Bioinformatics, 33(12), 2017, 1867–1869
doi: 10.1093/bioinformatics/btx057
Advance Access Publication Date: 6 February 2017
Applications Note

Genome analysis

cyvcf2: fast, flexible variant analysis with Python

Brent S. Pedersen* and Aaron R. Quinlan*

Department of Human Genetics, Department of Biomedical Informatics, and USTAR Center for Genetic Discovery, University of Utah, Salt Lake City, UT, USA

*To whom correspondence should be addressed.

Associate Editor: John Hancock

Received on December 9, 2016; revised on January 13, 2017; editorial decision on January 24, 2017; accepted on January 28, 2017

Abstract

Motivation: Variant call format (VCF) files document the genetic variation observed after DNA sequencing, alignment and variant calling of a sample cohort. Given the complexity of the VCF format as well as the diverse variant annotations and genotype metadata, there is a need for fast, flexible methods enabling intuitive analysis of the variant data within VCF and BCF files.
Results: We introduce *cyvcf2*, a Python library and software package for fast parsing and querying of VCF and BCF files and illustrate its speed, simplicity and utility.
Contact: bpederse@gmail.com or aaronquinlan@gmail.com
Availability and Implementation: *cyvcf2* is available from https://github.com/brentp/cyvcf2 under the MIT license and from common python package managers. Detailed documentation is available at http://brentp.github.io/cyvcf2/

1 Introduction

The VCF format (Danecek *et al.*, 2011) is the standard for representing genetic variation observed in DNA sequencing studies. The strength of the VCF format is its ability to represent the location of a variant, the genotypes of the sequenced individuals at each locus, as well as extensive variant metadata. Furthermore, the VCF format provided a substantial advance for the research community, as it follows a rigorous format specification that enables direct comparison of results from multiple studies and facilitates reproducible research. However, the consequence of this flexibility and the rather complicated specification of the VCF format, is that researchers require powerful software libraries to access, query and manipulate variants from VCF files.

While bcftools (Li, 2011) provides a high performance programming interface in the C programming language, as well as a powerful command line interface, developing custom analyses requires either expertise in C, or combinations of multiple options and sub-commands from the bcftools package. Furthermore, some analyses (e.g. the first example below) are not yet possible with the bcftools framework. In contrast, pysam (unpublished) and pyvcf provide researchers with direct access to VCF files through Python programming libraries. However, while they are relatively simple to use, these libraries lack the speed required to rapidly manipulate and prioritize variants in VCF files from large-scale studies (e.g. whole-genome sequencing) that often yield tens of millions of genetic variants and many individual genotypes per variant.

2 Approach

In developing *cyvcf2*, we sought to create a high performance library that also provides researchers with an intuitive Python interface for manipulating VCF files. *Cyvcf2* provides the ability to filter variants based upon variant annotation, interrogate the details of each sample's genotype information, and rapidly compute both variant and sample-level statistics. Computational performance is enabled by leveraging *htslib*, an efficient software library for accessing VCF files using the C programming language. Cyvcf2 wraps *htslib* for use within Python programming interface with Cython (Behnel *et al.*, 2011). Cython provides the ability to write performance critical aspects of *cyvcf2* in C, while exposing an interface that is familiar to python programmers.

Cyvcf2 strives to provide the user with the ability to access every aspect of the VCF specification. The most commonly accessed variant properties are provided as Python attributes. For example, each variant in a VCF file can be accessed through a Python iterator and each variant object has a *start* attribute that returns the 0-based start coordinate. Similarly, the *end* attribute returns the end of the variant and is properly computed for structural variants, even when that requires consulting the *END* defined in the INFO field of the VCF record.

When accessing the sample genotype data (e.g. the predicted genotype, observed sequencing depth and genotype quality) for each variant, cyvcf2 automatically returns a high performance numpy (Van Der Walt *et al.*, 2011) array, which enables efficient manipulation and access of all samples, a single sample, or specific subsets of samples, even when the VCF or BCF file includes thousands of individuals.

3 Usage

In an effort to demonstrate the power and performance of *cyvcf2*, the following sections highlight typical VCF analyses and illustrate commonly used features in *cyvcf2*. Other examples and further details of the *cyvcf2* library can be found at http://brentp.github.io/cyvcf2/.

3.1 Counting heterozygous genotypes per sample

Imagine one is interested in counting the number of high quality heterozygous genotypes for each sample in a given locus of interest. The following example first opens a VCF file (line 3), then creates an iterator of each variant that overlaps our locus of interest (line 5). We ignore any INDEL (line 6) or low quality (i.e. Phred-scaled quality less than 10; line 7) variant. We then extract a numpy array of the observed sequencing depths for each sample (line 9) and increment the counts if each sample is a heterozygote and has an alternate depth greater than 10. Alternate depths are stored in the second column of the depths array, while the observed depth for the reference allele is stored in the first column. For convenience, *cyvcf2* also provides keywords for each of the three diploid genotypes: homozygous for the reference allele (HOM_REF), heterozygous (HET) and homozygous for the alternate allele (HOM_ALT), as well as an unknown genotype (UNKNOWN). Finally, in line 10, we report, for each sample, the number of heterozygous SNP genotypes having an alternate depth greater than 10 observed in the locus of interest.

