**CountRabbit: A fast and flexible Reference-based read/k-mer counter for NGS Datasets**

***About this documentation:*** *This documentation is designed to serve several purposes:*

*I. Explain what CountRabbit is designed to do and under what circumstances its use might be indicated*

*II. Provide a high level introduction to the structure of a CountRabbit run and its output.*

*III. Provide a narrative quick-start guide*

*IV. Describe the "Packing List" (files included with the distribution)*

*V. Provide an alternative recipe-style "How To" for building CountRabbit command lines (in the style of Build-A-Bear™)*

*VI. Provide a comprehensive listing of command line options*

*VII. Describe some basics of the architecture of the code and algorithms*

*It is not anticipated that anyone would need to read the entire documentation-- rather one might start with the overviews, test to see if the Build-A-Bear™ style recipe and resulting output is at least partially matched to the needs at hand, then delve (as needed) into the detailed options.*

***Happy Counting, The Rabbits***

**I. Overview of CountRabbit**

**You have**

One (or many) datasets from next generation sequencing: "Data"

One (or many) sequence assemblies or references (genes?, genomes?, segments?): "Reference"

**You want**

To generate a table that quantifies matches to each reference in each dataset: "Counts"

**CountRabbit is a tool providing such tables**

**There are many such tools:** CountRabbit has similarities/differences + advantages/disadvantages

CountRabbit has some shared capabilities with

Local alignment-based search tools like BLAST/BLAT

Graph-based alignment tools like Bowtie/BWA

Other K-mer based tools like Kraken/Kallisto

and to many other tools (including some home-grown tools [REVA, HQAlign, Jazz18Heap])

While any of these tools can be used for any task, there are advantages to programs that

Take your data in the form it is already in

Provide output in the form that you can understand and quickly use.

Work quickly and have the options you need to filter/trim/arrange the input and output

Because the goals of different bioinformatic analyses are different, different tools will be optimal

CountRabbit has been built to count read or K-mer matches in workflows we use in the Fire Lab.

CountRabbit may or may not be applicable/optimal for your workflow

**The goal of CountRabbit is to provide a simple and intuitive workflow for generating count tables in high throughput sequencing analysis.**

The simplest workflow:

User provides their "Reference" and "Data" input files (e.g. a gene list and some fastq files)

Program provides a simple table of read or K-mer counts for each reference in each dataset

But different applications will involve different choices for how best to search, count, and analyze

This leads to a bunch of decisions that the user needs to make (command line options)

The variety of contexts for which a count table can be useful is vast, so there are a lot of decisions

How are the inputs (Data and Reference) provided?

Should the program filter, mask, or trim reads to remove potential artefacts?

Should the program filter, mask or trim the reference sequences?

What constitutes a "match" to be counted?

How should match counts be reported to the user?

Several additional 'architecture' options don't affect results but can maximize computing efficiency

Users may face a choice between hardware availability (time, memory, processors) and speed.

User control of hardware use is unnecessary in most (moderate throughput) cases

For large jobs, an ability to specify memory/processor use may help optimize resource use

**Even a very experienced user won't apply all options (or even need to know them)**

So the option list represents a comprehensive reference-- not leisuretime reading material

**II. CountRabbit: 30,000 foot view**

**What goes in:**

The names of your reference file

The location of sequencing datafiles

<Optional> choices of parameters for

**What comes out:**

A table of match counts (K-mers and/or Reads)

A very quick visualization

Important Features: Quick, Simple, Consistent handling of sequences shared between references

**CountRabbit is K-mer based**

Any read that matches at least one k-mer in a reference genome is assigned to that genome

If only one reference genome is matched, a k-mer or read is considered to be "uniquely" mapped

Reads (or K-mers) that match multiple references are considered to be "**public**"

A third (optional) category "**local repeat**" describes K-mers

present multiple times

but all in one reference sequence

With the default K-mer length of 31, specificity of matches is such that false positive will be very rare

**CountRabbit is primarily a tool for counting matches (not for many other tasks):**

CountRabbit is not designed to

provide strandedness data

detect distant homology, or

assemble sequencing data into contigs.

CountRabbit is not designed for (or helpful in) detecting mutations, rearrangements, or termini

CountRabbit provides a provisional (and customizable) graphic output

But this graphic is intended for quick experimental examination

publication may require additional tools.

