Jellyfish 2 User Guide

December 13, 2013

Contents

Chapter 1

Getting started

1.1 Counting all k-mers

The basic command to count all k-mers is as follows:

```
jellyfish count -m 21 -s 100M -t 10 -C reads.fasta
```

This will count canonical (`-C`) 21-mers (`-m` 21), using a hash with 100 million elements (`-s` 100 M) and 10 threads (`-t` 10) in the sequences in the file reads.fasta. The output is written in the file 'mer_counts.jf' by default (change with `-o` switch).

To compute the histogram of the k-mer occurrences, use the histo subcommand (see section 3.1):

```
jellyfish histo mer_counts.jf
```

To query the counts of a particular k-mer, use the query subcommand (see section 3.3):

```
jellyfish query mer_counts.jf AACGTTG
```

To output all the counts for all the k-mers in the file, use the dump subcommand (see section 3.2):

```
jellyfish dump mer_counts.jf > mer_counts_dumps.fa
```

To get some information on how, when and where this jellyfish file was generated, use the info subcommand (see section 3.4):

```
jellyfish info mer_counts.jf
```

For more detail information, see the relevant sections in this document. All commands understand `--help` and will produce some information about the switches available.

1.1.1 Counting k-mers in sequencing reads

In sequencing reads, it is unknown which strands of the DNA is sequenced. As a consequence, a k-mer or its reverse complement are essentially equivalent. The canonical representative of a k-mer m is by definition m or the reverse complement of m, whichever comes first lexicographically. The `-C` switch instructs to save in the hash only canonical k-mers, while the count is the number of occurrences of both a k-mer and it reverse complement.

The size parameter (given with `-s`) is an indication of the number k-mers that will be stored in the hash. For sequencing reads, one this size should be the size of the genome plus the k-mers generated by sequencing errors. For example, if the error rate is e (e.g. Illumina reads, usually $e \approx 1\%$), with an estimated genome size of G and a coverage of c, the number of expected k-mers is $G + Gcek$. This assume

> NOTE: unlike in Jellyfish 1, this `-s` parameter is only an estimation. If the size given is too small to fit all the k-mers, the hash size will be increased automatically or partial results will be written to disk and finally merged automatically. Running 'jellyfish merge' should never be necessary, as now jellyfish now takes care of this task on its own.

If the low frequency k-mers (k-mers occurring only once), which are mostly due to sequencing errors, are not of interest, one might consider counting only high-frequency k-mers (see section 1.2), which uses less memory and is potentially faster.

1.1.2 Counting k-mers in a genome

In an actual genome or finished sequence, a k-mer and its reverse complement are not equivalent, hence using the `-C` switch does not make sense. In addition, the size for the hash can be set directly to the size of the genome.

1.2 Counting high-frequency k-mers

Jellyfish offers two way to count only high-frequency k-mers (meaning only k-mers with count > 1), which reduces significantly the memory usage. Both methods are based on using Bloom filters. The first method is a one pass method, which provides approximate count for some percentage of the k-mers. The second method is a two pass method which provides exact count. In both methods, most of the low-frequency k-mers are not reported.

1.2.1 One pass method

Adding the `--bf-size` switch make jellyfish first insert all k-mers first into a Bloom filter and only insert into the hash the k-mers which have already been seen at least once. The argument to `--bf-size` should the total number of k-mer expected in the data set while the `--size` argument should be the number of k-mers occurring more than once. For example:

```
jellyfish count -m 25 -s 3G --bf-size 100G -t 16 homo_sapiens.fa
```

would be appropriate for counting 25-mers in human reads at 30× coverage. The approximate memory usage is 9 bits per k-mer in the Bloom filter.

The count reported for each k-mer (by 'jellyfish dump' or 'jellyfish query') is one less than the actual count. Meaning, the count 1 k-mer are not reported, count 2 k-mer are reported to have count 1, etc.

The drawback of this method is some percentage of the k-mer that should not be reported (because they occur only once) are reported. This is due to the random nature of the Bloom filter data structure. The percentage is $< 1\%$ by default and can be changed with the`--bf-fp` switch.

1.2.2 Two pass method

In the two pass method, first a Bloom counter is created from the reads with 'jellyfish bc'. Then this Bloom counter is given to the 'jelllyfish count' command and only the k-mers which have been seen twice in the first pass will be inserted in the hash. For example, with a human data set similar that in section 1.2.1:

```
jellyfish bc -m 25 -s 100G -t 16 -o homo_sapiens.bc homo_sapiens.fa
jellyfish count -m 25 -s 3G -t 16 --bc homo_sapiens.bc homo_sapiens.fa
```

The advantage of this method is that the counts reported for the k-mers are all correct. Most count 1 k-mer are not reported, except for a small percentage (set by the `-f` switch of the bc subcommand) of them which are reported (correctly with count 1). All other k-mers are reported with the correct count.

