kevlar Documentation

Release 0.7+15.gebabd62

Daniel Standage

Nov 22, 2019

Contents

The **kevlar** library is a testbed for developing reference-free variant discovery methods for genomics. The initial focus of the project has been discovery of *de novo* germline variants in simplex pedigrees, but we are laying the groundwork to support a wider range of experimental designs such as case/control studies.

We advertise kevlar as a “reference-free” method in that it does not require a reference genome to identify variant-spanning reads or to assemble these reads into contigs. However, the reference genome is currently still used for making the final variant call by aligning each assembled contig to a small cutout of the reference genome. One of the aspirations of the project moving forward is to reduce and eventually eliminate dependence on a reference genome completely.

kevlar is currently under heavy development, and internal features are not yet stable. However, the core features and workflows are reasonably well tested, and leverage software components from various third-party libraries that are very well tested and widely used.

CHAPTER 1

Installing **kevlar**

1.1 For the impatient

If this isn't your first time in the wing, the following commands should be sufficient to install kevlar in the majority of cases. Otherwise, we suggest reading through the entire installation instructions before beginning.

```
pip3 install pysam networkx pandas scipy intervaltree git+https://github.com/dib-lab/
↪khmer.git
pip3 install biokevlar
```

1.2 Virtual environment

We recommend installing kevlar and its dependencies in a dedicated virtual environment using venv or conda.

- If you use `venv`, the command `python3 -m venv kevlar-env` will create a new virtual environment, and only needs to be executed once. The command `source kevlar-env/bin/activate` will need to be re-executed any time you open a new session in your terminal.
- If you use `conda`, the command `conda create --name kevlar-env python=3.6` will create a new virtual environment, and only needs to be executed once. The command `source activate kevlar-env` will need to be re-executed any time you open a new session in your terminal.

Note: The label `kevlar-env` can be replaced with an alternative label if desired.

1.3 Dependencies

The kevlar package **requires Python 3** and has several dependencies that are not in the standard Python libraries.

- the networkx package

- the pysam module
- the pandas library
- the scipy library
- the intervaltree library
- the khmer package

Also, kevlar requires the bwa command to be callable from your `$PATH` environmental variable.

When kevlar is installed from PyPI, most Python dependencies *should* handled automatically. But since kevlar currently relies on an unreleased version of khmer this last dependency must be installed manually.

```
pip3 install git+https://github.com/dib-lab/khmer.git
```

Note: According to PEP 394 a Python 3 distribution should include a `pip3` command for package management, but in some configurations this may not be true. If you've confirmed that Python 3 is installed correctly, you're probably safe using the `pip` command if `pip3` is unavailable.

Note: In some cases pip cannot install all dependencies automatically, and so manual installation is required.

```
pip3 install pysam>=0.11.2 networkx>=2.0 pandas scipy git+https://github.com/dib-lab/
↪khmer.git
```

1.4 Installation

Once the prerequisites are installed, kevlar can be installed with the pip.

```
pip3 install biokevlar
```

This installs the most recent stable release. If you want to install the latest (possibly unstable) version, pip can install kevlar directly from GitHub.

```
pip3 install git+https://github.com/dib-lab/kevlar.git
```

To test whether kevlar is installed and running correctly, use pytest.

```
pip3 install pytest
pytest --pyargs kevlar.tests
```

1.5 Development environment

If you'd like to contribute to kevlar's development or simply poke around, the source code can be cloned from Github. In addition to the dependencies listed above, a few additional dependencies are required for a complete development environment. These can be installed with `make` for your convenience.

```
git clone https://github.com/dib-lab/kevlar.git
cd kevlar
make devenv
pip3 install -e .
```

Hack away! Feel free to ask questions or submit bug reports to the kevlar issue tracker.

CHAPTER 2

Running **kevlar**

The kevlar library can be invoked via a command-line interface or a Python API.

2.1 Command line interface

Once installed, the kevlar software can be invoked from the shell using the `kevlar` command. The kevlar command line interface (CLI) uses the *subcommand* pattern, in which a single master command supports several different operations by defining multiple subcommands (such as `kevlar novel` and `kevlar partition`). Invoke `kevlar -h` or `kevlar --help` to see a list of all available subcommands, and invoke `kevlar <subcommand> -h` to see detailed instructions for that subcommand. Comprehensive documentation of the kevlar CLI is also available *here*.

Starting with version 1.0, the CLI will be under semantic versioning.

2.2 Python interface

The main procedure of each kevlar subcommand is implemented as a generator function and can be executed programmatically. The following example shows how a standalone script or notebook would invoke the `kevlar partition` procedure.

```
fh = kevlar.open('novel-reads.augfastq.gz', 'r')
readstream = kevlar.parse_augmented_fastx(fh)
partitioner = kevlar.partition.partition(readstream, maxabund=200)
numreads_per_partition = [len(part) for partid, part in partitioner]
plt.hist(numreads_per_partition, bins=25)
```

It is also possible to mimic the behavior of the CLI with Python code. The following example shows how to execute `kevlar partition` from a standalone script or notebook.

```
arglist = [
    'partition', '--out', 'partitioned-reads.augfastq.gz',
    '--max-abund', '200', 'novel-reads.augfastq.gz'
```

(continues on next page)

(continued from previous page)

```
]
args = kevlar.cli.parser().parse_args(arglist)
kevlar.partition.main(args)
```

Other units of code in the kevlar package may also be useful for standalone Python programs. However, the Python API is not yet slated for semantic versioning and is not as stable or well documented as the CLI. Have fun and knock yourself out, but be prepared for changes in internal behavior in subsequent releases!

