?

WG-FAST was designed as a tool to phylogenetically genotype unknown samples, even those with extremely low read coverage, in the context of a well-studied dataset.

What does *WG-FAST* not do?

WG-FAST is not intended to identify new SNPs in a dataset. If too many samples are processed with *WG-FAST*, a phylogenetic discovery bias can exist.

Installation

-The code is housed here: https://github.com/jasonsahl/wgfast.git

-Install the code with:

```
>git clone https://github.com/jasonsahl/wgfast.git
```

The following line in "wgfast.py" must be edited to reflect your installation directory:

```
WGFAST_PATH="/Users/jsahl/wgfast"
```

Unit tests

-To test that all functions are working correctly in *WG-FAST*, type:

```
python tests/test_all_functions.py
```

If everything is correct, all tests should pass.

Dependencies not included with *WG-FAST*:

1. GATK – tested version is 2.72. This version requires Java 1.7; known issues have been observed with Java 1.8 and GATK. Should be back compatible with older versions. **Download Jar file and place in WGFAST_PATH/bin**. Can be obtained from: https://www.broadinstitute.org/gatk/download
2. Samtools – tested version is 0.1.19. Must be in PATH as "samtools". Can be obtained from: http://samtools.sourceforge.net/

3. BWA-MEM – tested version is 0.7.5a. Must be in PATH as "bwa". Can be obtained from: http://bio-bwa.sourceforge.net/
4. BioPython – must be in PYTHONPATH. Can be obtained from: https://github.com/biopython/biopython

Additional dependencies included with *WG-FAST*.

5. Trimmomatic. Included with *WG-FAST*. Can also be obtained from: http://www.usadellab.org/cms/?page=trimmomatic
6. Picard tools – tested version is 1.79. Included in "WGFAST_PATH/bin" directory. Can also be obtained from: http://broadinstitute.github.io/picard/
7. RAxML – tested version is 8.1.7. Must be in PATH as "raxmlHPC-SSE3" and "raxmlHPC-PTHREADS-SSE3", if the sub-sample routine will be used. Can be obtained from: https://github.com/stamatak/standard-RAxML. The PTHREADS version does not support the ASC substitution models. Because RAxML is under constant development, a stable version is included with *WG-FAST* (see build directions in the standard-RAxML directory ("README"). As improvements are made to RAxML, the version will be updated in *WG-FAST*, as long as changes don't affect performance.
8. DendroPy – tested version is 3.12.0, must be installed in PYTHONPATH. Can be obtained from: https://github.com/jeetsukumaran/DendroPy. Dendropy is also included with *WG-FAST* with the following included information:

Copyright 2009-2010 Jeet Sukumaran and Mark T. Holder

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL JEET SUKUMARAN OR MARK T. HOLDER BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Citations for all dependencies

BioPython: Cock, P.J. *et al.* Biopython: freely available Python tools for computational molecular biology and bioinformatics. *Bioinformatics* **25**, 1422-3 (2009).

Trimmomatic: Bolger, A.M., Lohse, M. & Usadel, B. Trimmomatic: a flexible trimmer for Illumina sequence data. *Bioinformatics* **30**, 2114-20 (2014).

BWA-MEM: Li H. Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM. arXivorg. 2013(arXiv:1303.3997 [q-bio.GN])

RAxML v8: Stamatakis, A. RAxML version 8: a tool for phylogenetic analysis and post-analysis of large phylogenies. *Bioinformatics* (2014).

RAxML evolutionary placement algorithm: Berger SA, Krompass D, Stamatakis A. Performance, accuracy, and Web server for evolutionary placement of short sequence reads under maximum likelihood. Syst Biol. 2011;60(3):291-302.

Samtools: Li H, Handsaker B, Wysoker A, Fennell T, Ruan J, Homer N, Marth G, Abecasis G, Durbin R, Genome Project Data Processing S. The Sequence Alignment/Map format and SAMtools. Bioinformatics. 2009;25(16):2078-9.

