# Variant Annotation, Analysis and Search Tool

# User's Guide

**12 December 2011**

**University of Utah, Department of Human Genetics
Omicia Inc.**

# Table of Contents

VAAST User's Guide - Version 1.0.1                                                     2

## Introduction

VAAST, the Variant Annotation, Analysis and Search Tool is a probabilistic search tool for identifying damaged genes and their disease-causing variants in personal genome sequences. VAAST builds upon existing amino acid substitution (AAS) and aggregative approaches to variant prioritization, combining elements of both into a single unified likelihood-framework that allows users to identify damaged genes and deleterious variants with greater accuracy, and in an easy-to-use fashion. VAAST can score both coding and non-coding variants, evaluating the cumulative impact of both types of variants simultaneously. VAAST can identify rare variants causing rare genetic diseases, and it can also use both rare and common variants to identify genes responsible for common diseases.

## License

VAAST was developed as a collaboration between the Yandell Lab at the University of Utah and Omicia, Inc. of Emeryville, CA. The University of Utah freely licenses VAAST for academic research use. For commercial, clinical and all other uses please contact Martin Reese <license@omicia.com> at Omicia, Inc.

To obtain an academic research license please visit the Yandell Lab website at http://www.yandell-lab.org/software/vaast.html

## Installation

Installation of VAAST is pretty simple.  Unpack the VAAST_Code_1.0.1.tar.gz file and you're ready to go.

```
tar –xzvf VAAST_Code_1.0.1.tar.gz
```

The main scripts that run a VAAST analysis are in the bin directory and the accessory scripts are in the bin/vaast_tools directory.  All of the Perl dependencies are included with the package.   One of the included Perl modules, Set::IntSpan::Fast::PP has a size limitation on integers that can cause an 'Out of Range' error on some variant files.  We have never seen these errors with human variant data, but we have seen it when using VAAST with non-human variant files. You can avoid (or fix) this problem by installing the compiled version Set::IntSpan::Fast::XS

```
sudo cpan Set::IntSpan::Fast::XS
```

The accessory script vaast_imager has additional requirements that are discussed below with a description of that tool.

## Basic Workflow

VAAST combines variant frequency data with AAS effect information on a feature-by-feature basis using the likelihood ratio described in the VAAST paper to compare the variants in a set of disease genomes to those in a set of healthy genomes.

The set of genomes being analyzed for disease causing features are referred to below as the target genomes.  The set of healthy genomes that the target genomes are compared to are referred to as the background genomes.

The basic inputs to VAAST consist of a set of single nucleotide variants (SNVs) for the target and background genomes, a set of genomic features (typically genes) that will be scored, and a copy of the appropriate reference genome.

Provided with these inputs the VAAST analysis pipeline occurs in three basic steps:

1. Variant Annotation – The variants for the individual genomes are annotated for the effects that they cause on the genomics features.  Common SNV effects on genes are synonymous and non-synonymous codon changes, stop-codon loss and gain and splice-site variants.
2. Variant Selection – Some set of (or commonly all) the variants in the target genomes are combined together for comparison against the set of all background variants.  Individual research designs may however call for selection of a subset of variants, for example all variants shared by affected family members, but not present in unaffected family members and VAAST has tools to do this.
3. Variant Analysis – All the variants found in the target genomes are then compared to all the variants in the background genomes and the features that contain those variants are scored and ranked by the likelihood that they are disease causing.

## Connecting and Getting Help

A  step-by-step tutorial style guide is provided with the VAAST package in the docs directory and also is available on the Yandell Lab website:

docs/VAAST_Quick-Start-Guide.pdf
http://www.yandell-lab.org/software/ VAAST_Quick-Start-Guide.pdf

This guide walks through several example analyses with data included with the VAAST distribution.

The main source of information for downloading software and data and getting help with the VAAST package is the VAAST website.  Please visit us at:

http://www.yandell-lab.org/software/vaast.html

If you experience difficulties with the VAAST software that you cannot resolve from reading the documentation please join and ask questions on the VAAST users mailing list:

http://yandell-lab.org/mailman/listinfo/vaast-user_yandell-lab.org

If you think you have found a bug in the VAAST code we would love to get a bug report from you on the mailing list also.


## Input File Formats

VAAST uses three common input file formats and produces two additional output file formats that are unique to the VAAST software.  All of these file formats are text files and where appropriate are tab-delimited.

### Fasta File Format

The Fasta file format is one of the most ubiquitous formats in bioinformatics and yet oddly it doesn't have an official specification  - primarily because it's so simple it doesn't really need one.  Have a look at the Wikipedia page for the Fasta format if you'd like a lot of detail and a little bit of interesting history and trivia (http://en.wikipedia.org/wiki/FASTA_format).

Fasta files are used within VAAST to provide a copy of the reference genome sequence.  The requirements of the Fasta format for VAAST are pretty straightforward.  Each sequence (a chromosome or a contig) begins with a header line.  This header line in turn begins with the '>' character.  The first contiguous set of non-whitespace characters after the '>' are used as the ID of that sequence.  This ID must match EXACTLY the 'seqid' column described below for the sequence feature and sequence variants.  On the next and subsequent lines the sequence is represented with the characters A, C, G, T, and N.  All other characters are disallowed.  The sequence lines can be of any length, but they must all be the same length, except the final line of each sequence which can terminate whenever necessary at the end of the sequence.  Below is an example of a short Fasta file.

```
>chr1
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNTCTATCAACTTTCTAAGTATAACCACCAATTGaataaac
atcaactaaaaagcatttttctattcccttgcatgtgttagctcattaaa
tcctcacaaaagccccatgaagcagggatcaacaccattttacaaaaaaa
aaaaaaaaaaaaaaaagcacaaaaattacctatgggcacaGAATACAAG
TCAAAATGCATGCCACAATATGGCTAAAATAATGATGTTCTTTAAATAGA
>chr2
TCCTCCATATGATGTCAGTGTCCTCTGTATGACATCAATATCCTCCATAC
GATGCCCCTGTCCTTCATATGATGTCAGTGTCCTTTTGTGAGCACCAGTG
TCCTTTGTATGACATCAGTAGTCTCCCATGAATGTCACTGTCTTCCCATA
GATGTCAGTGTCCTCTccaaaagacaagcagaagctgtNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
```

## GFF3 File Format

The GFF3 (General Feature Format version 3) file format is a widely used format for annotating genomic features.  Although various versions of GTF and GFF formats have been in use for many years, GFF3 was introduced by Lincoln Stein to standardize the various gene annotation formats to allow better interoperability between genome projects.  The Sequence Ontology group currently maintains the GFF3 specification.   The official GFF3 specification can be found on the Sequence Ontology website at:

http://www.sequenceontology.org/resources/gff3.html

Briefly, a GFF3 file begins with one or more lines of pragma or meta-data information on lines that begin with '##'.  The only required pragma is '## gff-version 3', although VAAST tools can make use of other GFF3 pragmas if they are available.  Header lines are followed by one or more (usually many more) feature lines.  Each feature line describes a single genomic feature.  Each feature line consists of nine tab-delimited columns.  Each of the first eight columns describes details about the feature and its location on the genome and the final line is a set of tag value pairs that describe attributes of the feature.  A full description of GFF3 is beyond the scope of this document, so please refer to the specification noted above for more detail.  GFF3 files for use with VAAST can be downloaded from the VAAST website.  A short example of a GFF3 file is given below.

```
##gff-version 3

##genome-build hg19
##sequence-region chr11 1 135006516

chr11  UCSC_refGene  gene  71950121  71955220 . - . ID=PHOX2A
chr11  UCSC_refGene  mRNA  71950121  71955220 . - . ID=NM_005169;Parent=PHOX2A;
chr11  UCSC_refGene  exon  71950121  71951242 . - . ID=NM_005169:exon:01;Parent=NM_005169
chr11  UCSC_refGene  CDS   71950793  71951242 . - 0 ID=NM_005169:CDS:01;Parent=NM_005169
chr11  UCSC_refGene  CDS   71952146  71952333 . - 0 ID=NM_005169:CDS:02;Parent=NM_005169
chr11  UCSC_refGene  exon  71952146  71952333 . - . ID=NM_005169:exon:02;Parent=NM_005169
chr11  UCSC_refGene  exon  71954832  71955220 . - . ID=NM_005169:exon:03;Parent=NM_005169
chr11  UCSC_refGene  CDS   71954832  71955048 . - 1 ID=NM_005169:CDS:03;Parent=NM_005169
```

## GVF File Format

The GVF (Genome Variation Format) is a file format developed and supported by the Sequence Ontology group for use in describing sequence variants.  It is based on the GFF3 format and is fully compatible with GFF3 and tools built for parsing, analyzing and viewing GFF3.  The GVF specification can also be found on the Sequence Ontology website at:

http://www.sequenceontology.org/gvf.html

GVF shares the same nine-column format for feature lines as found in GFF3, but specifies additional pragmas for use at the top of the file and additional tag/value pairs to describe feature attributes in column nine that are specific to variant features.  An example of GVF file is given below.

```
##gvf-version 1.04

##individual-id ID=NA19240;Gender=female;
##sequence-region chr1  1 247249719

chr1  SOLiD  SNV  886339  886339  .  +  .  ID=01;Variant_seq=G;Reference_seq=A;
chr1  SOLiD  SNV  887427  887427  .  +  .  ID=02;Variant_seq=C;Reference_seq=T;
chr1  SOLiD  SNV  888186  888186  .  +  .  ID=03;Variant_seq=C;Reference_seq=T;
```

VAAST provides a tool, 'vaast_converter', for converting SNV variant files in VCF format to GVF format.  VCF is another common format for describing sequence variants that was developed by the 1000 Genomes Project (http://www.1000genomes.org/node/101).