CHAPTER 3

Quick start

If you have not already done so, install kevlar using *the following instructions*, and Snakemake version 5.0 or greater.

The simplest way to execute kevlar's entire *de novo* variant discovery workflow is using the provided Snakemake workflow configuration. Processing the example data set below should be able to run on a laptop in less than 5 minutes while consuming less than 200 Mb of RAM. The results (`workdir/calls.scored.sorted.vcf.gz`) should include 5 variant calls: a 300 bp insertion and 4 single-nucleotide variants.

A *more detailed tutorial is available*, and a complete listing of all available configuration options for each kevlar command can be found in *the CLI documentation*, or by executing `kevlar <subcommand> -h` in the terminal.

```
# Download data
curl -L https://osf.io/db82p/download -o mother.fq.gz
curl -L https://osf.io/6vrnz/download -o father.fq.gz
curl -L https://osf.io/wt5h8/download -o proband.fq.gz
curl -L https://osf.io/35wgn/download -o refr.fa.gz
bwa index refr.fa.gz

# Download and format configuration file
curl -L https://osf.io/86adm/download | sed "s:/home/user/Desktop:$(pwd):g" > helium-
↪config.json

# Invoke the workflow
snakemake \
    --snakefile kevlar/workflows/mark-I/Snakefile \
    --configfile helium-config.json --cores 4 --directory workdir -p calls
```

CHAPTER 4

Tutorial

This tutorial will cover material similar to that introduced in the *kevlar quick start*, but with a bit more detail and commentary.

4.1 Test data

The data for this tutorial comes from a simulated 25 Mb genome and can be downloaded anonymously from the Open Science Framework.

```
curl -L https://osf.io/b93gw/download -o mother.fq.gz
curl -L https://osf.io/gs6pt/download -o father.fq.gz
curl -L https://osf.io/np9wh/download -o proband.fq.gz
curl -L https://osf.io/mny56/download -o refr.fa.gz
```

The reference genome must be indexed for BWA searches before proceeding.

```
bwa index refr.fa.gz
```

4.2 Before getting started

For this simple example, all required input data and configuration options are provided. However, when analyzing one's own data, it's important to consider a few points.

- **How much memory is needed to store k-mer counts for each sample?** Kevlar uses a probabilistic data structure to store approximate *k*-mer counts, which saves space over exact data structures but also introduces error. The accuracy of the approximate counts depends on two factors: the amount of memory used to store the counts, and the number of distinct *k*-mers present in the sample. This accuracy is summarized by a statistic called the *false positive rate (FPR)*. The best way to reduce Kevlar's memory requirements while keeping the FPR steady is to reduce the number of *k*-mers that need to be tracked. One way this is done is by creating a **mask** of uninteresting *k*-mers for kevlar to ignore: see below. Another way is to perform error correction on reads prior to *k*-mer counting. The majority of distinct *k*-mers in a sample span a sequencing error and occur only once. Performing error correction on the reads will eliminate many of these errors and drastically reduce the number of distinct *k*-mers present. Note, however, that error correction algorithms sometimes struggle to distinguish sequencing error from true variation, especially when there is low coverage. One can expect a small loss of sensitivity for SNV discovery if error correction is used.

It's also important to note that different amounts of memory can be used for different samples. The *k*-mer counts in the case sample (proband) are recomputed and corrected throughout the workflow, and an initially high FPR can be corrected at a later stage. This is not true for the control samples (parents), where a high FPR will lead to a loss of sensitivity.

We recommend an FPR 0.5 for the case sample and 0.05 for the control samples. For properly masked reads, this requires about **10G-20G per sample for error-corrected reads** and anywhere between **36G-72G per sample for non-error-corrected reads**.

- **Which k-mers should Kevlar mask?** Any *k*-mer present in the reference genome is not a signature of novel mutation, and therefore doesn't need to be tracked. Also, *k*-mers from any known or suspected sources of contamination (adapters, bacterial contaminants, etc.) may present false signatures of novel variation. Kevlar uses a **mask** to ignore such *k*-mers during counting and filtering stages of the workflow. It is recommended that the reference genome and any suspected sources of contamination (such as UniVec) be included in the mask. Consequently, if adapter sequences are in the mask, adapter trimming of reads prior to analysis is not necessary.
- **Which thresholds should I use?** In our experience, `--case-min 5` and `--ctrl-max 1` work well for human genomes sequenced at about 30x coverage. We've found that `--ctrl-max 0` is too strict due to sequencing errors and inflated *k*-mer counts. The `--case-min` threshold can be adjusted for higher or lower coverage, with the usual caveats: the stricter the threshold, the more confidence you have in your results but the more true variants you miss.
- **Which k-mer size should I use?** In our experience, *k=31* works well for both SNV and indel discovery. However, variants occurring in repetitive regions may be undetectable without a larger *k* size (between 45 and 55). Increasing the *k*-mer size will have only a moderate effect on the amount of memory required to store *k*-mer counts. However, note that as *k* increases each *k*-mer is more likely to span a sequencing error, and accordingly error correction will a much larger impact on performance.
- **How can I filter variant calls?** Kevlar implements a variety of filters to distinguish true *de novo* variants from inherited variants and false positive variant calls. The documentation for *kevlar call* and *kevlar simlike* describe these filters. However, it's also common to filter out variant predictions that are common variants (not *de novo*) or that occur in problematic regions such as segmental duplications or simple sequence repeats. The *varfilter* setting accepts a BED file that specifies genomic regions from which to filter out variant predictions. Make sure that the chromosome names match those in the reference genome file!