DendroPy: Sukumaran J, Holder MT. DendroPy: a Python library for phylogenetic computing. Bioinformatics. 2010;26(12):1569-71. Epub 2010/04/28. doi: 10.1093/bioinformatics/btq228. PubMed PMID: 20421198.

GATK: McKenna A, Hanna M, Banks E, Sivachenko A, Cibulskis K, Kernytsky A, Garimella K, Altshuler D, Gabriel S, Daly M, DePristo MA. The Genome Analysis Toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data. Genome research. 2010;20(9):1297-303.

Required input files

1. **Directory of sequence reads**. The reads must be named according to Illumina HiSeq or MiSeq conventions. Reads must be in the Illumina 1.9+ FastQ format. If you have old Illumina FastQ encodings, they must be converted before running *WG-FAST*. **Important: names must not have periods ".", brackets "[]", or other weird characters "=:" in the header.**
2. **SNP matrix**. The easiest way to generate this is by using NASP (https://github.com/TGenNorth/NASP). If other SNP matrix formats are used, they must conform to having the first column including (contig::coordinate) and the column following the SNP calls must be (#SNPcall). For the sub-sampling routine to complete, a genome must be present in your matrix that is called 'Reference'. **Important: sample names must not have periods "." in the header.**

3. **Phylogeny**. A script is included with *WG-FAST* that can generate an appropriate phylogeny from a NASP matrix (see below). This script also generates a 'Parameters' file, which can be used with *WG-FAST* and cuts down on the computational time required for each subsequent run. The best way to guarantee downstream compatibility is to generate the phylogeny with RAxML, which is used by the wgfast_prep.py script described below.
4. **Reference genome in FASTA format**. This should be the same FASTA that was used to call SNPs with NASP.

Complete list of Arguments:

-h, --help show this help message and exit
-m MATRIX, path to NASP-formatted SNP matrix [REQUIRED]
-t TREE, path to input tree in Newick format [REQUIRED]
-r REFERENCE, path to reference fasta [REQUIRED]
-d DIRECTORY, path to directory of fastq.gz files [REQUIRED]
-x PARAMETERS, parameters file for RAxML insertion [OPTIONAL]
-p PROCESSORS, # of processors to use - defaults to 2
-c COVERAGE, minimum SNP coverage required to be corrected, default = 3x
-o PROPORTION, proportion of alleles to be corrected, defaults = 0.9
-k KEEP, keep temporary files? Defaults to F (T or F)
-s SUBSAMPLE, run subsample routine? Defaults to T (T or F)
-n SUBNUMS, number of subsamples to process, defaults to 100
-g DOC, run depth of coverage on all files? Defaults to T (T or F)
-e TMP_DIR, temporary directory for GATK, default = "/tmp" (Must be PATH)
-z insertion method (MP or ML), defaults to ML, MP not well tested
-f How close does a subsample have to be from true placement? Defaults to 0.1 (Float)
-y Only run sub-sample routine and exit? Defaults to F (T or F)
-j MODEL, which model to run with raxml (GTRGAMMA, ASC_GTRGAMMA)

Test Data:

1. To give *WG-FAST* a try to make sure everything is installed correctly, check out the test_data directory.
2. The following command was run from a "run" directory created within the *WG-FAST* installation:

```
$python ../wgfast.py -m ../test_data/nasp_matrix.tsv –t ../test_data/nasp_raxml.tree
-r ../test_data/reference.fasta -d ../test_data/reads -x ../test_data/nasp.PARAMS \
-c 1
```

What is *WG-FAST* doing?

WG-FAST is a pipeline that wraps other tools together to place the samples into a phylogenetic context, based on a well-characterized dataset. The order of functions conducted by *WG-FAST* include:

