The Journal of Open Source Software

cerebra: A tool for fast and accurate summarizing of variant calling format (VCF) files

Lincoln Harris[1], **Rohan Vanheusden**[1], **Olga Botvinnik**[1], **and Spyros Darmanis**[1]

1 Chan Zuckerberg Biohub, San Francisco, CA

DOI: 10.21105/joss.02432

Software

- Review
- Repository
- Archive

Editor: Lorena Pantano

Reviewers:

- @betteridiot
- @afrubin

Submitted: 25 June 2020
Published: 03 July 2020

Motivation

A single "typo" in the genome can have profound consequences on an organism's biology. Identifying the functional consequences of genomic typos (termed "variants") is a fundamental challenge in bioinformatics. Tools exist for identifying variants and predicting their functional consequences, however, wrangling variant calls and functional predictions across thousands of samples remains an unsolved problem. `cerebra` addresses this need by offering a fast and accurate framework for summarizing variant calls and functional predictions across many samples.

To find variants in the genome, researchers often begin with a DNA-sequencing (DNA-seq) or RNA-sequencing (RNA-seq) experiment on their samples of interest. Sequencing is followed by alignment of reads to the reference genome with tools like STAR or BWA, followed by variant calling with tools like GATK HaplotypeCaller or freebayes (A, CA, F, J, & others, 2013; E & G, 2012; H & R, 2009; R, V, M, & others, 2018). Variant callers produce tab delimited text files in the variant calling format (VCF) for each processed sample. VCF files encode: i) the genomic position, ii) reference vs. observed DNA sequence, and iii) quality associated with each observed variant. Shown below are the first 4 lines of a sample VCF file. Note that only a single record is displayed, and that the record line has been artificially wrapped.

```
##fileformat=VCFv4.2
##source=HaplotypeCaller
#CHROM  POS ID  REF ALT QUAL    FILTER  INFO    FORMAT
chr1    631391  .   C   T   72.28   .   AC=2;AF=1.00;AN=2;DP=2;
        ExcessHet=3.0103;FS=0.000;MLEAC=1;MLEAF=0.500;MQ=NaN;
        QD=25.36;SOR=2.303  GT:AD:DP:GQ:PL  1/1:0,2:2:6:84,6,0
```

Current methods for variant calling are incredibly powerful and robust, however, a single sequencing run can generate as many as 10^8 unique VCF records. Only a small portion of these VCF records are likely to be relevant to the researcher. In addition, variant callers report only the genomic location and not the *functional* consequences of the variant, *i.e.* the effect the variant has on the translated protein sequence. We refer to these functional variants as "peptide-level variants." To address the unmet need for high-throughput VCF summary tools, we introduce `cerebra`, a python package that provides fast and accurate peptide-level summarizing of VCF files.

Functionality

`cerebra` comprises three modules: i) `germline-filter` removes variants that are common between control/germline samples and samples of interest, ii) `count-variants` reports total number of variants in each sample, and iii) `find-peptide-variants` reports likely peptide-level variants in each sample. Here we use *variant* to refer to single nucleotide polymorphisms (SNPs) and short insertions/deletions. `cerebra` is not capable of reporting larger structural variants such as copy number variations and chromosomal rearrangements.

A data structure crucial to `cerebra` is the *genome interval tree*, which matches RNA transcripts and peptides to each feature in the genome (Figure 1). Interval trees are self-balancing binary search trees that store numeric intervals and can quickly retrieve every such interval that overlaps a given query interval (*see also*). Given n nodes, interval trees have theoretical average-case $O(\log n)$ and worst-case $O(n)$ time complexity for search operations, making them tractable for genome-scale operations (T, C, R, & C, 2009, *see also*). Tree construction proceeds at $O(n \log n)$ time complexity, making construction rather than search the bottleneck for most VCF sets (A & C, 2007). The *genome interval tree* is constructed with a reference genome sequence (FASTA format, often with a `.fa` extension), and a genome annotation (gene transfer format, GTF, `.gtf` extension). We rely on the ncls python library for fast interval tree construction and lookup operations.