In addition to the GVF specification mentioned above, a set of 10 whole genome sequence variant files referred to as the 10Gen data set can be found in GVF format at:

http://www.sequenceontology.org/resources/10Gen.html

The following Genome Biology publication describes the GVF format and the 10Gen dataset.

http://www.ncbi.nlm.nih.gov/pubmed/20796305

## Output File Formats

VAAST tools produce two file formats that are unique to the VAAST package.  These formats are mentioned briefly here and are described in more detail with the tools that produce them below.

A condenser or CDR file is produce by VST for input to VAAST.  CDR files have a '.cdr' extension.  The CDR format is a condensed representation of the variants for a set of genomes (the target and background genomes) and is used as input to VAAST.

The final output of a VAAST analysis is a VAAST file with a '.vaast' extension.  This file contains information for every feature in the genome, including the features rank, significance level, raw score from the composite likelihood ratio test and the variants that occur within that feature in both the target and background genomes.  Note that an additional file ending in a '.simple' extension is also produced by VAAST.  This file just has a minimal set of the information found in the full VAAST output file listed with one feature per line.

# Data Sources

Input files to VAAST are available for download from the VAAST website.

http://www.yandell-lab.org/software/VAAST/data/

The files available are divided into two directories one for the 2006 hg18 (NCBI Build 36) assembly and one for the 2009 hg19 (GRCh37) assembly. It is critical that you know which assembly your variants are annotated on and that you use the appropriate input files from that same assembly when running VAAST.

## Fasta Files

The Fasta files used by VAAST represent the genome assembly that was used as the reference genome for variant calling.  Two Fasta files are currently available and are copies of the associated file provided by the UCSC Genome Browser:

vaast_hsap_chrs_hg18.fa
vaast_hsap_chrs_hg19.fa

### GFF3 Files

The GFF3 files used by VAAST are based on NCBI RefSeq annotations that have been mapped to hg18 and hg19 assembly coordinates by the UCSC Genome Browser team. The data in the UCSC table refGene was converted to GFF3 by the VAAST development team and one file is available for each assembly.

refGene_hg18.gff3
refGene_hg19.gff3

### GVF Files

The GVF files containing the genome variants are the data that you provide VAAST for your analysis. These files may come to you in GVF format from your sequence provider, or you may need to convert your existing file to GVF format. VAAST provides a tool vaast_converter for converting variant files from VCF format to GVF. Contact the VAAST community through the vaast-user@yandell-lab.org mailing list for information on converting other file formats to GVF. GVF files input into VAT and VST must be sorted. They will often come sorted from the data supplier, but if they are not (or you're not sure) see the section below on VAT for a simple command line tool that will sort GVF files. Once they have been sorted once, VAT will keep them in sorted order, so it is not necessary to sort them again before running VST.

### Background CDR Files

Any set of genomes that you wish to use as a background file of healthy genomes can be converted to CDR files with VST as described below. VAAST can reuse the same background CDR files for many different analyses. The VAAST data repository contains background CDR files for the following datasets:

### Low Coverage 1000 Genomes (July 2010) 179 Individuals

1KG_179_refGene_hg18.cdr
1KG_179_refGene_hg19-liftover.cdr

### 10Gen Dataset - 10 Individuals

10Gen-refGene-hg18.cdr
10Gen-refGene-hg19-liftOver.cdr

# Individual Tools

## VAT - Variant Annotation Tool

VAT calculates a rich set of annotations on the effects that variants have on genomic features.  These effects, like synonymous and non-synonymous codon changes and stop-gain and loss provide additional information about the underlying variant, which VAAST uses to score features.  VAT annotates other classes of variant effect such as splice donor/acceptor/region, UTR variants and others; these additional annotations will be used by VAAST in a future release.  VAT needs to know which assembly you are working with and the gender of the individual represented in the variant file.  These can be annotated in the GVF variant file, but if they are not, they can be given on the command line.  If this information is not provided VAAST will try to guess the gender from the ratio of heterozygous variants on chromosome X (see VAT documentation for more detail), and will get chromosomal boundaries from the length of the sequences in the Fasta file.  VAT prints its output to STDOUT and this should be redirected to a file as shown below.

### VAT Input

The inputs to VAT are a genome annotation file (containing gene models for the given genome) in GFF3 format, a sequence file containing the assembly (the chromosomes and/or contigs that make up the genome) in Fasta format and a GVF file containing the SNVs found within an individual genome.  The features in the GFF3 and GVF file MUST be sorted for maximum performance, and this can be done very easily with the supplied vaast_sort_gff tool:

```
vaast_sort_gff -in_place file1.gff3 file2.gvf
```

Note that this same tool will sort both GFF3 and GVF files.  Several VAAST tools index GFF3, GVF and fasta files so that features, ranges of features and sequences may be accessed without loading the entire file.  These files have '.vfi' and '.vgi' extensions for Fasta and GFF3/GVF files respectively.  VAT and VST both use these indexed files.  If the files are not index, the VAAST library will index them on the fly the first time they are encountered.  There is also an accessory script (vaast_indexer) available that will index these files, and it can be more efficient (especially in the case where you need to index multiple files) to use vaast_indexer.  See the section describing vaast_indexer below for more details.

### VAT Command Lines

To see the full VAT documentation run:

```
VAT --help
```

The most basic VAT command line would look like this:

```
VAT –f genes.gff3 –a sequence.fasta variants.gvf > variants.vat.gvf
```

VAT will process the genome in chunks to limit memory usage. This makes it more tractable to run VAT on smaller machines or to run many copies of VAT in parallel on larger machines. The default chunk size is 50 MB, but can be set the chunk size larger or smaller manage the trade off between speed and memory usage. You probably don't want to process with a chunk size of less than about 1 MB as this will start to incur a significant performance penalty with little improvement in memory usage. If you have plenty of memory just set the chunk size larger then your largest chromosome or contig and VAT will process all chromosome/contigs in one chunk. A more complete VAT command line would look like this:

```
  VAT –f genes.gff3 –a sequence.fasta –c 50000000 –g female –b hg19
variants.gvf > variants.vat.gvf
```

## VAT Output

The output from VAT is the same as the GVF input file with the addition of Variant_effect (an other) attributes in column 9. The attributes from an example Variant_effect attribute are shown below. The different attributes are broken up across multiple lines for clarity in printed form but in the GVF document they would appear on a single line separated by semicolons. The GVF specification will provide all the details of these lines, but briefly, multiple attribute values are comma separated, and data points within a single Variant_effect value are space separated and consist of:
   1. The variant effect, a Sequence Ontology (SO) term describing the effect that the variant has on some genomic feature.
   2. A 0-based index to the Variant_seq attribute values. This defines which variant sequence (allele) causes this effect.
   3. The affected variant type. This is also a SO term describing the type of variant that is affected.
   4. All remaining space separated values are a list of the feature IDs from the GFF3 file for the affected features.

In the example below the effects of an SNV are described. The second given variant sequence (allele) at this locus causes a non_synonymous_codon effect on an mRNA - the NM_001014980 mRNA in particular.

```
ID=SNV_01;
Variant_seq=A,G;
Reference_seq=A;
Variant_effect=non_synonymous_codon 1 mRNA NM_001014980,
                coding_sequence_variant 1 mRNA NM_001014980,
                amino_acid_substitution 1 polypeptide NP_001014980,
                gene_variant 1 gene FAM132A;
Variant_codon=TGT,CGT;
Variant_aa=C,R;
Reference_codon=TGT;
Reference_aa=C;
Genotype=0:1;
```

## VST - Variant Selection Tool

VST performs set operations (intersection, union, compliment, and difference) on a group of annotated GVF files to produce a merged representation in a CDR file. This is typically used to create a merged set of variants for the target or background genomes required as input to VAAST, but can also include more complex selections as described below. VST will process the genomes in chunks to manage memory usage (see description of VAT above and the VST documentation for more detail). By default the chunk size in 50,000,000 bases, but an alternate value can be specified on the command line. VST also needs to know information about the chromosome/contig sizes it will encounter in the GVF file (or the genome assembly), and the gender of each individual. VST will also infer gender like VAT if this info is not provided in the GVF file or on the command line, but because it does not use the Fasta file, information on chromosome sizes must either appear in the GVF files as ##sequence-region pragmas or VST must be given the genome assembly build on he command line. VST can perform multiple set operations in a single run and these will be applied recursively as explained below.

### VST Input

The input to VST is one or more variant files in GVF format. These should be files that have been annotated with VAT, and a '.vat.gvf' extension is recommended. Keep in mind that these GVF files will need to be indexed. VST will do this as soon as it encounters an un-indexed file, but that means that each of potentially many files will be indexed one at a time and this can add to VST run time. See the vaast_indexer section for an easy way to parallelize indexing.

### VST Command Line Options

To see the full VST documentation run:

```
VST --help
```

VAAST User's Guide - Version 1.0.1                                                13

The --ops (-o) argument provides VST with the set operations to perform.
The various types of set operations available and their one-letter abbreviations are:

1.  U - Union: All variants.
2.  I - Intersection: Variants shared by all files.
3.  C - Compliment: Variants unique to the first file.
4.  D - Difference: Variants unique to any file.
5.  S - Shared: Variants shared by n files:  S(n,0..2). The value for n can be a positive integer, in which case all variants present in at least n files will be retained.  In addition, the value of n can be an integer preceded by one of the following comparison operators:
    a.  eq - Exactly n files share the variant.
    b.  gt - Greater than n files share the variant.
    c.  lt - Less than n files share the variant.

A simple union on the first through fourth files on the command line would have an --ops argument like this:

```
--ops U(0, 1, 2, 3)
```

In the above example U refers to the set operation to be carried out on the files (union), and the integers 0, 1, 2, 3 are a 0-based index of the files given on the command line to be operated on.  The above operation would produce the union of the first, second, third and fourth files on the command line.  Since 0, 1, 2, 3 is a range of contiguous file index IDs the range could have been specified as a range instead to save some typing.  The same operation, specified with a range, would appear as:

```
--ops U(0 .. 3)
```

In both cases above the spaces are optional.