1. Adapters are trimmed using Trimmomatic. A list of adapters is included in the "bin" directory ("illumina_adapters_all.txt") within the *WG-FAST* distribution. This file includes all of the standard Illumina adapters that I could find, plus additional ones that we use in our laboratory. If you have different adapters than these, add the sequences into this file when you run *WG-FAST*. The minimum length of sequences to keep is hard coded as 50, but I could change this value to reflect different lengths if needed.
2. A dictionary is created from the Reference fasta with Picard Tools and the reference fasta is indexed with samtools.
3. Reads are paired into a single sample based on their names, although single end reads are also supported. *WG-FAST* assumes that names be something like: *S1_R1_001.fastq.gz" or "R1_001.fastq.gz". If the name pairings aren't recognized, they will be run as single ended.
4. Reads are mapped to the reference with BWA-MEM, using default settings. SNPs are then called with the UnifiedGenotyper method in GATK, using the "EMIT_ALL_CONFIDENT_SITES" method.
5. If selected, coverage across the reference is calculated with the "DepthOfCoverage" method in GATK.
6. SNPs are compared against the SNPs in your initial SNP matrix. If the SNP is missing from your VCF file, a "-" is inserted. Furthermore, if a position fails either the user-defined thresholds for depth or proportion, the position is replaced with a "-". The number of discarded SNPs, or those that didn't pass the filters, is reported.
7. The new matrix is converted into a multi-fasta and processed with the evolutionary placement algorithm in RAxML v8, placing the unknown into a user-provided phylogeny. The resulting tree is converted into Nexus, such that the unknowns can be easily visualized (Red) in FigTree (http://tree.bio.ed.ac.uk/software/figtree/).
8. For the optional sub-sampling routine, the following functions are performed:
    a. For each query, the two closest genomes are identified, based on lowest pairwise patristic distances calculated by DendroPy
    b. Two matrices are created for each query, one for each of the nearest neighbors. Each of these contains the name of the query and the name of the neighbor, ending in "tmp.matrix". If you generate these files (and keep them), then run *WG-FAST* in "-y T" mode, these files will be skipped and will not need to be created again.
    c. Each matrix is then converted into a FASTA file and a parameters file is created with RAxML. The thought process here is that once the parameters files are created, they can be used for future comparisons. If you run *WG-FAST* with "-y T", you can re-use these parameters files and won't need to generate again. This method uses the PTHREADS method to take advantage of multiple processors. Currently, only four separate parameters files can be generated concurrently; this value is hardcoded into the script. This is due to the large computational demand required by this method, but could be made into a tunable parameter, if needed.
    d. The initial tree is pruned of the genome to be re-inserted. Each sub-sampled neighbor is then inserted back into the pruned phylogeny, using the GTRGAMMA method, to stay consistent with the parameters file. The patristic distance is then calculated between the neighbor and the "Reference". This is done because if a NASP-formatted matrix is generated, the reference FASTA file will always be called "Reference". The "true" distance is also identified based on the original SNP matrix using all of the original source data. The sub-sampled distance is then divided by the true distance in

order to get a ratio of placement replication. If this value falls within a user-defined threshold, then the placement is considered to be correct. For each unknown, the number of incorrect placements is divided by the total number of placements in order to get an idea of how stable the placement is in that region of the tree at the level of SNP density provided by the unknown. The resulting p-value can then be used to assess the robustness of a placement, based on significance values chosen by the user.

Output printed to screen:

1. The parameters that you called. Default values are also printed so the read can be reproduced.
2. Number of callable positions, including polymorphic and monomorphic positions. These are all positions called in each sample, compared to the reference. This is the number prior to any filtering due to mixed SNP positions.
3. Number of SNPs. These are the number of observed polymorphisms, based on calls made by GATK.
4. Number of discarded SNPs. These are polymorphisms that were called by GATK, but were thrown out because they failed to meet the depth and/or proportion filters.
5. Insertion likelihood values. The higher the likelihood value and the fewer the number of possible insertion nodes, the more trusted the placement, although caveats exist (see Manuscript).
6. If sub-sample routine is invoked, information is also available for how often the sub-sample was placed correctly. A sub-sample is considered to be "correct" by comparing the patristic distance from the sub-sampled genome to the "Reference", then comparing that to the distance between the un-sub-sampled genome to the "Reference". If this ratio falls within the "fudge-factor" range, then it is considered to be correct. The number of times that the sub-sample falls within this range is divided by the total number of iterations and a p-value is reported.