Features of CountRabbit overlap with

Many excellent and free public tools

Several local tools: REVA,HQAlign,Polybench, Jazz18Heap

If everything is working properly, CountRabbit can be much faster to implement and/or run

CountRabbit is a work in progress

For now, any results (or lack thereof) should be checked/verified with other tools

Please report any bugs/crashes to afire@stanford.edu

**III. Count Rabbit: Basic Operation**

**Syntax**

python CountRabbit<ver>.py Reference=<file,file..> Data=<file,file..>

you will need the module "VSG\_ModuleFP.py" in the same folder (can obtain from Github site)

For the program to run as fast as possible, the PyPy interpreter should be used (www.pypy.org)

**Required Parameter 1:** A Reference file with one or many 'reference' sequences in fasta format

Reference=<file[s]> : Datasets to gather K-mers for analyzing data

File lists can be comma separated (no spaces) and/or specified with a wildcard (\*)

Wildcards require quotes around name

Input files can be gzipped or not (or mixture). Please no spaces,commas,semicolons in filenames.

**Required Parameter 2:** Data Files and Sample Names

Direct FileName Option: Data=<fasta/q file[s]>

Here you input one or more sequencing data files for analysis

This can be a single file or a comma-delimited list and can include wildcards (\*)

"Data" can also be a directory

In this case, all fasta/fastq/fastq.gz/fastq.gz in the directory will be analyzed).

By default the sample name will be the filename

By default, R1/R2 data file pairs are identified by CountRabbit and treated as a single ('Rx') entity

SampleToData MetaFile Option

For more flexibility, provide SampleName/ReadFileNames combinations in your own file

This is invoked with SampleToData=<your SampleFile> instead of Data= in the command line

SampleDescriptor files have simple line-by-line format where each line consists of

A sample name, followed by tab, followed bya comma-delimited list of relevant the file paths

**"Most-likely-to-be-useful"** parameters to explore as you evaluate CountRabbit as a tool

(all of these [and other] parameters have defaults, so you can ignore these and just run the program)

**klen**: K-mer length (default=31) can be anything >~16; values >31 will slow CountRabbit down

**OutFileName:** Default=<assigned by program > : Allows user to specify base filenames for output.

**MaxReadsPerFile**: Max reads to check for each Data file (default is none: all reads processed)

**CircularReference**: (Default=False) Setting to true will treat reference sequences as circles.

**ReferenceFileByFile**: (Default=False) Setting to true treats each reference file as a single entity.

**Express** (Default=False)=True runs in mode that identifies just the first-K-mer match for each read

**FilterList:** Providing a fasta file and a cutoff number (e.g. myFilterFile.fa>5) limits the searched K-mers based on K-mer copy number in a separate reference file

**Threaded**: (Default=SingleThread) Threaded=True instructs rabbit to use a multi-processor mode and decide how many threads to use. Of you can specify a number (e.g. Threads=3 uses three threads). Multithreading is only useful if you have multiple input samples.

**SingleCount**: (Default=False) This will run CountRabbit in a very limited but quick mode where only a single read count is reported for each reference and dataset.

**IV. Packing List:**

A full package for this project consists of a zipped folder (CountRabbitFolder) with the following contents:

1. The CountRabbit python script (CountRabbit\_ver\_date.py)

2. The needed VSG\_ModuleFP.py script

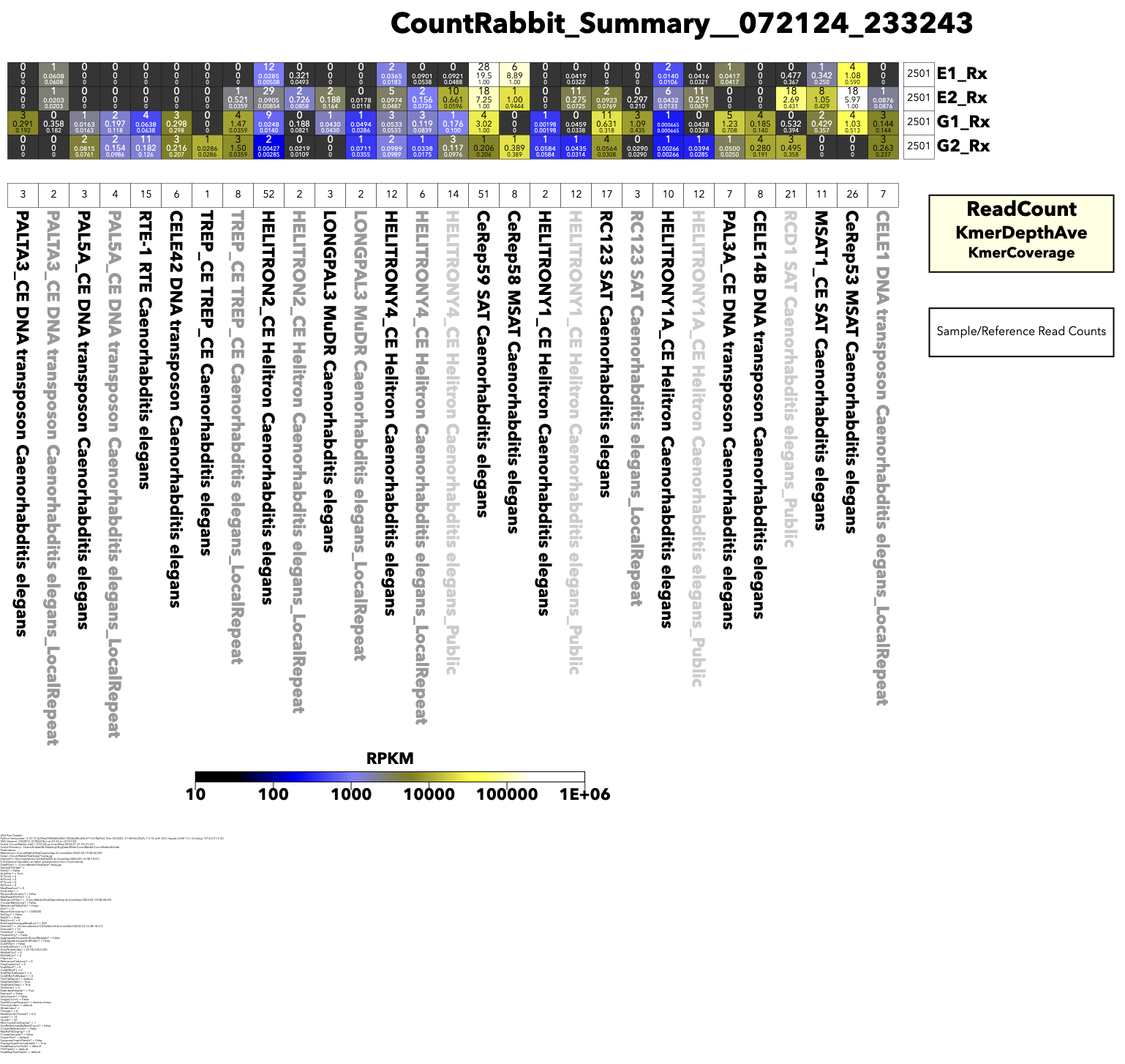
3. This documentation (CountRabbit\_Docs\_ba03\_072324.docx)

4. A useful junk-linker file (illuminadetritis1223wMultiN.fa)

5. A subdirectory containing a set of test data and references sequences (CountRabbitTestData: should contain 8 .fastq.gz files plus a cerep.fa reference file)

The following script, if run in the CountRabbitFolder directory should produce usable test output and a nice graphic within a few seconds. If not, give us a holler.

*pypy3 CountRabbit\_ba03\_072324.py Reference='./CountRabbitTestData/celrep.fa' Data='./CountRabbitTestData/\*.fastq.gz' DetritisF='./illuminadetritis1223wMultiN.fa'*



***V. Build-a-Bear*-inspired guide to building a *CountRabbit* command line**

**Step 0: Get the CountRabbit and VSG\_Module scripts and put into a working directory**

Both scripts can be obtained from www.github.com/firelabsoftware/CountRabbit

**Step 1: Choose your data source**

Option 1A (A single file with reads)

*data=myReadFile*

Read files can be in fasta or fastq format and can be gzipped

Files should be named (or renamed if needed) to avoid spaces, commas and semicolons.