In order to analyze multiple VCF records at once, we use multiprocessing with the Python pathos library module. We extract relevant information – including genomic interval, observed base, and read coverage – from each variant record. In the `germline-filter` module variants are compared to one another and filtered out if found to be identical. In `count-variants` variants are simply matched to whichever gene they came from. In `find-peptide-variants` variants are queried against our *genome interval tree* – if a matching interval is found we convert the DNA-level variant to a peptide-level variant. Finally, peptide-level variants across all VCF files are reported in tabular format.

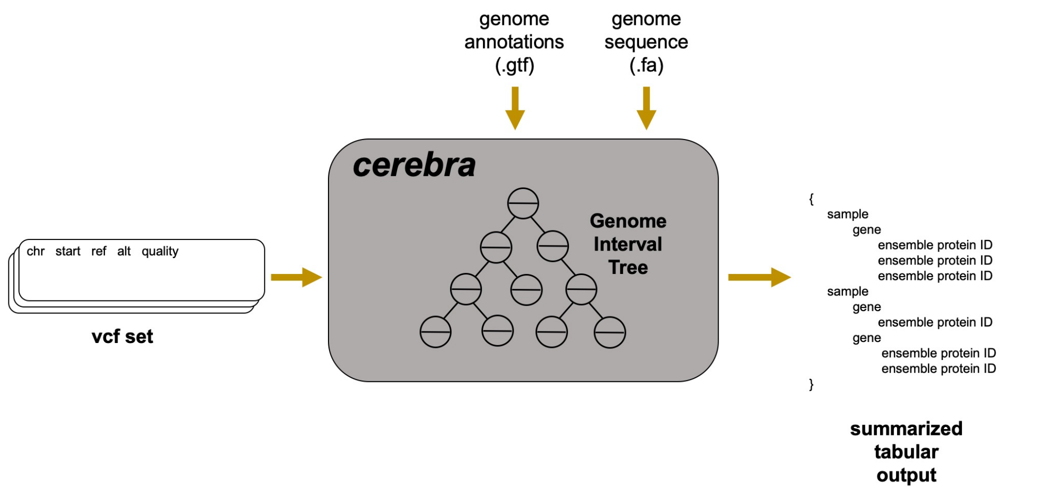

Figure 1: Workflow describing the `find-peptide-variants` module. We construct a genome interval tree from a genome annotation (.gtf) and a reference genome sequence (.fa), then process VCF files in parallel to create a single tabular output file (CSV or JSON).

`germline-filter`

If the research project is centered around a "tumor/pathogenic vs control" question, then `germline-filter` is the proper starting point. This module removes germline variants that are common between the control and the experimental tissue so as to not bias the results by including non-pathogenic variants. The user provides a very simple metadata file (see

README.md) that indicates which experimental samples correspond to which control samples. Using the vcfpy library we quickly identify shared variants across control/experimental matched VCF files, then write new VCFs that contain only the unique variants. These steps are performed by a subprocess pool so that we can process multiple discreet chunks of input at the same time. There is also the option to limit the reported variants to those found in NCBI's dbSNP and the Wellcome Sanger Institute's COSMIC databases. This option is designed to give the user a higher degree of confidence in the pathogenicity of each variant. If independent experiments have reported a given variant in pathogenic human tissue, it is less likely to be an artifact. The output of `germline-filter` is a set of trimmed-down VCF files, which will be used for the next two steps. If you do not have access to "control" tissue then proceed directly to `count-variants` or `find-peptide-variants`.

`count-variants`

The `count-variants` module reports the raw variant counts for every gene across every sample. We first create a *genome interval tree* from the reference GTF, then read in a VCF file and convert it to a vcfpy object, then processes VCF records in parallel. Each variant is matched to its corresponding gene, and gene-wise counts are stored in shared memory. If working with cancer samples, the user has the option to filter out all variants that are not found in the COSMIC database and are therefore less likely to be pathogenic. `count-variants` produces two output files, one containing raw variant counts and one containing COSMIC filtered variant counts for every gene in the genome.