The most basic VST command line would look like this:

```
VST --ops 'U(0..3)' *.gvf > vst_output.cdr
```

The above command line would produce a CDR file with all variant loci present in any of the first four files given.  If there were more than four files represented by the glob above only the first four would be operated on.

Set operations can be combined recursively to produce more complex operations. For example, a complex VST command that would identify the variants found on the

X chromosome of an affected boy and his carrier mother and grandmother, but not on his unaffected brother and uncle would look like this:

```
VST -o 'C(I(0..2),U(3,4))' -c chrX -b hg19 -h 50000000 affected.gvf
mother.gvf grandmother.gvf brother.gvf uncle.gvf > vst_output.cdr
```

In the above example the intersection is found between the variants in genomes of the affected boy, his mother and his grandmother. The union is found between the variants found in the genomes of the unaffected brother and uncle. Finally, the results from those first two set operations are used as input to the final set operation which takes the compliment (only those variants found in the first set) providing as output all variants found in all three of the affected/carrier genomes but not found in either of the unaffected genomes.

## VST Output

The output from VST is a CDR file. The CDR file contains a condensed representation of the variants in one or more GVF files. The CDR file then becomes the input for the target and background genomes to VAAST. CDR files have two types of lines. At the beginning of the file are locus lines. These lines describe the variants at every locus in the set of genomes where any genome has a variant.

```
chr1 878681 878681 SNV synonymous_codon A|E 48-56,137,200|A:G|E:G
```

The locus lines have the following columns:

1. Seqid - The chromosome
2. Start
3. End
4. Variant type - (Currently always SNV for VAAST)
5. Variant effect(s) - Sequence Ontology terms describing the effect of the variant.
6. Reference sequence. The reference nucleotide is always present and if applicable the reference amino acid is also given separated by a vertical bar '|'. If multiple reference amino acids are possible due to multiple transcripts translated in different frames, then only the most common (random if a tie) amino acid is given.
7. Column 7 and onward show all the genotypes that were found in any individual in the given set - a separate column for each genotype seen. The data in each of these columns is separated by vertical bars '|' into the following values.
   a. A comma separated list of index numbers (or ranges, 5-8 represents 5,6,7,8) that represent the individuals that have the given genotype.

The numbers are a 0-based index to the files given on the command line.
    b. The variant genotype (e.g. A:C).
    c. If the given variant falls within a coding region the variant amino acids corresponding to the genotype are given in the same order (e.g. C:R).

At the end of the CDR file several lines of meta-data are given that apply to the entire file. Each of these lines begins with a double-hash '##'. The data following the double-hash are tab separated. The following meta-data lines are present:

1. ##W0C6.25701499933934e-05
    Represent amino acid substitution frequencies.
       a. The first column has two amino acids separated by a zero.
       b. The second column has the frequency at which that particular substitution occurs within the data set calculated as number of times that substitution is seen divided by the cumulative length of all genomes in the set.
2. ##    PSUEDO-COUNT    3.2303107427267e-11
    The pseudo count is used by VAAST as a proxy for the frequency of any amino acid substitution not seen in a particular set and is 10X less than the minimum amino acid substitution frequency seen within the dataset.
3. ##    GENOME-LENGTH   3095677412
    The length of the genome.
4. ##    GENOME-COUNT    5
    The number of genomes in the set.
5. ##    CUMULATIVE-GENOME-length      15478387060
    The cumulative length of all genomes in the set.
6. ##    COMMAND-LINE    /home/bmoore/VAAST_RC_1.0/bin/vaast_tools/vst -o U(0..4) file0.vat.gvf file1.vat.gvf file2.vat.gvf file3.vat.gvf file4.vat.gvf
    The command line used.
7. ##    PROGRAM-VERSION 1.0.1
    The program version
8. ##    GENDER F:0-2 M:3-4
    The gender of the individuals in the file (0-based index to the files on the command line).
9. ##    FILE-INDEX    0     file0.vat.gvf
    The 0-based index with the corresponding files given on the command line.

An example excerpt from a CDR file is shown below (some information is truncated with ... for display purposes.

```
…
chr1 879304 879304 SNV non_sy…_codon G|G 0,4-5|A:A|D:D  1|A:G|D:G
chr1 879317 879317 SNV synonymous_codon C|Y 2|C:T|Y:Y

…
##      A0E     4.7243294612378e-06
##      C0M     9.69093222818011e-10
##      R0Q     3.40591043470133e-05

…
##      PSUEDO-COUNT    3.2303107427267e-11
##      GENOME-LENGTH   3095677412
##      GENOME-COUNT    453
##      CUMULATIVE-GENOME-length        1402341867636
##      COMMAND-LINE  file0.vat.gvf file1.vat.gvf file2.vat.gvf …
##      PROGRAM-VERSION rc_1.0
##      GENDER  F:0-2 M:3-4
##      FILE-INDEX      0    file0.vat
##      FILE-INDEX      1    file1.vat
##      FILE-INDEX      2    file2.vat
##      FILE-INDEX      3    file3.vat
##      FILE-INDEX      4    file4.vat
##      FILE-INDEX      5    file5.vat
…
```

## VAAST – Variant Annotation, Analysis and Search Tool

VAAST, the Variant Annotation, Analysis and Search Tool is a probabilistic search tool for identifying damaged genes and their disease-causing variants in personal genome sequences. VAAST builds upon existing amino acid substitution (AAS) and aggregative approaches to variant prioritization, combining elements of both into a single unified likelihood-framework that allows users to identify damaged genes and deleterious variants with greater accuracy, and in an easy-to-use fashion. VAAST can score both coding and non-coding variants, evaluating the cumulative impact of both types of variants simultaneously. VAAST can identify rare variants causing rare genetic diseases, and it can also use both rare and common variants to identify genes responsible for common diseases.

### VAAST Input Files

VAAST takes three input files:

1. A feature file in GFF3 format containing the features (usually genes) to be evaluated.  This feature file should be the same used to annotate the background and target genomes.
2. A condenser file in CDR format that contains the set of variants found in the background genomes
3. A condenser file in CDR format that contains the set of variants from target genomes.

The GFF3 file format is described above and GFF3 files can be downloaded from the VAAST data repository. The CDR file format is produced by the VST program and is described above in the section on that program.

## Required VAAST Command Line Options

### mode and outfile

The simplest VAAST command line would look something like this:

```
VAAST –-mode lrt –-outfile output genes.gff3 background.cdr target.cdr
```

The output file will have a suffix of '.vaast', in the example shown above the file would be 'output.vaast'. Note that a second file with a '.simple' extension is also provided. This file provides a limited amount of the information available in the '.vaast' file, but is formatted with one feature on each line, so can provide a quick first look at the output from VAAST.

In this example, VAAST will use the likelihood ratio test (-mode lrt) to score features based on the frequency difference of variants between background and target genomes. The genes are ranked based on their p-value, which is calculated from the composite-likelihood ratio test (CLRT) by permuting the target and background status of each genome. The likelihood ratio test (lrt) mode is recommended in most cases, however an alternative method for ranking candidate genes - the weighted sum statistics (WSS) is available and discussed separately below.

### codon_bias and rate

Under the 'lrt' mode, you can provide additional information about the disease you are studying. Consider the following command line:

```
VAAST –-mode lrt –-rate 0.001 –-codon_bias –-gp 10000 –-outfile output
genes.gff3 background.cdr target.cdr
```

Here we introduce two important options:
- o  –-codon_bias (-k)
- o  –-rate (-r)

The –-codon_bias option causes VAAST to use amino acid substitution (AAS) frequencies to estimate the how deleterious each amino acid change is. By default AASs found in the OMIM database are used to represent the frequencies of AASs in a disease state. Other AASs can be utilized (see the aas_matrix option below). This option significantly increases the power of VAAST for a variety of common and rare disease analyses. The –-rate option takes as it's argument a real number that allows you to specify an estimate of the maximum expected disease allele frequency within

the background population. By setting this option, the CLRT will be constrained such that the allele frequency of healthy genomes in the alternative model cannot exceed the specified value.  Neither the –-codon_bias nor the –-rate options are required, however it is recommended that you always use the -–codon_bias option for disease gene discovery, and the –-rate option wherever a disease allele frequency estimate is available.

### gp and sp

Note that whenever we are using either the –-rate or the –-codon_bias options, the CLRT is un-nested, so we cannot convert the raw feature score to a p-value.  Instead, we perform permutations to determine the p-value.  In this example, we are using standard genome permutation (-gp <n>, n being the maximum number of permutations to perform).  This method permutes the background and target status of each genome, which controls for linkage disequilibrium (LD) between variants.  When analyzing many background and target genomes, genome permutation becomes computational intensive.  In this situation, site-permutation "-sp <number>" is an attractive alternative, although it does not correct for linkage disequilibrium.  The following command line demonstrates this option:

```
VAAST -–mode lrt -–rate 0.001 -–codon_bias -–sp 10000 -–outfile output
genes.gff3 background.cdr target.cdr
```

### Options for Masking and Grouping Variants

### variant_masking

It can be useful to mask out variants from genomic regions with repetitive or homologous sequences.  Variant calling may be error-prone because of difficulties in uniquely mapping reads. The -–variant_mask option is designed to address this problem.  The following command line shows this option:

```
VAAST -–mode lrt -–codon_bias -–gp 10000 -–variant_mask <masking_file>
outfile output genes.gff3 background.cdr target.cdr
```

Multiple variant-masking files are available from the VAAST data repository and you should choose the one that best describes your sequencing/variant-calling pipeline. Be aware, however, that variant masking may mask out genes of interest if they have paralogous copies or conserved motifs, so caution is advised.

