This small package is written in c# and built with .Net Core 2.2.3 with command “dotnet.exe publish -c Release -r win10-x64 --self-contained”. And you do need .Net Core 2.2 runtime on your computer to run it.

To run it, download the whole package from github (<https://github.com/bioinfojh/TestExAC>) or you can download the “TestExAC\bin\Release\netcoreapp2.2\win10-x64\publish” folder and run the TestExAC.exe in that folder as “TestExAC myvcf.vcf myoutput.txt” to test the program. One example output file (example\_output.txt) is also included in the “doc” folder.

Or you can compile the source code and regenerate the executable. (Can use Visual Studio Code or visual studio)

The package can be used to parse VCF file and then annotate with ExAC API. For demo purpose, it is written mostly as a skeleton to support general VCF file parsing and the annotation processing. It can be further extended to a version with full functions in the future.

Requirements and strategy:

1. Type of variation, if there are multiple possibilities, annotate with the most deleterious possibility.

*The VCF file has “Type” in the “INFO” field and we can extract that as it is. And ExAC also return the detailed “Consequence” to transcripts, we will parse and export the most deleterious consequence along with its associated gene/transcripts. Some variants do not have mapping at ExAC site, so the consequence will be annotated as “.”. Alternatively, we could load the reference library and gene model to do some prediction ourselves, that is outside the scope of this demo project, will be left to future coding.*

1. Depth of sequence coverage at the site of variation.

*Use “DP” field of the “INFO”, notice this is the depth combining all samples*

1. Number of reads supporting the variant.

*Use “AO” fields for each alternative allele variant and “RO” field in “INFO” for reference allele*

1. Percentage of reads supporting the variant versus those supporting reference reads.

*For each variant, use AO/RO, you might get infinity if RO = 0; alternative, we probably can report AO/DP + “:”+RO/DP if try to avoid infinit.*

1. Allele frequency of variant from Broad Institute ExAC Project API

*Use /rest/variant and parse the variant section of the json output*

1. Additional optional information from ExAC that you feel might be relevant.

*I export Consequence, related Gene/Transcripts, rsID, AlleleCount, AlleleNumber, HomozygousNumber, AlleleFrequence from ExAC json response as well as an ExAC browser link*

Notice:

For variants with indel, queries using the existing reference allele and alternative allele often fail, e.g.

Insert 1-6184728-TGGGGGGGGGGGA-TGGGGGGGGGGGGA is recorded as 1-6184728-T-TG

Delete 1-43771016-TAA-TA is recorded as 1-43771016-TA-T by ExAC , etc. I use a strategy to remove suffix and then prefix and rescue lots of variants. Notice remove prefix will change the query starting position

There are other special case like:

variant 1-1650797-ATTTT-GTTTC in the example vcf, ExAC treats that as two independent variants at

1-1650801-T-C and 1-1650797-A-G, searching the original variant will not yield any results. Those complex cases will be left for future improvements.

Some explanation of the code:

General steps:

Load VCF file => Annotate the variance with ExAC API => parse json response and prepare outputs

1. “.vcf” file is loaded with VCFFile class (VCFFile.cs).
   1. The current demo version only supports local “.vcf” files. A full-fledged version should support.gz (bzgf format) and allow online streaming of VCF files from web and cloud.
   2. Since VCF files could be huge, to save memory and computing resources, a delay- loading mechanism is used. During initial VCF file loading, only the header sections are processed until the first line of variants. The remaining variant lines will be loaded on demand with transverse functions with c# delegates.
2. “ExACAnnotator” is used to annotator VCF files with its AnnotateVcfFile() function
   1. Implement a general IVcfAnnotator interface that can be used to implement other VCF annotators (e.g. implement GTAK Funcotator)
   2. Support both single VCF query mode and batch query mode. Default using batch query mode with 1000 lines by default.
   3. This class will prepare json queries, submit those queries to ExAC API and then parse the json responses into ExACVariantAnnotation objects
3. ExACVariantAnnotation is used to capture json results from ExAC API
   1. It implements a “IVariantAnnotation” interface for potentially other annotation tool extensions.
   2. It captures the five major sections of ExAC API json response (consequence, base\_coverage, variant, metrics and any\_covered).
   3. To find the most deleterious consequence for a variant, I could use “/rest/variant/ordered\_csqs” API to preform another round of search, but that adds additional searching overhead. After some searching, I found ExAC consequence ordering logic from their source code and reimplemented that using ExAC’s preferred csq\_order.
4. Some variants have multiple alternative alleles, I choose to output each of them as an independent line during output so that the results can be opened and checked in Excel easily. But it is trivial to combine the lines to make a “.vcf” type of outputs. The DP and OA fields will keep their original values of the variant.
5. The results can be further validated by GATK using its VariantToTable and Funcotator tools, I tried that in docker. On Linux, it should be easy to wrap GATK and create another independent VCF annotator with those tools.