Files generated by *WG-FAST*:

1. "transformed.tree". This file is a nexus file of the starting tree with the unknowns placed. If this file is opened with figtree (http://tree.bio.ed.ac.uk/software/figtree/), the unknown isolates are shown in red.
2. "nasp_matrix_with_unknowns.txt". This file is a modification of the original NASP matrix with the unknowns inserted, so the user can identify specific SNPs. Information is lost from this matrix, so it cannot be used with additional *WG-FAST* runs.
3. "coverage_out.txt". If you choose to have genome coverage information, the coverage across the reference genome is provided in this file.
4. "all_patristic_distances.txt". This is the patristic distance, or tree path distance, between all pairwise comparisons in your analysis.

Additional scripts included with *WG-FAST*:

1. wgfast_prep.py

-What does it do? Given a NASP matrix, this script will generate a maximum likelihood phylogeny with RAxML and will also generate a "parameters" file that can be used for future *WG-FAST* runs. The use of a parameters cuts down on the computational requirements when running additional *WG-FAST* runs using the same input files.

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

- Python > 2.7 < 3.0
- NASP matrix
- RAxML in your $PATH as "raxmlHPC-SSE3"

-What does the output look like? Two files are produced:

1. "nasp_raxml.tree". Your tree. Names have been fixed to work with *WG-FAST*. Make sure that names do not include periods.
2. "nasp.PARAMS". Parameters file. Use this with the "-X" flag described above.

3. Example usage:

```
$python wgfast_prep.py -m nasp.matrix
```

-Note: if you run this script with a model separate from ASC_GTRGAMMA, you will also need to use this model when running the main wgfast script, if you use the parameters file.

2. subsample_snps_pearson.py

-What does it do? Given a NASP matrix, the script generates a new matrix over a given number of iterations at a given level of SNP sampling. The script then conducts a Mantel test using the Pearson correlation between the original similarity matrix and a matrix sampled at a given SNP density.

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

- Python > 2.7 < 3.0
- NASP matrix
- 'mothur' executable in your PATH. Mothur can be freely obtained from: http://www.mothur.org/wiki/Download_mothur

-What does the output look like? One file is generated "results.txt", that is new-line delimited, with each line containing a Pearson correlation value (0 to 1).

-Example usage:

```
$python subsample_snps_pearson.py -m nasp.matrix -s 100
```

3. subsample_reads_and_place.py

-What does it do? This script helps sub-sample your SNP matrix and identify how robust specific regions of the phylogeny are. This script will prune the genome from the phylogeny, randomly subsample the SNPs at a given level *n* separate times, then will re-insert into the tree. The distance between the "correct" placement will be compared to the re-sampled placement. This is a computationally intensive method and works best with a job management system (see below).

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

- Python > 2.7 < 3.0
- NASP matrix
- Phylogeny
- Name of genome to test
- Dendropy

-What does the output look like? For each genome, you will get an output file, showing each iteration (1st column), the number of correct placements (2nd column) and the best SNP level, if one is identified. For example:

```
100 1
200 10
optimal value is for Salmonella_enterica_subsp_enterica_serovar_Uganda_str_R8-
3404nucmer is 200
```

-Example usage:

```
$python subsample_reads_and_place.py -m matrix.txt -t nasp_raxml.tree -n name -s 100
-o 6 -e 10000
```

-Now let's say you want to run this for each genome in your phylogeny using a job management system. First, let's extract all of the genome names out of your tree:

```
$ python extract_tree_names.py -t nasp_raxml.tree > genome_names.txt
```

-We use Torque for job management, but this concept should work for others. Make a PBS script that looks something like:

```
PBS -N confidence
#PBS -l walltime=100:00:00
#PBS -l nodes=1:ppn=6
#PBS -l mem=12gb
#PBS -q batch
#PBS -m a
#PBS -j oe
cd $PBS_O_WORKDIR
```

python subsample_reads_and_place.py -m matrix_polymorphic.txt -t nasp_raxml.tree -n $NAME -s 100 -o 6 -e 10000

-Then you can run a for loop to submit each one of your jobs:

for file in `cat genome_names.txt`; do qsub –v NAME=$file pbs_script.sh; done