`find-peptide-variants`

The `find-peptide-variants` module reports the peptide-level consequence of genomic variants. First we load the reference GTF, then construct an index (.fai) of the genome fasta file with pyfaidx to enable fast random memory access. We then create a *genome interval tree* that will be used to quickly match genomic coordinates from VCF records to peptide-level variants. The user again has the option to filter out variants not found in the COSMIC database. VCF records are read in simultaneously; individual records are converted to *GenomePosition* objects to keep track of their genomic intervals and observed DNA bases. *GenomePositions* are then queried against the *genome interval tree*. If an overlapping interval is found we retrieve the peptide-level variant from this node of the *genome interval tree*. Peptide-level variants are converted to ENSEMBL protein IDs, in accordance with the HGVS sequence variant nomenclature. The output is a hierarchically ordered text file (CSV or JSON) that reports the the ENSEMBL protein ID and the gene associated with each variant, for each experimental sample.

Variant callers are known to produce a great deal of false positives, especially when applied to single-cell RNA-seq data (M, H, M, S, & others, 2017). To address this concern we have included the `--report_coverage` option. If indicated this option will report counts for both variant and wildtype reads at all variant loci. We reasoned that variants with a high degree of read support are less likely to be false positives. This option is designed to give the user more confidence in individual variant calls.

We should emphasize that `find-peptide-variants` does not *definitively* report peptide-level variants but rather the *likely* set of peptide variants. Definitively reporting protein variants from RNA-seq requires knowledge of alternate splicing – this represents an open problem in the field (Y & G, 2017). For example, if a read picks up a variant in exon 2 of a given gene, we can report each of the potential spliceforms of that gene that contain exon 2, but we **cannot** infer which of those particular spliceforms are actually present in our sample (see Figure 2). For the example shown in Figure 2 we would translate and report *t1* and *t3* as both of these contain exon 2. It is possible the sample does not actually express both of these spliceforms, however, determining the spliceform landscape of a sample from RNA-seq is outside the scope of this project.

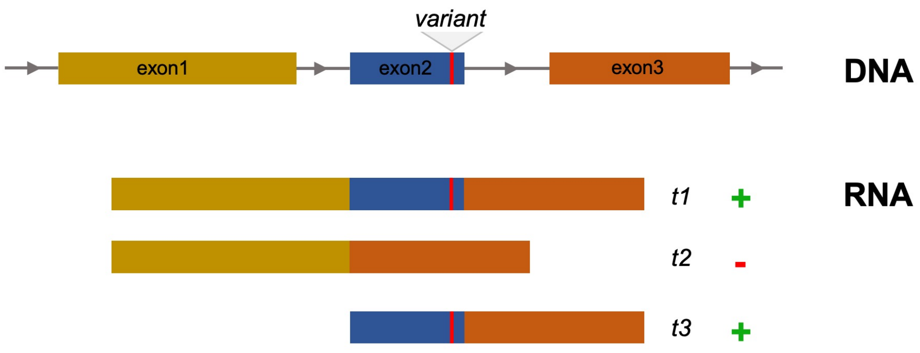

Figure 2: For a given mutational event, `cerebra` reports ALL potentially affected spliceforms.

To assess performance of `find-peptide-variants` we obtained VCFs from a single-cell RNA-seq study conducted on lung adenocarcinoma patient samples (A et al., 2019). These VCFs were produced with STAR (alignment) and GATK HaplotypeCaller (variant calling), and are on the order of megabytes, typical of a single-cell RNA-seq experiment. `cerebra` was run on standard hardware (MacBook Pro, 2.5GHz quad-core processor, 16 GB RAM). As show in Figure 3 `cerebra` processed a set of 100 VCF files in approximately 34 minutes.