Listing 1. Counting heterozygous genotypes per sample in an indexed VCF file

```
1  import cyvcf2
2  import numpy as np
3  vcf = cyvcf2.VCF(path)
4  sample_counts = np.zeros(len(vcf.samples), dtype=float)
5  for variant in vcf(“chr1:229993-329993”):
6  if variant.is_indel: continue
7  if variant.QUAL < 10: continue
8  depths = variant.format(“AD”)
9  sample_counts[(depths[:, 1] > 10) & (variant.gt_types
     == vcf.HET)] += 1
10 print(zip(vcf.samples, sample_counts))
```

3.2 Finding *de novo* mutations in a trio

Searching for *de novo* mutations (DNM) is common in studies of Mendelian disease. The following example demonstrates how *cyvcf2* can be used to screen for *de novo* mutations in a family trio. This example assumes that the proband, mother and father are the first, second and third samples in the VCF file (line 8). A common source of false positive DNM predictions is a lack of sequencing depth in one or both parents to allow the detection of an allele being transmitted from parent to child. Therefore, we require at least 10 aligned sequences for each family member at each candidate DNM (line 511). Next, we require the mother and father to have homozygous genotypes, whereas the child should have a heterozygous genotype, thereby indicating a potential DNM (line 13). Lastly, to further reduce false positive DNMs, we wish to ignore any DNM candidate in which the alternate allele was observed on one or more aligned sequences from the mother or father at the given locus (line 14).

Listing 2. Calling De novo mutations

```
1  import numpy as np
2  import sys
3  from scipy.stats import binom_test
4  path = sys.argv[1]
5
6  import cyvcf2
7  vcf = cyvcf2.VCF(path)
8  PRO, MOM, DAD = range(3)
9  for v in vcf:
10 if v.QUAL < 10: continue
11 if np.any(v.gt_depths < 10): continue
12 refs, alts = v.gt_ref_depths, v.gt_alt_depths
13 if not all(v.gt_types == [vcf.HET, vcf.HOM_REF,
   vcf.HOM_REF]): continue
14 if alts[MOM] > 1 or alts[DAD] > 1: continue
15 print(“%s\t%d\t%s\t%s\t%s” % (v.CHROM, v.start,
   v.REF, “,”.join(v.ALT), “,”.join(v.gt_bases)))
```

3.3 Performance

While simplicity and flexibility were important design goals, we also sought to achieve higher performance than existing Python-based VCF processing libraries. To demonstrate *cyvcf2*'s speed, we measured the running time required by *cyvcf2*, bcftools (Li, 2011), pysam (unpublished; pysam.readthedocs.io) and pyvcf (unpublished; pyvcf.readthedocs.io) to analyze all variants in the VCF file for chromosome 22 from the 1000 Genomes Project (Consortium *et al.*, 2015), which includes genotypes for 2,504 samples. We developed a script for each method (see https://github.com/brentp/cyvcf2/tree/master/scripts for details) to count the number of bi-allelic variants with a quality of at least 20 and an alternate allele frequency less than or equal to 0.05. Since both bcftools and *cyvcf2* leverage *htslib*, *cyvcf2* achieves processing speeds that are equivalent *bcftools* (Table 1). Furthermore, *cyvcf* is 6.9 and 168.1 times faster than pysam and pyvcf, respectively. We emphasize *cyvcf2*'s performance since it is not restricted to a limited set

Table 1. Time required to filter the 1000 genomes VCF for chromosome 22 using a single Intel Xeon CPU E5-26900@2.90GHz

Name	Time (s)	Ratio
bcftools	224	0.9
cyvcf2	248.6	1
pysam	1711.5	6.89
pyvcf	41786.6	168.1

of pre-defined filters and operations. Cyvcf2 offers full programmatic flexibility that can come with minimal performance penalties owing to the careful design. In particular, the underlying sample genotype data are exposed to the user as a *numpy* (Van Der Walt *et al.*, 2011) array that uses the original memory allocated by *htslib*. This speed and memory efficiency, along with the extreme flexibility offered by Python, are central to the inherent utility of *cyvcf2*.

4 Discussion

We have developed *cyvcf2*, a fast, flexible and efficient software package that enables simple yet powerful manipulations of VCF files. Its speed and analytical power offer research functionality to investigators studying genetic variation in diverse contexts and species.

Funding

This research was supported by a US National Human Genome Research Institute award to ARQ (NIH R01HG006693).

Conflict of Interest: none declared.

References

Behnel,S. *et al*. (2011) Cython: the best of both worlds. *Comput. Sci. Eng*., **13**, 31–39.

Consortium,G.P. *et al*. (2015) A global reference for human genetic variation. *Nature*, **526**, 68–74.

Danecek,P. *et al*. (2011) The variant call format and vcftools. *Bioinformatics*, **27**, 2156–2158.

Li,H. (2011) A statistical framework for SNP calling, mutation discovery, association mapping and population genetical parameter estimation from sequencing data. *Bioinformatics*, **27**, 2987–2993.

Van Der Walt,S. *et al*. (2011) The numpy array: a structure for efficient numerical computation. *Comput. Sci. Eng*., **13**, 22–30.