### grouping

In some cases, the disease under investigation may be caused by many relatively rare but distinct variants within the same gene.  Each variant alone may have been rare enough to escape detection by the CRLT. By grouping these rare variants in a feature before raw score calculation, we can gain power to detect such disease genes.  The following command line demonstrate the --grouping option:

```
VAAST —-mode lrt —-codon_bias —-gp 10000 —-grouping <integer> --outfile
output genes.gff3 background.cdr target.cdr
```

The argument to the --grouping option is an integer that sets the upper limit of allele count for variants to be grouped.  For example, '-grouping 3' means that any variant alleles present 3 times or less within the target genomes will be grouped.  The scored variants are treated as if from a single locus and scored in aggregate.  By default the value for the --grouping option is set to four. To disable variant grouping set the value of grouping to zero (--grouping 0).

## Options for Disease Modeling

VAAST provides a number of options that allow you to inform the algorithm of additional information that you may have about a disease.  This information when supplied to VAAST can result in increased power to detect the causative gene.  The sections below describe these options.

### inheritance

The --inheritance (-iht) option allows you to inform VAAST about the inheritance model.  This option takes as its argument either d(ominant) or r(ecessive).  When dominant is specified only the best-scoring variant locus in each feature in each individual will be scored. When recessive is specified, only the best-scoring homozygous locus or the two best-scoring heterozygous loci in each feature in each individual will be scored.

### penetrance

The --penetrance (-pnt) option allows you to specify whether the causal alleles are expected to be fully penetrant or not.  Valid arguments to the --penetrance option are c(omplete) and i(ncomplete).  The default is '--penetrance i'. When '--penetrance c' is specified with a dominant disease model (--inheritance d), any target variant found in the background genomes will be removed from the calculation.  Complete penetrance under a dominant model is a very stringent filter that should be used with great caution.

If '--penetrance c' is specified under a recessive model, the behavior is a little more complicated:

1. Any target variant that is found in homozygous state in any of the background genomes will be removed from calculation.
2. Any combination of two heterozygous loci found within one feature in a background genome will not be scored. See the -–no_max_allele_count option below to disable this behavior.

### locus_heterogeneity

The --locus_heterogeneity (-lh) option indicates that VAAST should allow locus heterogeneity – a situation where the trait is caused by mutations in genes at different chromosomal positions in different individuals. Allowed values for arguments to this option are y(es) and n(o). The default is "--locus_heterogeneity y". When "--locus_heterogeneity n" is specified under a dominant model, VAAST will require every target genome to have at least one variant with a CLRT score greater than 0 (i.e. every individual must have at least one putative disease causing variant). When "--locus_heterogeneity n" is specified under a recessive model, VAAST will require every target genome to have at least two variants with a CLRT score greater than 0. If these criteria are not met, the feature will receive a score of zero. Note that this is a stringent filter that will cause a true disease gene to be missed if just one individual in the target is heterogeneous or incorrectly genotyped and thus it is turned off by default.

### no_max_allele_count

The --no_max_allele_count (-n) option affects the behavior of the --inheritance option. Under recessive model (--inheritance r), VAAST scores only the two best-scoring heterozygous loci or the single-best homozygous scoring locus in each target feature within each a target genome. Under a dominant model (--inheritance d), VAAST scores only the single best-scoring locus in each feature within a target genome. The --no_max_allele_count option disables these restrictions and allows all variants to be score while the remainder of the inheritance model behavior remains active. In most circumstances, the default value is recommended (i.e. you don't need to set –no_max_allele_count).

A good combination of the above four disease model options may dramatically increase the sensitivity and/or specificity of a VAAST analysis. Several examples of the use of these options in combination are given below:

```
VAAST -–mode lrt -–codon_bias -–gp 10000 -–inheritance r -–outfile
output genes.gff3 background.cdr target.cdr
```

Above we have specified "--inheritance r" to use a recessive disease model. This will allow only the top two best-scoring variants in each feature within each target genome to be scored. Large genes often harbor many of variants and this increases the probability that minor systematic differences between background and target

genomes are magnified (e.g. population differences, sequencing platform differences), resulting in an overestimation of the significance level of the gene. Use of the -–inheritance option will remove this type of bias.

```
VAAST —-mode lrt —-codon_bias —-gp 10000 —-inheritance r —-penetrance c
—-outfile output genes.gff3 background.cdr target.cdr
```

In the example above we have specified the "–-inheritance r –-penetrance c" options in combination. This will greatly improve the specificity when searching for rare, recessive disease genes.

```
VAAST —-mode lrt —-codon_bias —-gp 10000 —-inheritance d —-penetrance c
—-lh n —-outfile output genes.gff3 background.cdr target.cdr
```

Our final example in this set is the most stringent. We specify "–-inheritance d –-penetrance c". Here we are saying that this is a fully penetrant, dominant disease. This combination removes all variants found in the background genomes. Disallowing locus heterogeneity in addition ("--locus_heterogeneity n") adds even more stringency. Any feature that has a zero score in any target genome will receive an overall score of zero. Please see the warnings above regarding the use of --lh and --penetrance c, and note that for most analyses, we do not recommend the use of such of a highly stringent filter.

### trio

A powerful research design for disease gene discovery in the case of highly penetrant recessive diseases is to sequence affected children and unaffected parents (trio data). In these cases VAAST can take advantage of the relationship between family members to enhance the specificity with which it detects casual genes. This option is demonstrated in the example below.

```
VAAST —-mode lrt —-codon_bias —-gp 10000 —-trio parents.cdr —-outfile
output genes.gff3 background.cdr target.cdr
```

The --trio option removes variants that are found in the affected genomes but not found in his/her parents, which are likely sequencing errors. Of course *de novo* mutations in the affected genomes would be filtered out by this step so this would not be a good option to use if you suspect the causative mutation may be *de novo*. However since the rate of de novo mutations is low relative to sequencing error rates (about 70 de novo mutations versus over 10,000 errors per genome) this option when used carefully can greatly improve the specificity of finding the causative gene. When '--penetrance c' is specified, VAAST will include the parental genomes among the background genomes for the purposes of evaluating penetrance criteria (see above).

Providing VAAST with trio data is a powerful way to increase specificity. In our validation study of the Miller Syndrome dataset (Roach et al. http://www.ncbi.nlm.nih.gov/pubmed/20220176) , running VAAST with a single quartet of genomes (two affected siblings and their healthy parents) resulted in two candidate genes - the two known causal genes for the two diseases affecting the children.

## Options Affecting Likelihood Ratio Scoring

### all_variant

By default VAAST will only score non-synonymous coding variants. The --all_variant (-e) option causes VAAST to score synonymous coding variants and variants in non-coding regions.

### aas_matrix

The --aas_matrix (-b) options allows you to use alternative amino acid mutation rate matrix for scoring codon bias. The VAAST data repository provides several substitution matrices such as the BLOSUM matrices.

### ref

The --ref (-a) option allows VAAST to score loci where the minor allele in the background genomes is the same as the allele in the reference genome. By default VAAST will not score these loci.

## An Alternative Feature-Ranking Algorithm

In addition to the likelihood ratio test '--mode lrt' as a scoring mode, VAAST also incorporates the Weighted Sum Statistic (wss) method (*Madsen and Browning (2009) A groupwise association test for rare mutations using a weighted sum statistic*) for ranking features. The following command line shows its use:

```
VAAST —-mode wss —-gp 10000 —-outfile output genes.gff3 background.cdr
target.cdr
```

### u

In addition to the '--gp' and '-sp' options for performing permutation tests, you can use '-u' option instead. This option will cause VAAST to estimate the significance of features based on the distribution of test statistics. This method is faster than genome permutation with '--gp'. In some scenarios, calculating significance while running '--mode lrt' mode can take a long time, while using '--mode wss -u' in

combination would give you an approximate significance estimate much faster. Keep in mind however that, "--mode wss" is typically less powerful than "-mode lrt," and is not compatible with "--codon_bias" and "--rate" options or the disease modeling options.

### Options Affecting Memory and Run-Time Performance

#### mp1

The --mp1 (-p) option allows VAAST to utilize multiple CPUs.  When using --mp1 each thread will process a separate feature.  The argument is an integer that specifies the number of CPUs (threads) that are to be used.

#### mp2

The --mp2 (-q) option also allows VAAST to utilize multiple CPUs.  When using --mp2 VAAST processes one feature at a time, spreading the permutations across multiple threads.  The argument is an integer that specifies the number CPUs (threads) are to be used.

#### chrs

The --chrs (-c) limits VAAST to scoring only score features on a specified chromosome(s). The argument to this option is a comma-separated list of chromosomes.  In this case only the actual chromosome number or letter is given. For example '–-chrs 4,15' will score features on chr4 and chr15 only. Specifying chromosome and Y would be done with '--chrs X,Y' (case insensitive).

#### features

The --features option allows the user to specify a comma separated list of feature IDs.  Only the specified features will be scored.

#### restart

The --restart option will restart your VAAST analysis if your previous VAAST run terminated unexpectedly. For a restarted run you only need to specify '-restart' and '--outfile' on the command line, all other options from the previous run will be recovered from temporary log file(s).  The --outfile argument however must be the same as it was in the failed run.

#### fast_gp

The --fast_gp option exploits the degeneracy within genotypes between the background and target genomes to accelerate genome permutations. This option is highly recommended if you are doing genome permutations (-gp) and your target genome number is less than about 20.

*significance*

The --significance (-j) option causes VAAST to stop permutations when the 0.95 confidence interval is above or below a specified significance level.  This can improve performance by causing VAAST to stop permutations when the p-value is accurate enough but has not yet converged, thus avoiding unnecessary permutations.

*daemon*

The --daemon option causes VAAST to spawn a daemon process to monitor its RAM usage and kill VAAST if RAM exceeds 90% of the system RAM.