Option 1B (A comma delimited list of files)

*data='myReadFile1,myReadFile2,myReadFile3,myReadFile4'*

Mixtures of fasta, fastq, are permitted (some, all, or none can be gzipped)

List must be enclosed in quotes

KCR will try to pair R1 and R2 files

If pairing fails for any reason, you can use an explicit semicolon-based syntax to specify R1;R2 pairs

e.g., data='myDataA\_R1.fastq;myDataA\_R2.fastq,myDataB\_R1.fastq;myDataB\_R2.fastq'

To turn off autopairing use the command line option AutoPair=False

Option 1C (A wildcard-based specification)

*data='/myDir/\*.fastq.gz'*

KCR can use a wild card syntax. In the example above all .fastq.gz files in directory myDir will be taken as data.

Option 1D (A directory)

*data='/myDir/'*

KCR will take all files in the directory that have a .fasta, or .fastq ending (with or without .gz)

Option 1E (A metafile list)

*SampleToData=mySampleToDataFile*

KCR will use the indicated file (mySampleToDataFile in this case) to obtain

a list of sample names

a list of files that contain data for each sample, and (optionally)

a category or display color for each sample

The format of SampleToDataFiles is simple

Tab-delimited format. Each line is one sample

The first column (first item on each line) is the sample name

Second and subsequent columns can contain filenames that carry data for the indicated sample

Comma delimited list or any of the formats above.

Non filenames will be ignored

Optional: Subsequent columns can also include a designation of category or display color

Category designation format: category(Reservoir)

Color designation format: color(red) or rgb(255,0,0)

Example:

**Sample Files Category(optional) Color(optional)**

mySample1 myFile1,myFile2 category(Reservoir) color(Blue)

mySample2 myFile3,myFile4 category(Well) color(Red)

**Step 2: Choose your reference dataset: what do you want counted in the data?**

Option 2A (A single FastA file)

*reference=myFastAReferenceFile*

KCR is designed for medium-sized reference datasets. Sets of genes, microbes, and even sets of microbes are handled well

Larger genomes (e.g. the human genome) can be handled with enough computer memory but may take time to generate indexes

Option 2B (A comma-delimited or wildcard-based list of FastA files)

*reference='myFastAReferenceFile1,myFastAReferenceFile2,...'*

A list of fasta reference files is permitted (as are wild cards).

Note that filenames should not contain commas. spaces or semicolons and lists should be in quotes.

By default, each fastA entry is treated as a one reference entity. Setting ReferenceFileByFile=True treats each file as one entity.

**That's it... KCR can now be run**: e.g., *pypy3 data=\*.fastq reference=myRef.fa*

The output will be two files...

a \*.tdv file

tab-delimited table with one row for each reference sequence and one column for each sample value being counted.

a \*.svg file

a graphic summary of counts that can be visualized using any browser (Firefox, Chrome, Safari, etc)

**But wait**... There are many more options in building your bear. No self-respecting bio-informatics software would be complete without a bunch of different optional settings to determine how the input data is treated, what is being counted, how the counting is done, and what to do with the data once obtained. The options are described below.

**VI. CountRabbit: All User settings and options**

**A. Essential Parameters**

Only two inputs are needed to run CountRabbit-- a Reference File (or list) and a Data File (or list)

**Data:** Input data file list. List should be enclosed by quotes (e.g. 'Run1.fastq,Run2.fastq,Run3.fastq'), and can be gzipped or include wildcards ('\*') **SampleToData** is an alternative means to input a list of datafiles (in which case a "Data=" statement is not needed). The SampleToData file specifies one sample nume on each line, followed corresponding list of input sequence run files (a full file path can be used). Format: Each line should have SampleName <tab> comma-delimited list of filenames

**Reference**: A .fasta file or list of files (can be .gz or .zip and can include wildcards) with reference sequences to match against

**B. Data Input Options**

By default, CountRabbit processes all bases in each sequence read (or read pair for paired-end sequence data). This can be adjusted as below in several different use cases where the goals are served by processing only a subset of either reads or only a part of each read.