4.3 The kevlar simplex workflow: Mark I

While each step of the kevlar workflow can be invoked independently using the `kevlar` command, we suggest using the Snakemake workflow provided with the kevlar source code distribution. Instructions for configuring and running the workflow are provided alongside the Snakefile and configuration template. The notes above should be helpful in setting parameter values and selecting appropriate input files.

4.4 Assessing accuracy

The output of the command above is a gzip-compressed VCF file, and in this case should contain 10 variant calls. The `kevlar-eval.sh` script below uses `bedtools intersect` to do a quick and simple evaluation of kevlar's accuracy.

```
curl -L https://osf.io/yj9nu/download -o neon-refr.vcf
curl -L https://osf.io/nz5vt/download -o kevlar-eval.sh
bash kevlar-eval.sh neon-refr.vcf calls.scored.sorted.vcf.gz
```

4.5 The kevlar simplex workflow in detail

- **[Step 0: count k-mers]** The *kevlar count* command is used to count *k*-mers for each sample, as well as for the reference genome and the mask.
- **[Step 1: find interesting k-mers]** The *kevlar novel* command uses pre-computed *k*-mer counts to find reads containing novel *k*-mers using the specified thresholds.
- **[Step 2: filter k-mers and reads]** The *kevlar filter* command recomputes *k*-mer counts and filters out *k*-mers with insufficient abundance or *k*-mers from contaminant sources. Any reads that no longer have any interesting *k*-mers after filtering are discarded.
- **[Step 3: partition reads]** Reads spanning the same variant will typically share numerous interesting *k*-mers. The *kevlar partition* command groups reads based on shared novel *k*-mers.
- **[Step 4: contig assembly]** The *kevlar assemble* command assembles each partition of reads into contigs for variant annotation.
- **[Step 5: localize reference targets]** The *kevlar localize* command identifies the appropriate target (or set of targets) in the reference genome for aligning each variant-spanning contig for variant annotation.
- **[Step 6: call variants]** The *kevlar call* command computes a full dynamic programming alignment of each reference-spanning contig to its corresponding reference target(s) and calls variants based on the alignment path.
- **[Step 7: score and rank variant calls]** The *kevlar simlike* command computes a likelihood score for each variant prediction and ranks variant calls based on this score.

CHAPTER 5

Terminology

Here is a brief glossary of terms used throughout the code and project.

- **novel k-mer**: a *k*-mer linked (or putatively linked) to a novel germline variant
- **ikmer**: short for *interesting k-mer*, a synonym for **novel k-mer** with an emphasis on its unverified status
- **partition**: a set of reads that share novel *k*-mers and are thus (putatively) associated with the same variant; sometimes this is abbreviated in the code using cc or CC, referring to the fact that these partitions are reflected as *connected components* in the *shared novel* **k*-mers read graph; other abbreviations in the code include PART and CALLCLASS
- **augfastx**: sequences in Fasta or Fastq format, augmented with annotations indicating the position and abundance of interesting *k*-mers and mate sequences; the `kevlar.parse_augmented_fastx` and `kevlar.print_augmented_fastx` commands can be used to read and write data in augmented Fasta or Fastq format
- **contig**: in the context of kevlar, a contig almost always refers to a sequence assembled from a set of reads sharing novel *k*-mers and thus (putatively) spanning the same novel variant
- **reference cutout**: the algorithm kevlar uses to align contigs and call variants is not designed to map a short contig to a long chromosome sequence; therefore kevlar computes a "reference target sequence" or a "cutout" of the genome to which each contig is aligned for variant calling

CHAPTER 6

File formats in **kevlar**

Although kevlar performs many operations on *k*-mers, read sequences are the primary currency of exchange between different stages of the analysis workflow. kevlar supports reading from and writing to Fasta and Fastq files, and treats these identically since it does not use any base call quality information. In most cases, kevlar should also be able to automatically detect whether an input file is gzip-compressed or not and handle it accordingly (no bzip2 support).

6.1 Broken-paired Fasta / Fastq files

While kevlar does not require pairing information for variant discovery, it can be helpful in the final stages of variant calling. The bioinformatics community uses two common conventions for encoding pairing information: paired files and interleaved files. In paired files, the first record in file1 is paired with the first record in file2, the second record in file1 is paired with the second record in file2, and so on. In an interleaved file, the first record is paired with the second record, the third record is paired with the fourth record, and so on.

If you want to retain and use pairing information, kevlar only supports reading pairing information from interleaved files. kevlar also supports "broken paired" files, where single-end/orphaned reads are occasionally scattered in between paired reads in an interleaved file.