## VAAST Output

VAAST writes its outputs to a file with a '.vaast' extension.  This file contains a ranked list of features and other pertinent information that helps users interpret feature ranking and scores. Below is a snippet from a VAAST output file:

```
## VAAST_VERSION        RC1.0
## COMMAND VAAST -o test -m lrt -k -d 1e5 -c 16 features.gff3 \
   background.cdr target.cdr
> NM_021195
chr16    -       3005361;3006023;-;16
TU:    23.41   3005636@16      C|V     N|2|C:G|V:L     N|0|G:G|L:L
TU:    0       3005925@16      G|T     N|2|A:G|T:T
BR:    3005597@16      T|I     1,3,5,9-10,14-16,21,23,25,29,31,33, \
   36,38,41-42,44,47,50,53-55,59-61,67|C:C|V:V
RANK:0
SCORE:23.40553616
genome_permutation_p:7.90624199419662e-05
genome_permutation_0.95_ci:3.46e-05,0.0001549
```

Two lines at the top starting with '##' appear at the beginning of every VAAST output file.  These two lines show the version of VAAST and the command line that generated the file.

Information on each ranked feature appears in the lines below that.  The record for a single feature appears on multiple lines in the VAAST output.  Each feature record begins with a line that begins with the character '>' and continues until the next record begins or until no more records are encountered.  After the '>' character on the first line the ID of the feature is given.  In the example above the details for the NM_021195 mRNA are shown.

The second line after the ID line in the feature record describes the structure of the gene model.  Three or more tab-delimited columns occur on this line.  Those columns are:

1. The first column records the chromosome/contig name (the seqid in GFF3 and GVF files).
2. The second column designates the strand that the feature is on.
3. The third and subsequent columns each specify the structural annotation for an exon.  Four values are separated by semicolons:
    a. The start coordinate of the exon.
    b. The end coordinate of the exon.
    c. Strand of the exon.
    d. Chromosome number of the exon.

On the third line and several additional lines, information on individual variant loci located within the given feature is described. These lines are split into tab-delimited columns.  The description of these columns differs depending on the value in the first column. If the first column has a value of either 'TU' or 'TR', the data on that line describes variant locus details for the target genomes.  If the first column has a value of 'BR' or 'PR' then the line describes variants found in the background or parent (--trio) genomes respectively.

Target genome columns:
1. Either 'TU' or 'TR' where 'TU' indicates that the variant is unique to the target genomes and 'TR' indicates that the background genomes share this variant.
2. Likelihood ratio score at this locus. The higher the score, the more likely this locus is disease-causing. A score of 0 or below indicates that the feature is very unlikely to be disease causal. This column is present only in target genome loci and only meaningful for '--mode lrt' scoring.
3. The 1-based genomic coordinate of this locus and the Seqid separated by '@'. For example, '56289513@16' indicates position 56289513 on chromosome 16.
4. All subsequent columns provide details of individual genotypes observed at this locus within the target genomes.  The genotype CDR file provided representation of genotype at this locus. For loci in the target genomes you'll see either 'B|' or 'N|' at the beginning of genotype.  A 'B' indicates that this variant is present in the background and an 'N' indicates that it is not.

After all of the lines describing variant loci within the feature have printed, there are several additional lines in each record, some of which are optional:

**RANK:**  The 0-based rank of this feature relative to all other features in the GFF3 file.
**SCORE:**  The raw CLRT score of this feature, which is equal to 2 x ln(composite likelihood) – 2 x (number of variant sites). This score is only meaningful with the likelihood ratio test mode '--mode lrt'.
**genome_permutation_p:**  The significance level obtained by doing genome-permutations. This value is probably the most indicative figure of the disease relevance of the feature. However, keep in mind that the number of genomes in your target/background CDR constrains the lowest achievable significance level. If VAAST did not observe a single permutation with a CLRT score higher than the

actual CLRT score, it will report the lowest possible p-value bound by the number of target/background genomes.

**genome_permutation_0.95_ci**: The 95% confidence interval of the genome permutation p value.

**IS_permutation_p:** The significance level obtained by non-LD corrected site permutation.

**IS_0.95_confidence_interval:** The 95% confidence interval of the site permutation p-value.

**IS_converged:** This value indicates whether there were a sufficient number of permutations for the site permutation p-value to converge. It is a Boolean value 1 for 'yes' and 0 for 'no'.

**UPF:** This value is only present when using trio option (--trio). It indicates whether the target genome(s) variants in this feature can all be found in one of the healthy parents.

A second file is created by VAAST, which has a '.simple' extension. This contains a limited subset of the information in the main '.vaast' output file.

```
RANK    Gene    p-value    Score       Variants
1       DHODH   8.61e-07   52.95337954 chr16:70614936;10.86;C->T;R->W;0,1
2       MPV17L  1          0
```

The headers in the example above explain the columns. The final column may represent many variants and for each one the values given correspond to the ones given in the file above as follows:

1. chr16 - The chromsome/conting on which the feature is found.
2. 70614936 - The start location of the variant on the given chromosome/contig
3. 10.86 - The score of this variant
4. C->T - The reference and variant allele
5. R->W - The reference and variant amino acid
6. 0,1 - The 0-based index list describing which individuals are affected by this allele. See the description in the VAAST output above for more details.

## Accessory Tools

### vaast_indexer

The vaast_indexer script will index GFF3, GVF and FASTA file for use with the tools in the VAAST software package. If the file(s) have a .fa, .fasta, .gff, .gff3, or .gvf extension then vaast_indexer will know which type of file it is indexing and do the right thing. If your files have other extensions, use the --format option to specify either fasta or gff.

```
vaast_indexer file1.fasta [file2.gff3 file3.gff file4.gvf ...]
```

The above command will index each of the files one at a time.  Try this simple method of parallelizing vaast_indexer (and other commands on the linux command line).

```
ls *.vat.gvf | nohup xargs -I {} -n 1 -P 20 vaast_indexer {} &
```

This will run vaast_indexer on all '.vat.gvf' files in the current directory using 20 CPUs simultaneously.  See the xargs man page for more details on this great tool.

### vaast_converter

The vaast_converter will convert VCF file to GVF for use with the tools in the VAAST software suite.

```
vaast_converter --build hg19 --path ./gvf file1.vcf file2.vcf
```

### vaast_validator

The vaast_validator will validate GFF3, GVF and FASTA file for use with the tools in the VAAST software suite.

```
NOT YET IMPLIMENTED
```

### cdr_manipulator.pl

The cdr_maipulator script allows you to merge, split and liftOver CDR files

```
cdr_manipulator.pl merge first.cdr second.cdr output.cdr
cdr_manipulator.pl split original.cdr output.cdr 0,1,2,3
cdr_manipulator.pl bootstrap original.cdr output.cdr 0,1,2,3
cdr_manipulator.pl liftover liftover_chain_file original.cdr  output.cdr
```

### quality-check.pl quality-check-maf.pl quality-check2.pl

This script will check if there are ethnicity or platform mismatches between two sets of CDR files (target and background for example) by determining if there is an over-representation of rare SNVs in the target compared to the control.

```
quality-check.pl -sim 1000 background.cdr target.cdr
```

### simple_output.pl

Make a simple output file from a full VAAST '.vaast' output file. This file contains a limited subset of the information in the main '.vaast' output file.  See the section above on VAAST Output for more details.

```
simple_output.pl file.vaast > file.simple
```

### transform2new-cdr.pl

Convert old VAAST CDR files to the current CDR format.

```
transform2new-cdr.pl file.gff3 old.cdr new.cdr
```

### transform2new-cdr-nonsyn.pl

Convert old VAAST CDR files that only contained non-synonymous SNVs to the current CDR format.

```
transform2new-cdr.pl old.cdr new.cdr
```

### vaast_sort_gff

Sort a GFF and/or GVF file by location.

```
vaast_sort_gff -in_place file1.gff file2.gvf
```

### vaast_imager

This script will render VAAST data into a chromosome ideograms with genes represented as vertical bars scaled and colored relative to their VAAST score.  The graphics file generated is in PNG format.

```
vaast_imager --seqid chr16 --begin 1 --end 1000000  --cyto cytoBandIdeo.txt --image_file image.png
--map_file map.txt --color_scheme permutation_p  output.vaast
```

Note that this tool requires the GD Perl library, which in turn requires the libgd2 library which also has dependencies of it's own.  Below is a brief description of the dependencies and some guidance on installing them.   A full description of the install process is beyond the scope of this guide.

*Non-Perl prerequisites:*

1. libgd,  http://www.libgd.org/Main_Page - The GD graphics library.
2. libpng, available from http://www.libpng.org/pub/png/ - Portable Network Graphics library; requires zlib.
3. zlib, available from http://www.gzip.org/zlib/ - Data compression library.

Installing libgd can be easy or a challenge depending on your platform.  Things like yum, apt and fink are your friends, but you may have to resort to installing some or all of the libraries from source.

Below are some system specific instructions for installing compiled prerequisites.  While we have made an effort to provide the correct package names, these OS distributions are constantly being updated, so you may have to search for the correct package name.

Red Hat/Fedora/CentOS

```
# Search with
#yum list | grep zlib | less
yum install zlib
yum install libpng
yum install gd
```

Debian/Ubuntu

```
# Search with
 # apt-cache search zlib | less
 apt-get install zlib1g
 apt-get install libpng3
 apt-get install libfreetype6
 apt-get install libjpeg62
 apt-get install libgd2-xpm-dev
```

MacOS X:

```
zlib is already included
```

```
fink install libpng3
```

CPAN Perquisites:

```
cpan GD
```

## Putting it all Together – Worked Examples

Let's do a complete (albeit small scale) VAAST run with the test files in the 'examples/data/' folder in the VAAST code directory. This example uses the data from Table 3 in the VAAST paper, where we benchmarked the power of VAAST to identify the causal gene in a rare disease (Miller Syndrome). The causative alleles reported in Ng *et al., 2010* were added into the GVF files of three healthy genomes (here we only use chromosome 16 data which contains the causative gene for Miller Syndrome - *DHODH*). We will run the whole VAAST pipeline and identify the causative gene. All the following command lines are executed in 'examples/' directory in the VAAST code repository.