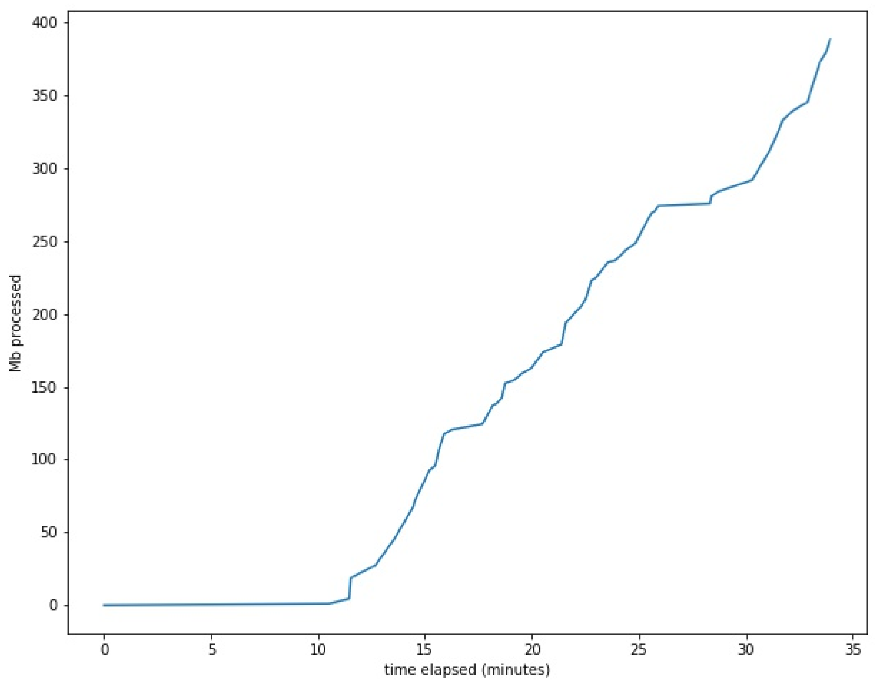

Figure 3: `cerebra` processes 100 VCF files (~400 Mb in total) in ~34 minutes.

The first 10 or so minutes of `cerebra find-peptide-variants` do not involve any VCF processing, instead, this time is attributed to the *genome interval tree* construction phase. After the tree is built, files are processed in a near-linear manner. Also of note is that `cerebra`'s search operations take advantage of multiprocessing. `cerebra` should scale better to high-memory machines than single-threaded tools, though it has been designed to run on standard hardware.

Conclusions

RNA/DNA sequencing paired with fast and accurate summarizing of variants is often crucial to understanding the biology of an experimental system. We present a tool that can be used to quickly summarize the variant calls contained within a large set of VCF files. As sequencing costs continue to drop, large-scale variant calling will become accessible to more members of the community, and summary tools like `cerebra` will become increasingly important. Our tool offers the advantages of parallel processing and a single, easy-to-interpret output file (CSV or JSON).

`cerebra` is already enabling research, see (A et al., 2019), a study that examines the tumor microenvironment of late-stage drug-resistant carcinomas with single-cell RNA-sequencing. Understanding the mutational landscape of individual tumors was essential to this study, and given the sheer volume of VCF records would not have been possible without `cerebra`. We hope that `cerebra` can provide an easy-to-use framework for future studies in the same vein.

Acknowledgments

Funding for this work was provided by the Chan Zuckerberg Biohub. The authors would like to thank Ashley Maynard, Angela Pisco and Daniel Le for helpful discussions and support.

Correspondence

Please contact `ljharris018@gmail.com`

Code

`cerebra` is written in Python 3. Code and detailed installation instructions can be found at https://github.com/czbiohub/cerebra. In addition `cerebra` can be found on PyPi.

References

A, A., & C, L. (2007). Nested Containment List (NCList): a new algorithm for accelerating interval query of genome alignment and interval databases. *Bioinformatics*.

A, D., CA, D., F, S., J, D., & others. (2013). STAR: ultrafast universal RNA-seq aligner. *Bioinformatics*.

A, M., C, M., J, R., L, H., S, D., C, B., T, B., et al. (2019). Heterogeneity and targeted therapy-induced adaptations in lung cancer revealed by single-cell RNA-seq. *bioRxiv*.

E, G., & G, M. (2012). Haplotype-based variant detection from short-read sequencing. *arXiv*.

H, L., & R, D. (2009). Fast and accurate short read alignment with Burrows-Wheeler Transform. *Bioinformatics*.

M, E., H, E. A., M, M., S, Q., & others. (2017). Single-Cell Analysis of Human Pancreas Reveals Transcriptional Signatures of Aging and Somatic Mutation Patterns. *Cell*.

R, P., V, R.-R., M, D., & others. (2018). Scaling accurate genetic variant discovery to tens of thousands of samples. *bioRxiv*.

T, C., C, L., R, R., & C, S. (2009). *Introduction to Algorithms* (3rd ed.). MIT Press. ISBN: 978-0-262-03384-8

Y, H., & G, S. (2017). BRIE: transcriptome-wide splicing quantification in single cells. *Genome Biology*.