**NoNs:** Setting this to true ignores any read with "N" bases (default=False)

**AutoPair**: True Setting this to true attempts to pair read1 and read2 files (default=True)

**R1Trim5**: Length to trim from 5' end in R1 (default=0)

**R2Trim5**: Length to trim from 5' end in R2 (default=0)

**R1Trim3**: Length to trim from 3' end in R1 (default=0)

**R2Trim3**: Length to trim from 3' end in R2 (default=0)

**MaxReadLen**: After filtering, reads will be truncated to this length (default=none)

**EndLinker**: A linker at the end of the read to be chewed off using an rfind command (default=none)

**RequireEndLinker**: Setting this to true removes any read without this linker (default=False)

**MaxReadsPerFile**: Set to a positive number to limit the number of reads per file (default=none)

**FastQDumpProgram**: :CountRabbit can directly download one or more SRR/ERR/DRR files that are described in the data= option. The files will be downloaded from NCBI, and the FastQDumpProgram=<path to fastq-dump> is needed just to provide a full path of fastq-dump/fasterq-dump program to execute the download.

**B. Reference Input Options**

By default, CountRabbit treats each sequence in the list of references as a single linear segment from which to extract K-mer words. This can be adjusted using the options below.

**CircularReference**: (Default=False) Setting to true will treat reference sequences as circles.

**ReferenceFileByFile**: (Default=False) Setting to true treats each reference file as a single entity.

**C. K-mer length option**

By default, CountRabbit uses a relatively long K-mer length (31), yielding a highly specific identification opportunity amongst a presumed sea of otherwise unrelated sequences. The length provides an effective balance of speed, specificity, and tolerance for some error in standard sequencing. Nonetheless, the length can be adjusted in cases where this is optimal for the task at hand.

**klen**: K-mer length for counting matched subsequences (default = 31)

**D. Avoiding adaptor junk ('Detritis' option)**

Most sequencing workflows use dedicated sequence adaptor oligonucleotides that provide operational access for amplification, selection, or priming to libraries of captured DNA or RNA molecules. These adaptors can themselves appear in multiple datasets in unexpected places, generally as artefacts of the library construction process, and can be present in some assemblies. To avoid spurious and at times misleading counting adaptor-derived K-mers in sequence counts, CountRabbit offers an option to avoid such K-mers. A fastA file (or several) with known sequencing adaptor/primers can thus be provided, along with a defined K-mer length to be used in deciding which segments to ignore in generating counts.

**DetritisF**: Name of tetritis file for optional on-the-fly masking of linker bits (file illuminatetritis1223wMultiN.fa, distributed with this program, can be used. Note that the distributed file has many of the publically available Illumina linkers as well as homopolymers of the four nucleotides

**DetritisK**: Length of K-mer for tetritis filtering (default=16)

**E. Options for detailed handling of multimapping/repeated sequences**

By default, CountRabbit places K-mers in three bins: unique (sequences present only once and only in a single reference), local repeat (sequences present multiple times, but in a single reference), and public (seuqneces present in more than one reference). Mapping [detailed below] consists of finding the K-mer that provides the most informative positioning, thus enabling a high fraction of reads to be mapped even if a fraction of the included K-mers are public or repetitive. The defaults here can be adjusted, ignoring repetitive K-mers or grouping them with unique K-mers. Generally these options are of most value if a tradeoff of faster execution and simpler output is desired at the expence of the precision which is the default. The final setting here (NoDup) increases precision of K-mer counts in some cases by avoiding double counting in the case of paired end reads (no effect on read counts).

**FirstDibs**: Setting this to true assigns each k-mer and read that is shared to only the first matching reference, disallowing multimapping (default=False)

**PrivateOnly**: Setting this to true ignores any k-mer present in more than one reference (default=False)

**AggregateUniqueAndLocalRepeats**: Setting this to true provides a single output that sums both unique and local repeats (default=False)

**AggregateUniqueAndPublic**: Setting this to true aggregates public k-mers with unique for each reference sequence. Generally used with FirstDibs and AggregateUniqueAndLocalRepeats (default=False)

**NoDup**: Setting NoDup to True instructs Rabbit to not to count k-mers in a paired end read that are also in R1. This provides somewhat more nuanced K-mer counts but will slow the program down substantially; Read counts are not affected (default = False)