The first step is to annotate the three GVF files with VAT. Each GVF file represents the genotype data for chromosome 16 of one affected genome.

```
VAT -f data/hg18-chr16.gff3 -a data/hg18-chr16.fasta data/miller-1.gvf  > data/miller-1.vat.gvf
VAT -f data/hg18-chr16.gff3 -a data/hg18-chr16.fasta data/miller-2.gvf  > data/miller-2.vat.gvf
VAT -f data/hg18-chr16.gff3 -a data/hg18-chr16.fasta data/miller-3.gvf  > data/miller-3.vat.gvf
```

Next we merge these three annotated GVF file into a single CDR file:

```
VST  -o 'U(0..2)' -b hg18 data/miller-1.vat.gvf data/miller-2.vat.gvf data/miller-3.vat.gvf >
data/miller_output.cdr
```

Finally, we will run VAAST on the target CDR file against a background CDR file containing 180 genomes from 1000 genome project and 9 genomes from 10Gen genome data set (Reese *et al.* (2010) A standard variation file format for human genome sequences):

```
VAAST -inheritance r --locus_heterogeneity n --fast_gp -gp 1e4 --outfile test --rate 0.00035 --mode
lrt --codon_bias data/hg18-chr16.gff3 data/189genomes-chr16.cdr data/miller_output.cdr
```

Here VAAST searches for genes causal for Miller Syndrome on chromosome 16, assuming a recessive inheritance model (--inheritance r) and no locus heterogeneity (--locus_heterogeneity n). We also used the amino acid substitution method (--codon_bias) and provided an estimate of the maximum frequency of the causal allele (-rate 0.00035).

After the analysis has finished you will find the VAAST report file ('test.vaast') in the same folder. The top-ranking gene is DHODH with a p-value of 8.61e-7, which has been reported to be the causal gene for Miller Syndrome. In this example, DHODH is ranked 1st , being the only statistically significant feature.  Keep in mind that in some cases, false-positives may receive genome-wide significance and may even out-rank the true causal gene(s) due to poor variant calling, population stratification or platform bias issues.

For a longer,  more in-depth tutorial, see the VAAST_Quick-Start-Guide.pdf file in the docs folder.


## Best Practices

When you run VAAST, one thing to keep in mind is that a carefully designed research strategy is vital to the success. VAAST could be sensitive to the systematic bias between the background and target genomes. For example, if the background and target genomes have distinct or disproportional makeup of ethnicities, VAAST may pick up genes showing different allele frequencies between different populations instead of disease-relevant genes. The same is true for platform bias, and therefore background and target genotypes should be generated from the same sequencing platform and variant calling pipelines whenever possible. If not, it is possible that you may see hundreds or thousands of features showing genome-wide significance because of the platform mismatch.  In addition to the introduction false positives, target and background mismatches can drastically increase computational run time.

As stated before, the most meaningful statistic that VAAST generates is the genome permutation p-value, which is corrected for LD and reflects the significance level of the feature. The genome-wide significance level threshold that you should use can be calculated from number of features in your GFF3 file. For example, if your feature file contains 21,000 protein-coding genes and you set the significance level to be 0.05, then the genome-wide significance level would be 0.05 / 21000 = 2.38e-6. It is desirable to check the number of features achieving genome-significance before looking at individual features.  If many features are statistically significant, it is very likely that systematic differences between background and target genomes are biasing the results.

VAAST 1.0.1 and later have two scripts, check_quality.pl and check_quality-maf.pl which can help you asses whether there is a significant mismatch between the rare variant frequencies between your target and background genomes.

When you get a ranked list of genome-wide significant genes and start looking at the genotypes for each individual feature, the variant score (the second column in the variant lines) is very helpful, as it tells you about the relative contribution of each locus toward the overall significance of this feature. For high-scoring variants, you may want to check their presence/frequency in background genomes or parent genomes (if trio data is available).

Occasionally, and especially with unmatched background and target genomes (e.g. different populations or sequencing platforms), some genes may receive a high rank simply because they have many variant sites. With more variant sites, the CLRT is more likely to identify small systematic differences between background and target genomes.

When used with the default settings for locus heterogeneity (yes) and penetrance (incomplete), the recessive inheritance option (-iht r) is appropriate for a wide range of study designs. This option allows up to two variants per genome per feature to receive a score. Therefore, as long as no individual carries more than two disease-causing variants in the same gene, this option will not result in false-negatives, even if the disease-gene is dominant. The recessive option can partially mitigate the problems introduced by unmatched background and target genomes by scoring only the alleles most likely to be disease causing in an individual.

Because the significance level of each feature is estimated by permutation, the amount of time required to accurately estimate the significance level increases exponentially as the significance level decreases. In most situations, obtaining a highly accurate estimate of the significance level is less important than knowing whether the significance level is above or below some specified threshold. For these situations, the --significance option can greatly reduce the computational requirements of the analysis. With these considerations in mind, when reporting VAAST results, the 95% upper bound on the p-value should be reported in addition to or instead of the estimated p-value. Using the –mp2 option with the –feature option on a small list of genes of interest is one option for obtaining accurate p-values with tight confidence intervals for the purposes of publication.

# Publications and Citing

*To cite VAAST in publication please cite:*

A probabilistic disease-gene finder for personal genomes.  Yandell M, Huff CD, Hu H Singleton M, Moore B, Xing J, Jorde L, Reese MG
Genome Res. 2011. Sep;21(9):1529-42.

*Other publications citing VAAST:*

Using VAAST to Identify an X-Linked Disorder Resulting in Lethality in Male Infants Due to N-Terminal Acetyltransferase Deficiency.  Rope AF et al.  Am J Hum Genet. 2011 Jul 15;89(1):28-43

*Other publications related to VAAST:*

GVF

A standard variation file format for human genome sequences. Reese MG, Moore B, Batchelor C, Salas F, Cunningham F, Marth GT, Stein L, Flicek P, Yandell M, Eilbeck K. Genome Biol. 2010;11(8):R88.

Miller syndrome

Exome sequencing identifies the cause of a mendelian disorder.
Ng SB, Buckingham KJ, Lee C, Bigham AW, Tabor HK, Dent KM, Huff CD, Shannon PT, Jabs EW, Nickerson DA, Shendure J, Bamshad MJ. Nat Genet. 2010 Jan;42(1):30-5.

# Appendices

## FAQ

### *Why do I get every variant scored as 0 with a permutation p-value of 1?*
This can be caused by a problem with the CDR files. Check to be sure that a previous VST run didn't fail and that both CDR files look good and have the expected number of variants.

### *Why do I get hundreds or thousands of highly significant genes?*
This can be caused if there is a significant variant mismatch between the target and background files. For example, if the target genomes and background genomes have differences in their sequencing and/or variant calling pipelines then VAAST may be detecting the platform specific biases between your target and background data sets rather than disease loci. Variant masking can help with this, but it does run the risk of masking real signal. See the section on variant masking, use the check_quality.pl script included with VAAST and do everything you can to match the target and background genomes for sequencing, variant calling and ethnicity.

### *I'm getting the following error message...what does it mean?*
Please see the following section "Error Codes" for a description of all the error codes, warnings and info statements returned by code in the VAAST package. VAAST returns messages in the following format:

Level : Code : Message

Where "Level" is one of INFO, WARN or FATAL; "Code" is a simple description of the general class of error; and "Message" generally has some information about that particular instance of error.

### *Why do I get any of the following errors?*
- "Use of uninitialized value"
- "Can't use an undefined value as an ARRAY reference"
- "Argument "" isn't numeric in array slice

These are errors coming from Perl itself that we haven't caught and handled in our code. In general these tend to occur early in the pipeline (vaast_converter, vaast_indexer or VAT) and are the result of variation in incoming data formats that we haven't anticipated in our code.

### I found a bug in VAAST.  How do I report it?

If you get a message that is not listed in the section on Error Codes above or you think you have found a bug in VAAST code, please contact the vaast-dev@yandell-lab.org mailing list to report it.  Bug fixing is always easiest when the bug report is accompanied by a small amount of data and a command line that will recreate the problem.  Please include such a test case whenever possible with bug reports.

### What/Where is the VAAST Report Viewer?

Some of the images in the VAAST paper were generated by code that was referred to in the "VAAST Report Viewer".  As of VAAST 1.0.1 we have included the script that generated these images - vaast_imager found in the bin/vaast_tools directory.

### How do I convert VCF to GVF?

Use the script vaast_converter in the bin/vaast_tools directory.

### What is the source of the refGene_hg??.gff3 files?

The VAAST development team generated this file.  There is a README file that is distributed with the data file that gives details on how it was created: http://www.yandell-lab.org/software/VAAST/data/hg19/Features/README

### What is a valid genome build?

The VAT recognizes the following names for hg18 and hg19 human genome builds (and will standardize them for downstream code) either on the command line or as a ##genome-build pragma in GFF3 and GVF files:

- hg18
- UCSC hg18
- NCBI B36
- NCBI build 36
- B36

- hg19
- UCSC hg19
- NCBI B37
- NCBI build 37
- B37
- GRCh37

### Can VAAST be used with non-human variant data?

We are using VAAST within our group on non-human variant data, so we have begun to resolve some of the issues associated with this, however difficulties remain.  One of the biggest challenges today is that large background datasets are hard to come by in many non-human organisms.  If you encounter problems using VAAST with non-human organisms please contact us on the vaast-user@yandell-lab.org mailing list.

### *Do you have other background files?*

We do have other background files that we use within our group. We have not made all of them public because some of them come with significant caveats related to the underlying quality of the original sequence data or they may introduce significant platform biases. In general we will always release background files publicly as soon as we feel that they are not causing more trouble than they are worth. As of this writing (Dec. 2011) we are preparing a set of background files from the Oct. 2011 release of the 1000 Genomes Phase I variant call release and the Complete Genomics Diversity Panel.

