Other GEMINI tools

Aaron Quinlan and Brent Pedersen
University of Utah

quinlanlab.org
The GEMINI annotate tool
Goal: extend a GEMINI database with custom annotations

annotate: adding your own custom annotations

It is inevitable that researchers will want to enhance the gemini framework with their own, custom annotations. gemini provides a sub-command called annotate for exactly this purpose. As long as you provide a tabix'ed annotation file in BED or VCF format, the annotate tool will, for each variant in the variants table, screen for overlaps in your annotation file and update a one or more new column in the variants table that you may specify on the command line. This is best illustrated by example.

Let's assume you have already created a gemini database of a VCF file using the load module.

```$ gemini load -v my.vcf -t snpEff my.db```

Now, let's imagine you have an annotated file in BED format (important.bed) that describes regions of the genome that are particularly relevant to your lab's research. You would like to annotate in the gemini database which variants overlap these crucial regions. We want to store this knowledge in a new column in the variants table called important_variant that tracks whether a given variant overlapped (1) or did not overlap (0) intervals in your annotation file.

To do this, you must first TABIX your BED file:

```$ bgzip important.bed
$ tabix -p bed important.bed.gz```
Goal: extend a GEMINI database with custom annotations

-a boolean Did a variant overlap a region or not?

Note

Formerly, the -a option was the -t option.

Now, you can use this TABIX’ed file to annotate which variants overlap your important regions. In the example below, the results will be stored in a new column called “important”. The -t boolean option says that you just want to track whether (1) or not (0) the variant overlapped one or more of your regions.

```
$ gemini annotate -f important.bed.gz -c important -a boolean my.db
```

Since a new columns has been created in the database, we can now directly query the new column. In the example results below, the first and third variants overlapped a crucial region while the second did not.

```
$ gemini query \
   -q "select chrom, start, end, variant_id, important from variants" \
   my.db \
   | head -3
chr22  100   101   1   1
chr22  200   201   2   0
chr22  300   500   3   1
```

http://gemini.readthedocs.org/en/latest/content/tools.html#annotate-adding-your-own-custom-annotations
Goal: extend a GEMINI database with custom annotations

-a  count How many regions did a variant overlap?

Instead of a simple yes or no, we can use the -t count option to count how many important regions a variant overlapped. It turns out that the 3rd variant actually overlapped two important regions.

```bash
$ gemini annotate -f important.bed.gz -c important -a count my.db
$ gemini query
  -q "select chrom, start, end, variant_id, crucial from variants" my.db
  | head -3
chr22   100   101   1   1
chr22   200   201   2   0
chr22   300   500   3   2
```

-a  extract Extract specific values from a BED file

Lastly, we may also extract values from specific fields in a BED file (or from the INFO field in a VCF) and populate one or more new columns in the database based on overlaps with the annotation file and the values of the fields therein. To do this, we use the -a extract option.

This is best described with an example. To set this up, let’s imagine that we have a VCF file from a different experiment and we want to annotate the variants in our GEMINI database with the allele frequency and depth tags from the INFO fields for the same variants in this other VCF file.
Goal: extend a GEMINI database with custom annotations

Annotating with VCF

Most of the examples to this point have pulled a column from a \textit{tabix} indexed bed file. It is likewise possible to pull from the INFO field of a \textit{tabix} index VCF. The syntax is identical but the \texttt{-e} operation will specify the names of fields in the INFO column to pull. By default, those names will be used, but that can still be specified with the \texttt{-c} column. Here are some example uses

```shell
# put a DP column in the db:
gemini annotate -f anno.vcf.gz -o list -e DP -t integer my.db

# ... and name it 'depth'
gemini annotate -f anno.vcf.gz -o list -e DP -c depth -t integer my.db

# use multiple columns
gemini annotate -f anno.vcf.gz -o list,mean -e DP,Qmeter -c depth,qmeter -t integer
```

Missing values are allowed since we expect that in some cases an annotation VCF will not have all INFO fields specified for all variants.

**Note**

We recommend decomposing and normalizing variants before annotating. See \texttt{Step 1. split, left-align, and trim variants} for a detailed explanation of how to do this.

http://gemini.readthedocs.org/en/latest/content/tools.html#annotate-adding-your-own-custom-annotations
Goal: extend a GEMINI database with custom annotations

Annotating with VCF

Most of the examples to this point have pulled a column from a `tabix` indexed bed file. It is likewise possible to pull from the INFO field of a `tabix` index VCF. The syntax is identical but the `-e` operation will specify the names of fields in the INFO column to pull. By default, those names will be used, but that can still be specified with the `-c` column. Here are some example uses:

```
# put a DP column in the db:
gemini annotate -f anno.vcf.gz -o list -e DP -t integer my.db

# ... and name it 'depth'
gemini annotate -f anno.vcf.gz -o list -e DP -c depth -t integer my.db

# use multiple columns

gemini annotate -f anno.vcf.gz -o list,mean -e DP,Qmeter -c depth,qmeter -
```

Missing values are allowed since we expect that in some cases an annotation VCF will not have all INFO fields specified for all variants.