6.2 Augmented sequences

"Interesing *k*-mers" are putatively novel *k*-mers that are high abundance in the proband/case sample(s) and effectively absent from control samples. To facilitate reading and writing these "interesting *k*-mers" along with the reads to which they belong, kevlar uses an *augmented* version of the Fasta and Fastq formats. Here is an example of an augmented Fastq file.

```
@read1
TTTTACCCGATGGGCGAGGTGAAATACTATGCCGATTTATTCTTACACAATTAAATTGCTAGTCCGGTTAGGGTTAGTTTGCGGCCTTCGTTCCAGCGCCC
+
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
     CCCGATGGGCGAGGTGAAA          18 1 0#
```

(continues on next page)

(continued from previous page)

```
                                                                                ␣
↪AGGGTTAGTTTGCGGCCTT           11 0 0#
@read2
AAGAGATTGTCGCTTGCCCCGTAAAGGAATTAGACCGGGCGACCAGAGCCTATTAGTAGCCCGCGCCTGTAGCACAAACGACTTTCGTACTATTATTAGA
+
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
 AGAGATTGTCGCTTGCCCC           14 0 1#
  GAGATTGTCGCTTGCCCCG           12 0 0#
   AGATTGTCGCTTGCCCCGT           14 0 0#
@read3
GAGACCATAAACCAGCTCTTGGTACCGAAAGAACACCTATGAATAACCGTGAGTGCATGATTCCTGTGAAGAGATTGTCGCTTGCCCCGTAAAGGAATTA
+
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
               CTCTTGGTACCGAAAGAAC           19 1 0#
                                                                                ␣
↪AGAGATTGTCGCTTGCCCC           14 0 1#
                                                                                 ␣
↪GAGATTGTCGCTTGCCCCG           12 0 0#
                                                                                  ␣
↪AGATTGTCGCTTGCCCCGT           14 0 0#
@read4
TCCGGTTAGGGTTAGTTTGCGGCCTTCGTTCCAGCGCCGTGTTGTTGCAATTTAATCCCGAGAAACCTCATGTAGCGGCTACTGGACCGCTGGGTAAGCT
+
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
       AGGGTTAGTTTGCGGCCTT           11 0 0#
```

As with a normal Fastq file, each record contains 4 lines to declare the read sequence and qualities. However, these 4 lines are followed by one or more lines indicating the "interesing *k*-mers", showing their sequence followed by their abundance in each sample (case first, then controls), with a `#` as the final character. Augmented Fastq files are easily converted to normal Fastq files by invoking a command like `grep -v '#$' reads.augfastq > reads.fastq` (same for augmented Fasta files).

The functions `kevlar.parse_augmented_fastx` and `kevlar.print_augmented_fastx` are used internally to read and write augmented Fastq/Fasta files. However, these functions can easily be imported and called from third-party Python scripts as well.

6.3 Mate sequences (deprecated)

Note: Mate sequences are no longer used by kevlar internally. Augmented Fastq files containing mate sequences will still be processed correctly, but they are no longer created by kevlar.

Although kevlar does not require pairing information, it can be used to improve calling when it's available. The augmented Fastq/Fasta format also allows mate sequences to be associated with each record. If a contig assembled from novel reads maps to multiple regions of the reference genome with the same score, this pairing information can be used to predict the most likely true variant.

The `mateseq` annotation should be placed after the first 4 lines of the record, and shown in the two records below.

```
@DraconisOccidentalisRead12/1
CAGGTAGTGTGATGCCTCCAGCTTTGTTCTTTTGGCTTAGGATTGACTTGGCAATTCGGGCTCTTTTTGGTTCCATATGAACTTTAAAGTAGTTTTTTC
+
888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888
```

(continues on next page)

(continued from previous page)

```
                                    TTCTTTTGGCTTAGGATTGACTTGGCAATTC          7 0 1#
                                     TCTTTTGGCTTAGGATTGACTTGGCAATTCG          5 0 1#
                                      CTTTTGGCTTAGGATTGACTTGGCAATTCGG          5 0 1#
                                       TTTTGGCTTAGGATTGACTTGGCAATTCGGG          5 1 1#
                                        TTTGGCTTAGGATTGACTTGGCAATTCGGGC          5 0 1#

↪#mateseq=CTGATAAGCAACTTCAGCAAAGTCTCAGGATACAAAATCAATGTACGAAAATCACAAGCGTTCTTATACACCAACAACAGACAAACAGAC
↪#
@DraconisOccidentalisRead56/1
TCTTGAATTCCCATGTGTTGTGGGAGGGACCCATTGGGAGGTAATTGAATCATGGGGGCACGTCTTTCCCATGCTGTTCTCATGATAGAGACTAAGTCTC
+
8888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888
     AATTCCCATGTGTTGTGGGAGGGACCCATTG          5 0 1#
      ATTCCCATGTGTTGTGGGAGGGACCCATTGG          5 0 1#

↪#mateseq=ATTAGAAAAAAAAAGTGCATTCGTAAATGTCATAACAATAAAATTATACTCCAAGACTTTGTACAAGATGAAAGTAATATGAAGAAGGGC
↪#
       TTCCCATGTGTTGTGGGAGGGACCCATTGGG          5 0 1#
        TCCCATGTGTTGTGGGAGGGACCCATTGGGA          5 0 1#
```

CHAPTER 7

K-mer banding

If memory is a limiting factor for *k*-mer counting, kevlar supports a scatter/gather approach to based on a strategy we call "*k*-mer banding." In brief, kevlar can achieve an N-fold reduction in memory usage in exchange for counting *k*-mers N batches. For each batch, kevlar ignores all *k*-mers except those whose hash values fall within a specified numerical range (band), reducing the memory required to achieve accurate *k*-mer counts.