### *Can I use VAAST with Illumina, 454, SOLiD or Complete Genomics data?*

Yes. You can run into issues platform specific biases when combing datasets from different sequencing platforms. Variant masking (described above), which removes variants from non-unique portions of the genome can go a long way to resolving these issues, but does have the potential to remove real signals as well. It is always best whenever possible to match the sequencing platform for your target and background genomes.

### *Can VAAST analyze indel or structural variant data?*

As of the VAAST 1.0.1 release only SNV data is supported. We have developed a version of VAAST that will support insertions and deletions of any size as well as splice variants, and this should be available in the first quarter of 2012.

### *Can VAAST use SNP microarray data?*

Yes, SNVs from any source can be included in the VAAST pipeline. An addition to VAAST called pVAAST has been developed which can include linkage analysis and SGS analysis within the VAAST analysis. When this is available, adding microarray data to exome data has the potential for a significant improvement in signal. pVAAST should be available in the first quarter of 2012.

### *How can I get an academic license for VAAST?*

Visit the VAAST home page:
http://www.yandell-lab.org/software/vaast.html

### *How can I get a license to use VAAST for commercial, clinical or diagnostic* purposes?

Contact Martin Reese of Omicia Inc. at mreese@omicia.com

### *Can I use VAAST on my Desktop, Laptop, iPhone?*

Yes, yes, and no - sorry, "There's NOT an App for that". While VAAST will run fine on you personal computer the compute cycles needed to run larger analyses (lots of samples with lots of permutations) make a server class computer a better choice.

### Will VAAST run on MS Windows?

We don't know, since we haven't tried it.  If you have success running VAAST on MS Windows, please share your experiences on the vaast-user@yandell-lab.org mailing list.

### Does VAAST run on MacOS X or Linux?

Yes.

### Does VAAST support multi-threading or MPI?

The VAAST script itself uses threading to use multiple CPUs for permutation.  The options available for using this features are described above.  The VAAST script also has some support for using multiple nodes on an MPI compatible cluster although we haven't tested this feature enough to support it yet.  There has been some discussion about this on the vaast-user@yandell-lab.org mailing list and you are welcome to continue the discussion there.  The other scripts in the VAAST package do not have threading or MPI support yet.

### How many CPUs do I need?

You only need one, however if you are doing heavy permutations with large background files, the more the better.  The servers that we use for a lot of our analyses have 4x12 core CPUs.

### How much RAM do I need?

For most analyses you should be fine with 4 GB of RAM or less.  Analyzing all variants (instead of just coding variants) may double that amount.

### Why do I have multiple results for the same gene in the VAAST output?

The features actually being scored are transcripts, so there may be multiple transcripts scored for each gene.

### What is GVF?

GVF is a file format for describing sequence alterations and the effects of those alterations on other sequence features.  It is the format used by VAAST for describing SNV (and ultimately other) sequence alterations. For more information see the Sequence Ontology GVF page at: http://www.sequenceontology.org/resources/gvf.html

### What is GFF3?

GFF3 is a file format for describing sequence features that can be located on a genome.  It is most often used to describe gene models and other associated sequence features, and this is the type of GFF3 file that VAAST uses.  For more information see the Sequence Ontology GFF3 page at: http://www.sequenceontology.org/resources/gff3.html

### Why won't VAAST analyze chrM?

VAAST will analyze chrM variants if the target, background and GFF3 feature files contain them. At this time many of the larger background files available don't have chrM variants.

### Can I use hg18 and hg19 variants together in the same analysis?

No, but you can use the cdr_manipulator.pl script included with VAAST to move a CDR file from one assembly to another. You can also use the UCSC liftOver tool (which is what cdr_manipulator.pl is using under the hood) to move GVF and GFF3 files from one assembly to another.

### My variant files have chromosome names without 'chr' - can I use these?

The seqid (the name of the chromosome) has to match exactly between all the GVF and GFF3 files. The data supplied with the VAAST package come from the UCSC Genome group and thus all of these files have a 'chr' added to chromosome names. If your data doesn't have these, you will either need to add it to your data or remove it from the data supplied with VAAST.

### Where can I find the VAAST Background, Feature, Fasta, Variant Masking data files?

http://www.yandell-lab.org/software/VAAST/data

### What is the minimum p-value that I can obtain for a given set of background and target genomes?

The minimum p-value that can be obtained can be calculated by:

1/(n choose k) where n is the total number of target and background genomes and k is the number of target genomes. The following Wikipedia articles describe more:
http://en.wikipedia.org/wiki/Combination
http://en.wikipedia.org/wiki/Binomial_coefficient

Online calculators are available:
http://www.math.sc.edu/cgi-bin/sumcgi/calculator.pl

### What p-value do I need to obtain for a gene to have a genome-wide significance level of 0.05?

Since we are testing many (usually many thousands) of genes all at the same time we encounter a multiple testing problem (http://en.wikipedia.org/wiki/Multiple_comparisons) where it is more likely simply by chance that some gene(s) will appear to differ significantly between the groups simply because we have done so many tests of significance on a single dataset leading to false positives. One common correction for multiple testing is the Bonferroni correction (http://en.wikipedia.org/wiki/Bonferroni_correction) because it is easy to calculate and fairly good at removing false positives (however with some risk of increasing false negatives). To do a Bonferroni correction on the p-value from a VAAST analyses simply multiply the permutation p-value for a given

feature by the total number of features tested.  VAAST tests for genes actually operate on transcripts.  In the refGene_hg19.gff3 feature file there are 31,990 mRNA, however since most alternate splice transcripts overlap within the gene overlap the same region (and thus retest the same region), it is valid in most cases to take the number of genes in the file as the number of tests performed. In the refGene_hg19.gff3 feature file there are 23,033 genes and thus as a rough guide a Bonferroni correction for a genome-wide coding variant analysis in VAAST using the refGene_hg19.gff3 as the feature file could be calculated as follows:

permutation p-value * 23,033

Working  backwards, to gain a genome-wide significance of 0.05 for a given gene would require an uncorrected p-value of:

$0.05 / 23,033 = 2.2 \times 10^{-6}$.

### *When should I use variant masking?*
-

### *What variant masking file should I use?*
-


## Error Codes

### Fatal Errors

### *can_call_gender_only_on_gvf_file* (VAAST::GFF3)

Some code tried to call the 'gender' method on a VAAST::GFF3 object that was created on a GFF3 file, and this doesn't make sense.  The 'gender' method can only be called on a GVF file.

### *conflicting_sequence_lengths* (VST)

Different GVF/GFF3 files have specified different sequence lengths with ##sequence-region pragma.  Since these pragmas are supposed to represent the same assembly this isn't possible.  Maybe you have one file from one assembly and one from another.

### *eval_error_caught* (VST)

VST uses the eval function to carry out the recursive set operations and an error occurred during one of these calls.  This is most likely caused by a typo in your --ops command line argument.

*file_does_not_exist* (vaast_indexer)

A file path was given to the script but the file does not exist. Please check the file path.

*incompatible_genome_builds* (VAT)

A different genome build was declared in the GVF file and GFF3 file.  You need to be sure you are using the same assembly build for both of these files (and for all work within a VAAST analysis).

*invalid_comparison_operator* (VST)

The VST -S option allows you to specify the comparison operator.  Allow values are =, <, > and you gave something else.

*invalid_file_index* (VST)

VST uses 0-based numerical indexes to reference the files given on the command line.  You've passed something other than a valid integer within the --ops option to VST.  Please check your command line.

*invalid_file_type* (vaast_indexer)

The script relies on the file extension to determine what kind of file it is indexing. Use one of the following as a file  extension or use the --format option.

*invalid_first_argument_to_S* (VST)

The first argument to the VST -S option should be of the form <OPR><INT>, where OPR is one of =, <, > and INT is an 0-based integer indexing one of the GVF files given on the VST command line.  You apparently gave it something else.  Please check your command line.

*invalid_gender* (VAT)

Valid arguments to the --gender option are female or male (case insensitive) and you gave something different.

*invalid_genome_build* (VAT, VST)

You provided a build argument to VST that it doesn't support. Currently only hg18 and hg19 are valid arguments to build.

*invalid_gff-version_pragma* (VAAST::GFF3)

You file has a gff-version pragma at the top that has some value other than 3. Only GFF3 files are supported by VAAST::GFF3.

*invalid_pragma_value* (VAAST::GFF3)

VAAST::GFF3 found a missing or invalid pragma value while parsing a GFF3/GVF file. More details about which pragma and what the invalid value was are given with the error.

*invalid_sequence_length* (VST)

While trying determining the length of chromosome/contigs for you file VST calculated a zero-length sequence, which is invalid.

*missing_chromosome_data* (VST)

VST needs to know the length of the chromosomes/contigs it will be processing. It gets this information from either the GVF file ##sequence-region pragmas, from the --build argument, or from the --chrs argument if you are processing a subset of the genome. It found none of these sources of information. The easiest fix for this is to give the --build option on the command line with either hg18 or hg19 as the argument.

*missing_file_argument* (VAT)

One of the required files: GFF3, Fasta or GVF was not provided to VAT.

*missing_log_file* (VAAST::Utils)

When VAAST is run with the --restart option it expects to find an outputfile.tmpftp file in the current directory from a previous run. This file was not present and so a restart is not possible.

*missing_NUM_GENOMES_line* (VAAST)

VAAST needs to know the number of genomes used to create the CDR file for the target and background genomes. This information appears near the end of the CDR file with a ## NUM_GENOMES pragma, and appears to be missing. Check the completeness of you CDR files to be sure they aren't truncated.

*missing_or_unreadable_file* (VAAST,VAAST::Utils)

A given file was either missing or unreadable.  Please check you file paths and if the files do exist check their permissions.