Note

We recommend decomposing and normalizing variants before annotating. See Step 1. split, left-align, and trim variants for a detailed explanation of how to do this.
The `variant_impacts` table
**variant_impacts** tracks the functional impact on every transcript

...whereas the **variants** table stores only the most *deleterious*

<table>
<thead>
<tr>
<th><strong>The variant_impacts table</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>column_name</strong></td>
</tr>
<tr>
<td>variant_id</td>
</tr>
<tr>
<td>anno_id</td>
</tr>
<tr>
<td>gene</td>
</tr>
<tr>
<td>transcript</td>
</tr>
<tr>
<td>is_exonic</td>
</tr>
<tr>
<td>is_coding</td>
</tr>
<tr>
<td>is_lof</td>
</tr>
<tr>
<td>exon</td>
</tr>
<tr>
<td>codon_change</td>
</tr>
<tr>
<td>aa_change</td>
</tr>
<tr>
<td>aa_length</td>
</tr>
<tr>
<td>biotype</td>
</tr>
<tr>
<td>impact</td>
</tr>
<tr>
<td>impact_so</td>
</tr>
<tr>
<td>impact_severity</td>
</tr>
<tr>
<td>polyphen_pred</td>
</tr>
<tr>
<td>polyphen_scores</td>
</tr>
<tr>
<td>sift_pred</td>
</tr>
<tr>
<td>sift_scores</td>
</tr>
</tbody>
</table>
The mendelian_error tool
mendelian_error

mendelian_error: Identify non-mendelian transmission.

<table>
<thead>
<tr>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>This tool requires that you identify familial relationships via a PED file when loading your VCF into gemini via:</td>
</tr>
<tr>
<td><code>gemini load -v my.vcf -p my.ped my.db</code></td>
</tr>
</tbody>
</table>

We can query for mendelian errors in trios including:

- loss of heterozygosity
- implausible de-novo mutations
- de-novo mutations
- uniparental disomy

```bash
$ gemini mendel_errors --columns "chrom,start,end" test.mendel.db --gt-pl-max 1
```

<table>
<thead>
<tr>
<th>chrom</th>
<th>start</th>
<th>end</th>
<th>family_members</th>
<th>family_genotypes</th>
<th>violation</th>
<th>violation_prob</th>
</tr>
</thead>
<tbody>
<tr>
<td>chr1</td>
<td>10670</td>
<td>10671</td>
<td>dad,mom,child</td>
<td>G/G,G/G,G/C</td>
<td>plausible de novo</td>
<td>0.962</td>
</tr>
<tr>
<td>chr1</td>
<td>28493</td>
<td>28494</td>
<td>dad,mom,child</td>
<td>T/C,T/T,C/C</td>
<td>loss of heterozygosity</td>
<td>0.660</td>
</tr>
<tr>
<td>chr1</td>
<td>28627</td>
<td>28628</td>
<td>dad,mom,child</td>
<td>C/C,C/C,C/T</td>
<td>plausible de novo</td>
<td>0.989</td>
</tr>
<tr>
<td>chr1</td>
<td>267558</td>
<td>267560</td>
<td>dad,mom,child</td>
<td>C/C,C/C,CT/C</td>
<td>plausible de novo</td>
<td>0.928</td>
</tr>
<tr>
<td>chr1</td>
<td>537969</td>
<td>537970</td>
<td>dad,mom,child</td>
<td>C/C,C/C,C/T</td>
<td>plausible de novo</td>
<td>0.996</td>
</tr>
<tr>
<td>chr1</td>
<td>547518</td>
<td>547519</td>
<td>dad,mom,child</td>
<td>G/G,G/G,G/T</td>
<td>plausible de novo</td>
<td>1.000</td>
</tr>
<tr>
<td>chr1</td>
<td>589081</td>
<td>589086</td>
<td>dad,mom,child</td>
<td>G/G,GAGAA,GAGAA,G/G</td>
<td>uniparental disomy</td>
<td>0.940</td>
</tr>
<tr>
<td>chr1</td>
<td>749688</td>
<td>749689</td>
<td>dad,mom,child</td>
<td>T/T,T/T,G/G</td>
<td>implausible de novo</td>
<td>0.959</td>
</tr>
<tr>
<td>chr1</td>
<td>788944</td>
<td>788945</td>
<td>dad,mom,child</td>
<td>C/C,G/G,G/G</td>
<td>uniparental disomy</td>
<td>0.914</td>
</tr>
<tr>
<td>chr1</td>
<td>1004248</td>
<td>1004249</td>
<td>dad,mom,child</td>
<td>G/G,G/G,G/C</td>
<td>plausible de novo</td>
<td>1.000</td>
</tr>
</tbody>
</table>
Speeding up database loading with vcfanno
GEMINI \textbf{current}

- most annotations are fixed
- can add some custom annotations after load
- loading is slow
- stuck to single genome version
GEMINI future

- recommended, vetted annotations
- fast loading
- any organism supported by VEP / SnpEff
- custom annotations treated same as vetted
Configuration

[[annotation]]
file="ALL.wgs.phase3_shapeit2_mvncall_integrated_v5a.20130502.sites.tidy.vcf.gz"
fields=["AF", "AMR_AF", "EUR_AF", ...]
names=["in_1kg_flag", "aaf_1kg_amr_float", "aaf_1kg_eas_float", ...]
ops=["flag", "max", "max", ...]

Specify an annotation, file, which (VCF INFO) fields to pull, and how to report them.

We’ll include a vetted file like this for gemini for human, but users can modify it and/or create their own for other organisms.

Possible to create custom database with only columns of interest.

https://github.com/brentp/vcfanno/blob/master/example/gem.conf
10 million ExAC variants annotated with 34 annotations from 11 distinct files in ~30 minutes

current gemini takes at least 40X longer to load the same number of variants.
Modularize functionality

- pedigree (determine modes of inheritance)
- inheritance models (rules for autosomal rec/dom, de novo)
- effect parsing/prioritizing
  - normalize between SnpEff / VEP
  - prioritize by impact (missense over synonymous)

Separating functionality improves code reuse, eases testing, and simplifies code maintenance
gemini query performance improvements with bcolz