The *kevlar count* and *kevlar novel* commands support *k*-mer banding, and the *kevlar unband* command merges novel reads from multiple batches into a single read set suitable for downstream analysis.

The example below is not intended to show the most efficient way of invoking the banding workflow, simply how it is intended to work.

```
# Count k-mers in 6 passes for a 6x reduction in memory
for $band in {1..6}
do
    for indiv in mother father proband
    do
        kevlar count \
            --threads 16 --memory 8G --counter-size 8 --ksize 31 \
            --mask refr-univec-mask.nodetable \
            --num-bands 6 --band $band \
            ${indiv}.kmer-counts.band${band}.counttable ${indiv}.reads.fastq.gz
    done
done

# Find novel k-mers in 6 passes
for $band in {1..6}
do
    kevlar novel \
        --case proband.reads.fastq.gz --case-counts ${indiv}.kmer-counts.band${band}.
↪counttable \
        --control-counts mother.kmer-counts.band${band}.counttable mother.kmer-counts.
↪band${band}.counttable \
        --case-min 5 --ctrl-max 1 --ksize 31\
        --num-bands 6 --band $band \
        --out proband-novel-${band}.augfastq.gz
```

(continues on next page)

(continued from previous page)

```
done

# Combine the novel k-mers from all 6 passes
kevlar unband --out proband-novel.augfastq.gz proband-novel-{1..6}.augfastq.gz

# The rest of the kelvar simplex workflow:
#    - kevlar filter
#    - kevlar partition
#    - kevlar kevlar assemble
#    - kevlar localize
#    - kevlar call
#    - kevlar simlike
```

CHAPTER 8

Comprehensive command-line interface reference

The kevlar command-line interface is designed around a single command `kevlar`. From this one command, a variety of tasks and procedures can be invoked using several *subcommands*.

Once kevlar is installed, available subcommands can be listed by executing `kevlar -h`. To see instructions for running a specific subcommand, execute `kevlar <subcommand> -h` (of course replacing `subcommand` with the actual name of the subcommand).

8.1 kevlar count

Compute k-mer abundances for the provided sample. Supports k-mer banding: see http://kevlar.readthedocs.io/en/latest/banding.html for more details.

```
usage: kevlar count [-h] [-k K] [-c C] [-M MEM] [--max-fpr FPR] [--mask MSK]
                    [--count-masked] [--num-bands N] [--band I] [-t T]
                    counttable seqfile [seqfile ...]
```

8.1.1 Positional Arguments

counttable	name of the file to which the output (a k-mer count table) will be written; the suffix ".counttable" will be applied if the provided file name does not end in ".ct" or ".counttable"
seqfile	input files in Fastq/Fasta format

8.1.2 Named Arguments

-k, --ksize	k-mer size; default is 31
-c, --counter-size	Possible choices: 1, 4, 8

	number of bits to allocate for counting each k-mer; options are 1 (max count: 1), 4 (max count: 15), and 8 (max count: 255); default is 8
-M, --memory	memory to allocate for the count table
--max-fpr	terminate if the estimated false positive rate for any sample is higher than "FPR"; default is 0.2
--mask	counttable or nodetable of k-mers to ignore when counting k-mers
--count-masked	by default, when a mask is provided k-mers in the mask are ignored; this setting inverts the behavior so that only k-mers in the mask are counted
--num-bands	number of bands into which to divide the hashed k-mer space
--band	a number between 1 and N (inclusive) indicating the band to be processed
-t, --threads	number of threads to use for file processing; default is 1

Example:

```
kevlar count --memory 500M case1.ct case1-reads.fastq
```

Example:

```
kevlar count --ksize 25 --memory 12G --max-fpr 0.01 --threads 8 \
    proband.counttable \
    proband-R1.fq.gz proband-R2.fq.gz proband-unpaired.fq.gz
```

8.2 kevlar novel

Identify "interesting" (potentially novel) k-mers and output the corresponding reads. Here we define "interesting" k-mers as those which are high abundance in each case sample and effectively absent (below some specified abundance threshold) in each control sample.

```
usage: kevlar novel --case F [F ...] [--case-counts F [F ...]]
                    [--control F [F ...]] [--control-counts F [F ...]] [-x X]
                    [-y Y] [-M MEM] [--max-fpr FPR] [--num-bands N] [--band I]
                    [-o FILE] [--save-case-counts CT [CT ...]]
                    [--save-ctrl-counts CT [CT ...]] [-h] [-k K]
                    [--abund-screen INT] [-t T] [--skip-until ID]
```

8.2.1 Case/control config

Specify input files, as well as thresholds for selecting "interesting" k-mers. A single pass is made over input files for control samples (to compute k-mer abundances), while two passes are made over input files for case samples (to compute k-mer abundances, and then to identify "interesting" k-mers). The k-mer abundance computing steps can be skipped if pre-computed k-mer abunandances are provided using the "–case-counts" and/or "–control-counts" settings. If "–control-counts" is declared, then all "–control" flags are ignored. If "–case-counts" is declared, FASTA/FASTQ files must still be provided with "–case" for selecting "interesting" k-mers and reads.

