`SiNPle`: Simplified Inference of Novel Polymorphisms from Large coveragE

`SiNPle` is a fast and sensitive variant caller based on a simplified Bayesian approach to compute the posterior probability that a variant is not generated by sequencing errors or PCR artefacts. It is particularly suited to low-frequency variant calling, as described in Ferretti et al. (2019). While calling heterozygous variants in samples with defined ploidy is relatively straightforward — for instance, the expected allele frequency is 50% or 100% in the diploid case — the underlying assumptions for low-frequency variant calling are very different and require a different statistical approach. In the Bayesian model used by `SiNPle` we explicitly consider the limit in which ploidy tends to infinity, which is equivalent to a mixture with a very large number of molecules, and thus no artificial constraints are imposed. You can still use `SiNPle` if you know what the ploidy of your system is — you just need to post-filter the calls made by `SiNPle`.

Comparison with other callers

Some variant callers, such as GATK, assume a ploidy of two. Others have hidden hardwired thresholds — for instance, a minimum number of covering reads needed to make a call — and hence miss many existing low-frequency variants. A detailed comparison with the low-frequency variant callers LoFreq and VarScan2 performed on CirSeq data can be found in Ferretti et al., 2019; it shows that the approach adopted by `SiNPle` is faster and much more accurate.

Citation

If you use `SiNPle`, please cite

> Ferretti L, Tennakoon C, Silesian A, Freimanis G, Ribeca P. SiNPle: Fast and Sensitive Variant Calling for Deep Sequencing Data. Genes. 2019; 10(8):561. https://doi.org/10.3390/genes10080561

Table of contents

1. Installation

> ⚠️ Note that the only operating systems we officially support are Linux and MacOS. ⚠️
>
> OCaml is highly portable and you might be able to manually compile/install everything successfully on other platforms (for instance, Windows) but you will have to do it yourself.

There are several possible ways of installing the software on your machine: through `conda`; by downloading pre-compiled binaries (Linux and MacOS x86_64 only); or manually.

1.1. Conda channel

> 🚧 Coming soon! 🚧

1.2. Pre-compiled binaries

You can download pre-compiled binaries for Linux and MacOS x86_64 from our releases.

1.3. Manual install

Alternatively, you can install `SiNPle` manually by cloning and compiling its sources. You'll need an up-to-date distribution of the OCaml compiler and the Dune package manager for that. Both can be installed through OPAM, the official OCaml distribution system. Once you have a working OPAM distribution you'll also have a working OCaml compiler, and Dune can be installed with the command

```
$ opam install dune
```

if it is not already present. Make sure that you install OCaml version 4.12 or later.

You'll also need a copy of the sources for the BiOCamLib library. We'll assume that you have cloned the repository in the directory `../BiOCamLib` with respect to the `SiNPle` sources; you'll have to modify the file `BUILD` in the `SiNPle` directory if that is not the case.

Then go to the directory into which you have downloaded the latest `SiNPle` sources, and type

```
$ ./BUILD
```

That should generate the executable `SiNPle`. Copy it to some favourite location in your PATH, for instance `~/.local/bin`.

2. How to run it

`SiNPle` reads as input the pileup format produced by commands such as `samtools mpileup`.

An example of generating SiNPle variant calls from the bamfile example.bam would be

```
$ samtools mpileup -d 1000000 -a -A -B -Q 0 -x example.bam | SiNPle > variants.txt
```

3. Interpreting the output

The output generated by `SiNPle` is made of tab-separated lines, one for each position of the sequence(s) being explored for which there is converage in the pileup. The number of columns is variable, and equals $2 + 4 * n$, where n is the number of genotypes — i.e., the different symbols occurring in the pileup; they might be nucleotides such as `A`, `C`, `G`, `T`, `N`, or indels such as `+AAA`. An example record might be

`My_precious` Tab `12037` Tab `A` Tab `85760` Tab `36.7` Tab `1` Tab `G` Tab `61` Tab `35.3` Tab `0.0243`

The first two columns,

`My_precious` Tab `12037`

are sequence name and position. They are followed by groups of four columns, such as

`A` Tab `85760` Tab `36.7` Tab `1`

which are the sequence of the genotype at that position (`A` in this case) with the number of reads it occurs in (85760), the average sequencing (*not* alignment) quality for such reads, and the posterior probability that that genotype is actually present in what you are sequencing (1 in this case).

In general there'll be more than one genotype in the output for each position (as many as seen in the pileup, in fact) possibly including the one present in the reference, which has no special status in the model used by `SiNPle`. They are

all considered independently, as the statistical hypothesis is that we are in the presence of a mixture of genotypes. In this case, at this position one also sees 61 `G`s, but, given the priors, the frequencies, and the qualities, the model thinks it is noise (posterior probability $p = 0.0234$). That is not always the case, and, as expected, sometimes you can have several genotypes whose presence at the same position is considered statistically significant by `SiNPle`.

Typically you would keep genotypes that have, for instance, $p >= 0.95$. The priors can be adjusted as needed, in particular the ones for indels.

4. Command line syntax

This is the full list of command line options available for the program `SiNPle`. You can visualise the list by typing

```
$ SiNPle -h
```

in your terminal. You will see a header containing information about the version:

```
This is the SiNPle variant calling program (version 0.8)
 (c) 2017-2019 Luca Ferretti, <luca.ferretti@gmail.com>
 (c) 2017-2019 Chandana Tennakoon, <drcyber@gmail.com>
 (c) 2017-2021 Paolo Ribeca, <paolo.ribeca@gmail.com>
```

followed by detailed information. The general form(s) the command can be used is:

```
SiNPle [OPTIONS]
```

Algorithmic parameters

Option	Argument(s)	Effect	Note(s)
`-t` `--theta`	*<non_negative_float>*	prior estimate of nucleotide diversity	default=*0.001*
`-T` `--theta-indel`	*<non_negative_float>*	prior estimate of indel likelihood	default=*0.0001*
`-I` `--quality-indel-short`	*<non_negative_integer>*	prior Phred-scaled quality for indels of length 1	default=*35*
`-L` `--quality-indel-long`	*<non_negative_integer>*	prior Phred-scaled quality for indels of length >1	default=*45*
`-p` `--pcr-error-rate`	*<non_negative_float>*	prior estimate of error rate for PCR-generated substitutions	default=*5e-07*
`-P` `--pcr-error-rate-indel`	*<non_negative_float>*	prior estimate of error rate for PCR-generated indels	default=*5e-08*
`--error-rate-substitution`	*<non_negative_float>*	prior estimate of error rate for sequencing-generated substitutions	default=*0.0001*
`--error-rate-indel-short`	*<non_negative_float>*	prior estimate of error rate for sequencing-generated indels of length 1	default=*1e-05*

Option	Argument(s)	Effect	Note(s)
`--error-rate-indel-long`	*<non_negative_float>*	prior estimate of error rate for sequencing-generated indels of length >1	default=*1e-06*

`-s`
`-S`
`--strandedness`
	forward|reverse|both	strands to be taken into account for counts	default=*both*

Input/Output

Option	Argument(s)	Effect	Note(s)
`-i` `--input`	*<input_file>*	name of input file (in mpileup format)	default=*<stdin>*
`-o` `--output`	*<output_file>*	name of output file	default=*<stdout>*

Miscellaneous

Option	Argument(s)	Effect	Note(s)
`-v` `--version`		print version and exit	
`-h` `--help`		print syntax and exit	