The drawback of this method is that it requires to parse the entire reads data set twice and the memory usage of the Bloom counter is greater than that of the Bloom filter (slightly less than twice as much).

Chapter 2

FAQ

2.1 How to read compressed files (or other format)?newmacroname

Jellyfish only reads FASTA or FASTQ formatted input files. By reading from pipes, jellyfish can read compressed files, like this:

```
zcat *.fastq.gz | jellyfish count /dev/fd/0 ...
```

or by using the '<()' redirection provided by the shell (e.g. bash, zsh):

```
jellyfish count <(zcat file1.fastq.gz) <(zcat file2.fasta.gz) ...
```

2.2 How to read multiple files at once?

Often, jellyfish can parse an input sequence file faster than gzip or fastq-dump (to parse SRA files) can output the sequence. This leads to many threads in jellyfish going partially unused. Jellyfish can be instructed to open multiple file at once. For example, to read two short read archive files simultaneously:

```
jellyfish count -F 2 <(fastq-dump -Z file1.sra) <(fastq-dump -Z file2.sra) ...
```

Another way is to use "generators". First, create a file containing, one per line, commands to generate sequence. Then pass this file to jellyfish and the number of generators to run simultaneously. Jellyfish will spawn subprocesses running the commands passed and read their standard output for sequence. By default, the commands are run using the shell in the SHELL environment variable, and this can be changed by the `-S` switch. Multiple generators will be run simultaneously as specified by the `-G` switch. For example:

```
ls *.fasta.gz | xargs -n 1 echo gunzip -c > generators
jellyfish count -g generators -G 4 ...
```

The first command created the command list into the 'generators' file, each command unzipping one FASTA file in the current directory. The second command runs jellyfish with 4 concurrent generators.

2.3 How to reduce the output size?

The output file was design to be easy to read, but the file generated can be rather large. By default, a 4 bytes counter value is saved for every k-mer (i.e. a maximum count of over 4 billion). Instead, a counter size of 2 bytes or 1 byte can be used with the switch `--out-counter-len`, which reduces significantly the output size.

The count of k-mers which cannot be represented with the given number of bytes will have a value equal to the maximum value that can be represented. Meaning, if the counter field uses 1 byte, any k-mers with count greater or equal to 255 will be reported of having a count 255.

Also, low frequency and high frequency k-mers can be skipped using the `-L` and `-U` switches respectively. Although it might be more appropriate to filter out the low frequency k-mers using Bloom filters, as shown in section 1.2.

Chapter 3

Subcommands

3.1 histo

The histo subcommand outputs the histogram of k-mers frequencies. The last bin, with value one above the high setting set by the `-h` switch (10 000 by default), is a catch all: all k-mers with a count greater than the high setting are tallied in that one bin. If the low setting is set (`-l` switch), then the first bin, with value one below the low setting, is also similarly a catch all.

By default, the bins with a zero count are skipped. This can be changed with the `-f` switch.

3.2 dump

The dump subcommand outputs a list of all the k-mers in the file associated with their count. By default, the output is in FASTA format, where the header line contains the count of the k-mer and the sequence part is the sequence of the k-mer. This format has the advantage that the output contains the sequence of k-mers and can be directly fed into another program expecting the very common FASTA format. A more convenient column format (for human beings) is selected with the `-c` switch.

Low frequency and high frequency k-mers can be skipped with the `-L` and `-U` switches respectively.

In the output of the dump subcommand, the k-mers are sorted according to the hash function used by Jellyfish. The output can be considered to be "fairly pseudo-random". By "fairly" we mean that NO guarantee is made about the actual randomness of this order, it is just good enough for the hash table to work properly. And by "pseudo-random" we mean that the order is actually deterministic: given the same hash function, the output will be always the same and two different files generated with the same hash function can be merged easily.

3.3 query

The query subcommand outputs the k-mers and their counts for some subset of k-mers. It will outputs the counts of all the k-mers passed on the command line or of all the k-mers in the

sequence read from the FASTA or FASTQ formatted file passed to the switch `-s` (this switch can be given multiple times).

3.4 info

The info subcommand outputs some information about the jellyfish file and the command used to generated it, in which directory and at what time the command was run. Hopefully, the information given should be enough to rerun jellyfish under the same conditions and reproduce the output file. In particular, the `-c` switch outputs the command, properly escaped and ready to run in a shell.

The header is saved in JSON format and contains more information than is written by the default. The full header in JSON format can be written out using the -j switch.

3.5 merge

The merge subcommand is a little direct use with version version 2 of jellyfish. When intermediary files were written to disk, because not all k-mers would fit in memory, they can be merged into one file containing the final result with the merge subcommand. The count will merge intermediary files automatically as needed.

3.6 cite

The cite subcommand prints the citation for the jellyfish paper. With the `-b`, it is formatted in Bibtex format. How convenient!