--case	one or more FASTA/FASTQ files containing reads from a case sample; can be declared multiple times corresponding to multiple case samples, see examples below

--case-counts	counttable file(s) corresponding to each case sample; if not provided, k-mer abundances will be computed from FASTA/FASTQ input; only one counttable per sample, see examples below
--control	one or more FASTA/FASTQ files containing reads from a control sample; can be declared multiple times corresponding to multiple control samples, see examples below
--control-counts	counttable file(s) corresponding to each control sample; if not provided, k-mer abundances will be computed from FASTA/FASTQ input; only one counttable per sample, see examples below
-x, --ctrl-max	k-mers with abund > X in any control sample are uninteresting; default is X=1
-y, --case-min	k-mers with abund < Y in any case sample are uninteresting; default is Y=6
-M, --memory	total memory allocated to k-mer abundance for each sample; default is 1M; ignored when pre-computed k-mer abundances are supplied via counttable
--max-fpr	terminate if the expected false positive rate for any sample is higher than the specified FPR; default is 0.2

8.2.2 K-mer banding

If memory is a limiting factor, it is possible to get a linear decrease in memory consumption by running *kevlar novel* in “banded” mode. Splitting the hashed k-mer space into N bands and only considering k-mers from one band at a time reduces the memory consumption to approximately 1/N of the total memory required. This implements a scatter/gather approach in which *kevlar novel* is run N times, after the results are combined using *kevlar filter*.

--num-bands	number of bands into which to divide the hashed k-mer space
--band	a number between 1 and N (inclusive) indicating the band to be processed

8.2.3 Output settings

-o, --out	file to which interesting reads will be written; default is terminal (stdout)
--save-case-counts	save the computed k-mer counts for each case sample to the specified count table file(s)
--save-ctrl-counts	save the computed k-mer counts for each control sample to the specified count table file(s)

8.2.4 Miscellaneous settings

-k, --ksize	k-mer size; default is 31
--abund-screen	discard reads with any k-mers whose abundance is < INT
-t, --threads	number of threads to use for file processing; default is 1
--skip-until	when re-running *kevlar novel*, skip all reads in the case input until read with name *ID* is observed

Example:

```
kevlar novel --out novel-reads.augfastq --case proband-reads.fq.gz \
    --control father-reads-r1.fq.gz father-reads-r2.fq.gz \
    --control mother-reads.fq.gz
```

Example:

```
kevlar novel --out novel-reads.augfastq.gz \
    --control-counts father.counttable mother.counttable \
    --case-counts proband.counttable --case proband-reads.fastq \
    --ctrl-max 0 --case-min 10 --ksize 27
```

Example:

```
kevlar novel --out output.augfastq \
    --case proband1.fq --case proband2.fq \
    --control control1a.fq control1b.fq \
    --control control2a.fq control2b.fq \
    --save-case-counts p1.ct p2.ct --save-ctrl-counts c1.ct c2.ct
```

8.3 kevlar filter

Discard k-mers and reads that are contaminant in origin or whose abundances were inflated during the preliminary k-mer counting stage.

```
usage: kevlar filter [-h] [-M MEM] [--max-fpr FPR] [--mask MSK] [-x X] [-y Y]
                     [-o FILE]
                     augfastq
```

8.3.1 Positional Arguments

augfastq	putatively novel reads in augmented Fastq format

8.3.2 Named Arguments

-M, --memory	memory to allocate for the k-mer re-counting
--max-fpr	terminate early if the estimated false positive rate for re-computed k-mer abundances is higher than "FPR"; default is 0.01
--mask	counttable or nodetable of k-mers to ignore when re-counting k-mers
-x, --ctrl-max	k-mers with abund > X in any control sample are uninteresting; default is X=1
-y, --case-min	k-mers with abund < Y in any case sample are uninteresting; default is Y=6
-o, --out	output file; default is terminal (stdout)

8.4 kevlar partition

Construct a graph to group reads by shared interesting k-mers. Nodes in the graph represent reads, and edges between a pair of nodes indicate that the two corresponding reads have one or more interesting k-mers in common. Connected components in the undirected graph correspond to distinct variants (or variant-related breakpoints).

```
usage: kevlar partition [-h] [-s] [--min-abund X] [--max-abund Y] [--no-dedup]
                        [--gml FILE] [--split OUTPREFIX] [-o FILE]
                        infile
```

8.4.1 Positional Arguments

infile	input reads in augmented Fast[q|a] format

8.4.2 Named Arguments

-s, --strict	require perfect identity between overlapping reads for inclusion in the same partition; by default, only a shared interesting k-mer is required
--min-abund	ignore k-mers with abundance lower than X; default is 2
--max-abund	ignore k-mers with abundance higher than Y; default is 200
--no-dedup	skip step to remove duplicates
--gml	write read graph to .gml file
--split	write each partition to a separate output file, each with a filename like "OUTPREFIX.cc#.augfastq.gz"
-o, --out	output file; default is terminal (stdout)

8.5 kevlar assemble

Assemble reads into contigs representing putative variants

```
usage: kevlar assemble [-h] [-p ID] [--max-reads N] [-o FILE] augfastq
```

