

VAAL manual (version 1.0, 7/8/08)

Overview. VAAL is a polymorphism discovery algorithm for short reads. To run it, you provide reads (and quality scores) from a ‘sample genome’ as input, along with a vector sequence to trim from the reads, and a reference sequence for a related genome to compare to. VAAL produces as output a an assembly for the sample genome, together with a mask showing which bases are ‘trusted’. VAAL then deduces from that a list of differences between the sample and related genomes.

Alternatively, VAAL can be provided as input read data for two sample genomes, together with a reference sequence for a related genome. In this case, VAAL produces assemblies for each of the sample genomes, and compares them to each other, thereby deducing a list of differences between them.

VAAL has been tested on bacteria, using single lanes of 36 bp unpaired reads from the Illumina platform.

Software availability. This first version of VAAL is available at <ftp://ftp.broad.mit.edu/pub/crd/VAAL/VAAL1.0.tar.gz>. Subsequent revisions will be made available at <http://www.broad.mit.edu/crd>. (Technical note: this release ‘1.0’ was produced by ‘MakeTar.csh VAALrun TruePoly’ from a cvs checkout dated 2pm EST 7/4/08.)

Support. VAAL is supported software. Requests for help should be sent to crdhelp@broad.mit.edu. Please put ‘VAAL’ in the subject heading.

System requirements. For bacterial genomes, VAAL uses roughly 10 GB of memory. VAAL should work on most Unix or Linux systems, but some changes may be needed. The current version is hardwired to require gcc version 4.3.0, but is likely to work with other recent gcc versions, with very small changes to the code.

Installation. Create a new directory and ‘cd’ to it. Type `zcat VAAL1.0.tar.gz | tar xf -`. This will unpack the source code into the directory. Type ‘make’ or ‘make -jn’. You will find a directory starting with ‘bin_’. Put this in your path. Make sure that your LD_LIBRARY_PATH is set appropriately for your version of gcc. Now you should be all set.

Preparing the inputs.

- Pick a directory *dir* where you will perform the calculations.
- Pick a project name. We’ll call this *reads*, but you may substitute any string that does not have a dot or slashes in it.
- Inside *dir*, create a directory named *reads*.
- Inside *dir/reads*, create two files *reads.fasta* and *reads.qual*. These may be links. The *reads.fasta* file should be in fasta format. The *reads.qual* file should be formatted like a fasta file, but instead of bases, have integer quality scores (between 0 and 255), separated by spaces.
- Inside *dir/reads*, create a file *ref.fasta*, the reference sequence for the related genome. You may use anything in place of *ref* (but no slashes, please). This may also be a link.

Running VAAL. Change directory to *dir*, and type `VAALrun SOURCE=reads REF=ref VECSEQ=v` where *reads* and *ref* are as above, without fasta extensions, and *v* is a vector sequence to be trimmed from the right side of the reads. For example, an appropriate value for unpaired Illumina reads is at present
AGATCGGAAGAGCTCGTATGCCGTCTTCTGCTTGAAAAAAAAAAAAAAAAAAAAAAAAAAAAA.

Optional arguments.

- You can see all arguments and default values by typing simply ‘VAALrun’.
- If you specify `RUN=runname`, *runname* will be the name of the run. Otherwise, your user name is used as the name of the run. See below for how the name of the run is used.
- If you specify `TRUNC=n`, reads will be truncated to *n* bases.
- If you specify `TRUE_REF=sample`, VAAL will look for a file *sample.fasta* in *dir/reads*, and use it to assess the correctness of its results. Of course to do this you need to be carrying out a controlled experiment for which you know exactly what the genome of the sample is in advance.

Finding the results directory. In *dir/reads*, VAAL will create a subdirectory *vs_ref*, where *ref* is the name you used for the related reference sequence. Inside this directory is a directory *runname* (see optional argument RUN, above). This directory contains the results. If you specified the TRUNC argument, the *vs_* subdirectory will instead be inside another subdirectory *trunc_n*.

Finding the assisted assembly. In the results directory, there will be files *reads.assembly.fasta* and *reads.assembly.trusted.txt*. The first is the assembly of the sample genome in fasta format, and the second is a human-readable fasta-style ‘vecbivector’ showing which bases on the assembly are ‘trusted’.

Finding the differences. Inside the results directory is a file *reads.VAAL4.K28.out*, containing the list of all observed differences (SNPs or indels) between the sample genome and the related genome. Composite events are decomposed into their constituent SNPs and indels. There are two header lines, followed by one line per difference, followed by summary statistics. A difference line for a SNP looks like

```
0 20895 left=TGGATCAGACCTTTAGCAGC sample=C ref=G right=TGACGGTCCACGATCGGTTG
```

which is interpreted as follows: on zero-based reference contig 0, at zero-based position 20895, the sample shows C but the reference shows G. The ‘left’ and ‘right’ arguments specify 20 flanking bases on each side, which may be used as a sanity check on coordinates. For a two-base insertion in the sample, one might see instead

```
0 2352451 left=GACCTGGCGAATCGCCTCTT sample=CG ref= right=TTTGTCTCCGGCCTTTTCGC
```

indicating that CG is inserted at the given position. For deletions, ‘sample’ is empty but ‘ref’ is not.

Alternatively, the file *reads.VAAL4.K28.compound.assembly* exhibits all polymorphisms, sorted into groups, with composite events grouped together.

Comparison to truth data. If you specified the `TRUE_REF` argument, there will be a file *reads.VAAL4.K28.compound.truepoly*, providing a list of all ‘callable’ polymorphisms, in the same format as the *compound.assembly* file, a file *reads.VAAL4.K28.compound.events*, listing all missed compound events and false positive calls, and a file *reads.VAAL4.K28.compound.subevents* containing some summary statistics.

Comparing two assemblies. If you have sequenced two samples, for which you have a common reference sequence, you can compare them using

```
TruePoly REFA=reads1.assembly.fastb TRUSTEDA=reads1.assembly.trusted  
REFB=reads2.assembly.fastb TRUSTEDB=reads2.assembly.trusted
```

where the *reads...* arguments are full paths of files in the results directories for the two samples. The coordinates of the polymorphisms will be on the sample 2 assembly. Composite events are decomposed into their constituent SNPs and indels.