*missing_sequence-region_pragma* (VST)

VST needs to know the length of the chromosomes/contigs it will be processing.  It gets this information from either the GVF file ##sequence-region pragmas, from the --build argument, or from the --chrs argument if you are processing a subset of the genome.  It found ##sequence-region pragmas for a given chromosome/contig, but the start and end values were missing. You should fix your GVF file sequence-region pragmas, however the easiest fix for this is to give the --build option on the command line with either hg18 or hg19 as the argument.

*options_error* (VAAST::Utils)

VAAST::Utils parses the command line options for VAAST, and it encountered an invalid option or argument.  More detailed information should have been given with the error message about which option/argument were invalid.

*unknown_data_type* (VAAST::Utils)

VAAST::Utils encountered an unknown data type.  If the error got this far it's probably a bug in VAAST and it would be great if you could report it to the VAAST users mailing list and the VAAST bug tracker.

*unreadable_file* (vaast_indexer)

A file was given to the script which and the file exists, but is unreadable.  Check the file permissions.

*untested_method* (VAAST::GFF3)

You've called a method on VAAST::GFF3 that has been written but not tested.  This suggests that you are writing you own code against the VAAST library (or someone else is doing that for you) and that's great, but you if that's the case you can probably look at the code yourself and find a workaround.  If not you may want to get on the VAAST developers list and discuss this issue.

*Value out of range* (VAAST::GFF3)

This error is coming from the Perl module Set::IntSpan::Fast that is included with the VAAST distribution.  This error can happen if (for example) you are using VST to operate on a very large GVF file (we've seen it at file sizes above 6 Gb for a single

GVF file).  The version of Set::IntSpan::Fast included with VAAST is written in pure Perl.  There is a complied version of Set::IntSpan::Fast::XS that will solve this error you can install the XS version of this module on your system with the following command.

```
sudo cpan Set::IntSpan::Fast::XS
```

**Warnings**

*adding_reference_seq_data* (VAT)

VAT encountered a line in a GVF file that didn't have a Reference_seq attribute, so it added the correct sequence from the reference fasta file.

*auto_setting_gender* (VAAST::GFF3)

The 'gender' method was called on a VAAST::GFF3 but gender information was missing from the GVF ('file ##individual-id Gender=female'), so the gender of the individual in the file was auto-set based on the ratio heterozygous/(heterozygous+homozygous) for all variants on chrX.  If that ratio is >= 0.45 the gender is set to female, otherwise it is set to male.  Using the presence or absence of variants on chrY is not always valid.

*cant_determine_gender_from_file_assuming_female* (VAT, VST)

VAT tried several ways to determine the gender (--gender option (VAT only), ##individual-id pragma in the GVF file, and auto-setting gender based on the ratio of heterozygous calls on chrX by VAAST::GFF3) but none of them worked.  VAT will annotate the file but will assume the genome is female for purposes of heterozygous calls on chrX.  Note that this will not prevent VAT from annotating chrY variants. Using an invalid gender for a genome will prevent inheritance models in VAAST from working and will cause invalid scoring on sex chromosomes.

*excessive_memory_usage* (VAAST)

When you pass the -daemon command to VAAST, it spawns a daemon process to monitor memory.  This process has detected that memory usage has exceeded 90% of system RAM and will terminate the parent VAAST process.  This is issued as a warning even though VAAST is about to die because the daemon needs to handle process termination a bit differently than the parent process.

*exon_boundary_beyond_transcript_seq_length* (VAT)

An exon boundary was detected in a GFF3 file that exceeded the bound of the parent transcript. This in invalid and likely represents a broken gene model and bad GFF3 file.

*heterozygous_locus_on_haploid_chromosome* (VAT)

A heterozygous variant was found on a chromosome region that should only have homozygous variant calls. Gender and pesudoautosomal regions are taken into account. You may want to make sure gender is set correctly or filter you GVF file for heterozygous calls on male sex chromosomes. Invalid variant calls like this will prevent inheritance models in VAAST from working and will cause invalid scoring on sex chromosomes, so VAT will fix them for you.

*ignoring_more_than_two_alleles_on_diploid_chromosome* (VAT)

VAT found more than two sequences given in the Variant_seq attribute for a variant in a GVF file. Only the first two sequences are used and all others are discarded.

*invalid_codon* (VAT)

VAT encountered a codon that it could not translate despite considering ambiguity codes. This may result from a codon with length less than 3 or because a nucleotide was specified with an invalid code (IUPAC ambiguity codes are valid) or because the ambiguity was too great to resolve the codon to a single amino acid.

*invalid_*pragma* (VAAST::GFF3)

VAAST::GFF3 issues a series of warnings about invalid pragmas while parsing GFF3 and GVF files. Each warning should provide some additional information that will give a clue to what the invalid pragma and value were.

*invalid_pragma_argument* (VAAST::GFF3)

An invalid argument was passed to one of the methods 'pragma_sequence_region' or 'pragma_individual_id'. More detail should be provided with the error message.

*invalid_pragma_value* (VAAST::GFF3)

An invalid value has been passed as an argument to one of the pragma get/set methods. More detail should be provided with the error message.

*mising_gender_in_CDR* (VAAST::Utils)

A CDR file is missing the required ## GENDER directive. Variant scores on sex chromosomes will be invalid.

*missing_or_invalid_variant_seq_data* (VAT)

VAT encountered a line in a GVF file that doesn't have a Variant_seq attribute.  This variant will be skipped.

*missing_or_invalid_reference_seq_data* (VAT)

VAT encountered a line in a GVF file that doesn't have a Reference_seq attribute.  The correct Reference_seq value will be added from the reference fasta file.

*multiple_non_ref_alleles_on_haploid_chromosome* (VAT)

Multiple sequences were found in the Variant_seq tag on a non-psuedoautosomal region of a haploid chromosome (even after the reference sequence was removed).  Only the first sequence will be considered.

*no_gff3_features_in_range* (VAT)

No features were found in the GFF3 file on the given chromosome/contig.  This will occur if you have variants for a given chromosome/contig in you GVF file and none for that same seqid in your GFF3 file.

*no_information_on_pseudoautosomal_regions_for_build* (VAT)

If you use a build other than hg18 or hg19, then VAT has no information on pseudoautosomal regions and will treat all of chrX and chrY as haploid in males.

*parent_feature_missing* (VAT)

An mRNA, exon, CDS or other feature was given, but the feature that it identified in its Parent attribute is not found in the file. This gene model will be discarded.

*replacing_existing_pragma_value* (VAAST::GFF3)

A pragma value is being set when a value for that pragma already existed - either because it occurred in the file, or has been set before by the program.  This is fine as long as that is what you intended to do.

*translating_codon_of_invalid_length* (VAT)

A codon of length less that 3 nts was encountered.  It may still be able to be translated due to the ambiguity of the genetic code.

*translating_sequence_of_invalid_length* (VAT)

A transcript sequence is being translated that has a length that is not divisible by three. This is almost certainly the result of a bad gene model in the GFF3 file. This error can generally be ignored, but be aware that any data for the genes involved will likely be bad.

*unable_to_guess_gender* (VAAST::GFF3)

When the 'gender' method is called on a VAAST::GFF object and the file did not specify the gender in an individual-id pragma, VAAST::GFF3 will attempt to determine the gender based on the ratio of heterozygous/(heterozygous + homozygous) variant calls on chrX. If that ratio is >= 0.45 the gender will be set to female otherwise it will be set to male. If chrX variants are not available then this warning will be thrown and an undefined value will be returned to the program calling the method.

*unable_to_determine_if_region_is_pseudoautosomal* (VAT)

The method overlaps_psuedoautosomal_regions was called, but one of the required arguments for seqid, start or end was undefined and thus VAT cannot determine if the region is within a pseudoautosomal region.

**Info Messages**

*processing_finished* (VAAST)

VAAST is letting you know she's all done!

*process_update* (VAAST)

VAAST is giving some update on the progress of the process.

*genome_build* (VAT)

VAT is letting you know what genome build it's working with.

## Data Internal to the Code

VAAST uses some information internally that is hard-coded and not supplied by the user.  The following section describes this data.

### Chromosome names and lengths

VAAST uses the following information on chromosome names and lengths for human genome builds.

*hg18*
- chr1247249719
- chr2242951149
- chr3199501827
- chr4191273063
- chr5180857866
- chr6170899992
- chr7158821424
- chr8146274826
- chr9140273252
- chr10135374737
- chr11134452384
- chr12132349534
- chr13114142980
- chr14106368585
- chr15100338915
- chr1688827254
- chr1778774742
- chr1876117153
- chr1963811651
- chr2062435964
- chr2146944323
- chr2249691432
- chrX154913754
- chrY57772954
- chrM16571

*hg19*
- chr1249250621
- chr2243199373

- chr3198022430
- chr4191154276
- chr5180915260
- chr6171115067
- chr7159138663
- chr8146364022
- chr9141213431
- chr10135534747
- chr11135006516
- chr12133851895
- chr13115169878
- chr14107349540
- chr15102531392
- chr1690354753
- chr1781195210
- chr1878077248
- chr1959128983
- chr2063025520
- chr2148129895
- chr2251304566
- chrX155270560
- chrY59373566
- chrM16571

## Pseudoautosomal regions

 VAAST uses pseudoautosomal regions to determine the areas of the X and Y chromosome where heterozygous variant calls are allowed.  The information comes from UCSC genome browser front page for each build:

http://genome.ucsc.edu/cgi-bin/hgGateway?hgsid=194029611&clade=mammal&org=Human&db=hg18

### *hg18*

- chrX
    - five_prime:  1-2709520
    - three_prime:  154584238-154913754
- chrY
    - five_prime:  1-2709520
    - three_prime:  57443438-57772954

### *hg19*

- chrX
  - five_prime: 60001-2699520
  - three_prime: 154931044-155260560
- chrY
  - five_prime: 10001-2649520
  - three_prime: 59034050-59363566