8.5.1 Positional Arguments

augfastq	annotated reads in augmented Fastq/Fasta format

8.5.2 Named Arguments

-p, --part-id	only assemble partition "ID" in the input
--max-reads	do not attempt to assemble any partitions with more than N reads (default: 10000)
-o, --out	output file; default is terminal (stdout)

8.6 kevlar localize

For each partition, compute the reference target sequence to use for variant calling. NOTE: this command relies on the *bwa* program being in the PATH environmental variable.

```
usage: kevlar localize [-h] [-d Δ] [-p ID] [-o FILE] [-z Z] [-x X]
                       [--include REGEX] [--exclude REGEX]
                       refr contigs [contigs ...]
```

8.6.1 Positional Arguments

refr	BWA indexed reference genome
contigs	assembled reads in augmented Fasta format

8.6.2 Named Arguments

-d, --delta	retrieve the genomic interval from the reference by extending beyond the span of all k-mer starting positions by Δ bp
-p, --part-id	only localize partition "ID" in the input
-o, --out	output file; default is terminal (stdout)
-z, --seed-size	seed size; default is 51
-x, --max-diff	split and report multiple reference targets if the distance between two seed matches is > X; by default, X is set dynamically for each partition and is equal to 3 times the length of the longest contig in the partition; each resulting bin specifies a reference target sequence to which assembled contigs will subsequently be aligned
--include	discard alignments to any chromosomes whose sequence IDs do not match the given pattern
--exclude	discard alignments to any chromosomes whose sequence IDs match the given pattern

8.7 kevlar call

Align variant-related reads to the reference genome and call the variant from the alignment.

```
usage: kevlar call [-A A] [-B B] [-O O] [-E E] [--gen-mask FILE]
                   [--mask-mem MEM] [--mask-max-fpr FPR] [-h] [-d]
                   [--no-homopoly-filter] [--max-target-length L]
                   [--refr FILE] [-o FILE] [-k K]
                   queryseq targetseq
```

8.7.1 Required inputs

queryseq	contigs assembled by "kevlar assemble"
targetseq	region of reference genome identified by "kevlar localize"

8.7.2 Alignment scoring

-A, --match	match score; default is 1
-B, --mismatch	mismatch penalty; default is 2
-O, --open	gap open penalty; default is 5
-E, --extend	gap extension penalty; default is 0

8.7.3 Mask generation settings

--gen-mask	generate a nodetable containing all k-mers that span any variant call
--mask-mem	memory to allocate for the node table
--mask-max-fpr	terminate if the estimated false positive rate is higher than “FPR”; default is 0.01

8.7.4 Miscellaneous settings

-d, --debug	show debugging output
--no-homopoly-filter	by default, short indels adjacent to homopolymers are filtered out; use this flag to disable that filter
--max-target-length	do not attempt to call variants if the target genomic sequence is > L bp; by default, L=10000
--refr	reference genome indexed for BWA search; if provided, mates of interesting reads will be used to diambiguate multi-mapping contigs
-o, --out	output file; default is terminal (stdout)
-k, --ksize	k-mer size; default is 31

8.8 kevlar simlike

Sort variants by likelihood score

```
usage: kevlar simlike --case CT --controls CT [CT ...] --refr REFR
                      [--ctrl-max X] [--case-min Y] [--mu μ] [--sigma σ]
                      [--epsilon ϵ] [--ctrl-abund-high H] [--case-abund-low L]
                      [--min-like-score S] [--drop-outliers]
                      [--ambig-thresh A] [-h] [--sample-labels LBL [LBL ...]]
                      [-f] [-o OUT]
                      vcf [vcf ...]
```

8.8.1 Positional Arguments

vcf

8.8.2 K-mer count files

Likelihood scores are based on the abundance of alternate allele k-mers in each sample and on the abundance of reference allele k-mers in the reference genome.

--case	k-mer counttable for case/proband
--controls	k-mer counttables for controls/parents/siblings, 1 counttable per sample
--refr	k-mer smallcounttable for reference genome

8.8.3 K-mer count thresholds

The thresholds originally used to detect novel k-mers are used at this stage to distinguish true variants from spurious predictions.

--ctrl-max	maximum abundance threshold for controls; default is 1
--case-min	minimum abundance threshold for proband; default is 6

8.8.4 K-mer coverage

Likelihood scores also depend on the estimated or observed distribution of k-mer abundances in each sample.

--mu	mean k-mer abundance; default is 30.0
--sigma	standard deviation of k-mer abundance; default is 8.0
--epsilon	error rate; default is 0.001

8.8.5 Heuristic filters

The following heuristic filters can improve accuracy when calling de novo variants but may require tuning for your particular data set.

--ctrl-abund-high	a variant call will be filtered out if either of the control samples has >H high abundance k-mers spanning the variant (where high abundance means > –ctrl-max); by default, H=4; set H<=0 to disable the filter
--case-abund-low	a variant call will be filtered out if the case sample has L or more consecutive low abundance k-mers spanning the variant (where low abundance means < –case-min); by default, L=5; set L<=0 to disable the filter
--min-like-score	filter out variant predictions with likelihood scores < S; by default, S = 0.0, but it's often possible to improve specificity without sacrificing sensitivity by raising S to, for example, 50.0
--drop-outliers	discard terminal variant-spanning k-mers with abunance much higher than average (representing k-mers that should be in the reference genome but are not); this will increase sensitivity, but will potentially introduce many false calls as well
--ambig-thresh	discard contigs that result in > A distinct, equally optimal variant calls; by default, A = 10; set A=0 to disable this filter

8.8.6 Miscellaneous settings

--sample-labels	list of sample labels (with case/proband first)
-f, --fast-mode	whenever possible, stop computations prematurely for any putative variants that have already been filtered out
-o, --out	output file; default is terminal (standard output)

8.9 kevlar alac

Assemble partitioned reads, localize to the reference genome, align the assembled contig to the reference, and call variants.

```
usage: kevlar alac [-p ID] [--max-reads N] [-z Z] [-d D] [-x X]
                   [--include REGEX] [--exclude REGEX] [--max-target-length L]
                   [-A A] [-B B] [-O O] [-E E] [--gen-mask FILE]
                   [--mask-mem MEM] [--mask-max-fpr FPR] [-h] [-o FILE] [-i I]
                   [-k K] [-t T]
                   infile refr
```

8.9.1 Required inputs

infile	partitioned reads in augmented Fastq format
refr	reference genome in Fasta format (indexed for bwa search)

8.9.2 Read assembly

-p, --part-id	only process partition "ID" in the input
--max-reads	do not attempt to assemble any partitions with more than N reads (default: 10000)

8.9.3 Target extraction

-z, --seed-size	seed size; default is 51
-d, --delta	retrieve the genomic interval from the reference by extending beyond the span of all k-mer starting positions by D bp
-x, --max-diff	split and report multiple reference targets if the distance between two seed matches is > X; by default, X is set dynamically for each partition and is equal to 3 times the length of the longest contig in the partition; each resulting bin specifies a reference target sequence to which assembled contigs will subsequently be aligned
--include	discard alignments to any chromosomes whose sequence IDs do not match the given pattern
--exclude	discard alignments to any chromosomes whose sequence IDs match the given pattern
--max-target-length	do not attempt to call variants if the target genomic sequence is > L bp; by default, L=10000

8.9.4 Alignment scoring

-A, --match	match score; default is 1
-B, --mismatch	mismatch penalty; default is 2
-O, --open	gap open penalty; default is 5
-E, --extend	gap extension penalty; default is 0

8.9.5 Mask generation settings

--gen-mask	generate a nodetable containing all k-mers that span any variant call
--mask-mem	memory to allocate for the node table
--mask-max-fpr	terminate if the estimated false positive rate is higher than "FPR"; default is 0.01

8.9.6 Miscellaneous settings

-o, --out	output file; default is terminal (stdout)
-i, --min-ikmers	do not report calls that a supported by fewer than *I* interesting k-mers
-k, --ksize	k-mer size; default is 31
-t, --threads	process T partitions at a time using T threads

8.10 kevlar unband

When kevlar is run in k-mer banding mode, the same read will typically appear in multiple output files, annotated with a different set of potentially novel k-mers in each case. This command will consolidate these duplicated records across files into a single non-redundant set of reads with the complete set of novel k-mer annotations.

```
usage: kevlar unband [-h] [-n N] [-o FILE] infile [infile ...]
```

8.10.1 Positional Arguments

infile	input files in augmented Fasta/Fastq format

8.10.2 Named Arguments

-n, --n-batches	number of batches into which records will be split; default is 16; N temporary files are created and each record from the input is written to a temporary file based on its read name; then each batch is loaded into memory and duplicate records are resolved
-o, --out	output file; default is terminal (stdout)

8.11 kevlar augment

Internally, kevlar annotates sequences with “interesting k-mers” and uses “augmented” Fastq and Fasta formats. Processing sequences with third-part tools usually requires discarding these annotations. This command is used to augment/reaugment a set of sequences using annotations from an already augmented sequence file.

```
usage: kevlar augment [-h] [-o FILE] augseqs seqs
```

8.11.1 Positional Arguments

augseqs	augmented sequence file
seqs	sequences to annotate

8.11.2 Named Arguments

-o, --out	output file; default is terminal (stdout)

8.12 kevlar mutate

Apply the specified mutations to the genome provided.

```
usage: kevlar mutate [-h] [-o FILE] mutations genome
```

8.12.1 Positional Arguments

mutations	mutations file
genome	genome to mutate

8.12.2 Named Arguments

-o, --out	output file; default is terminal (stdout)

CHAPTER 9

Contributor Code of Conduct

As contributors and maintainers of this project, we pledge to respect all people who contribute through reporting issues, posting feature requests, updating documentation, submitting pull requests or patches, and other activities.

We are committed to making participation in this project a harassment-free experience for everyone, regardless of level of experience, gender, gender identity and expression, sexual orientation, disability, personal appearance, body size, race, age, or religion.

Examples of unacceptable behavior by participants include the use of sexual language or imagery, derogatory comments or personal attacks, trolling, public or private harassment, insults, or other unprofessional conduct.

Project maintainers have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct. Project maintainers or contributors who do not follow the Code of Conduct may be blocked from further contribution and engagement with the project.

This Code of Conduct is adapted from the Contributor Covenant, version 1.0.0, available at http://contributor-covenant.org/version/1/0/0/.

CHAPTER 10

Links

- Github repository
- License