**F. K-mer Complexity Filtering** (inspiration: DustFilter work from R Agarwala, D Lipman)

K-mer complexity Filtering Overview: Invoking K-mer Complexity filtering allows CountRabbit to ignore K-mers with a repetitive or low complexity sequence composition. The algorithm is inspired by work of Richa and David, in this implementation several different k-mer lengths are chosen (variable:SubKLen, default value: 3,6,9). DustThresholds is then a set of thresholds (coincidence frequencies)-- essentially the probability that two randomly chosen k-mer of the indicated lengths will coincide. The code calculates how many sub-k=mers of each length match other sub-k-mers (total coincidences as a function of the maximum possible number/ A homopolymer k-mer a c/mc ratio (coincidences/maximal-coincidences) of 1.0. A sequence with no sub-k-mer appearing more than once will have a c/mc of 0.0. Sequences of intermediate complexity will have an intermediate c/mc. Heuristic exploration leads to a provisional recommendation of subK-mer lengths of SubKLen=(3,6,9) for 32-mer words, with thresholds >0.18, >0.05, >0.02 for calling a given k-mer degenerate

**DustFilter**: Setting this to true implements a simple filter to avoid low complexity search K-mers (default=False)

**DustSubKLen**: These are the K-mer lengths that we'll test complexity at (too few different subk-mers of any of these lengths will trigger a triaging of the parent K-mer. (default=3,6,9)

**DustThresholds**: These are the maximum coincidences/max-coincidence thresholds for the different K-mer lengths to be tested (default=0.18,0.05,0.02)

**G. Reference K-mer Filtering 1 (Filtering reference Contigs by Length/Coverage)**

By default, CountRabbit uses every contig or scaffold in the Reference fasta as a source of K-mers to count in the user data files. In some cases there may be a large number of short contigs or contigs with minimal representation that could usefully be ignored. The following settings allow the user to enable such reference filtering.

**MinRefLen**: setting this to a positive integer ignores all reference segments that are smaller than this value (default=0)

**MinRefCov**: setting this to a positive floating point ignores all reference segments where the coverage reported by MegaHit or Spades is lower than this value (default=0). Note that this option only works with MegaHit or Spades output, since it uses to scaffold/contig description as the source of coverage level.

**H. Reference K-mer Filtering 2 (Filtering K-mers by incidence in a reference)**

By default, CountRabbit uses every K-mer in each reference contig or scaffold. In some cases there may be k-mers with a high copy number in the reference that would give anomalous results. The following settings allow the user to ignore K-mers whose copy number in the reference file (or other files if desired) exceeds a user-provided threshold.

**FilterList**: Setting this to a filename (or file list) and an inequality will filter k-mers based on k-mer copy number in that (arbitrary) fasta file (or group of files). Adding FilterList='MyFile.fa>5' to the command line will only use K-mers with >5 instances in MyFile.fa. As an important special case of this, using the word 'Reference' instead of a filename will filter based on the number of instances of a given k-mer in the reference files used for the run. So FilterList='Reference>5' will restrict the searching by CountRabbit to only k-mers present >5 times in the provided reference file. To provide a more complex instruction for FilterList, multiple files can be separated by commas, and multiple filters seprated by semicolons Thus as a complex example FilterList='reference>5;myFileA.fa>3;myFileB,myFileC<2' will keep and analyze only K-mers that meet the criteria

More than 5 copies of the K-mer the aggregate reference files specified in ReferenceFiles=

More than 3 copies of the together in myFileA.fa file

Less than 2 copies (total) in the files myFileB myFileC

**I. Reference K-mer Filtering 3 (Filtering K-mers by incidence in the dataset)**

Some k-mers show an unexpectedly high frequency in a dataset, indicative of potential artefacts, such as the k-mer may matching a different organism, or reflect a prominent contaminant. K-MER Recusal filtering provides an approach toward sensitive and accurate identification of sample-to-sample count differences that will avoid small numbers of confounding K-mers with unusually high incidence. Rabbit has two options to "recuse" K-mers based on frequency-in-data (rank-based and median based), described bwlos. In general the recusal filters work by obtaining a counts-per K-mer distribution for each reference segment in the actual data. Once these simple K-mer counts are obtained, a filtering step recuses K-mers that are outside of a range that is set by the user. "Hot" filters are designed for the primary purpose of removing k-mers that are unusually frequent, while "Cold" filters remove k-mers that that are unusually rare (the latter being a less likely application, but provided nonetheless)

**i. Rank-based recusal**

The user can set fractional thresholds based on the actual distribution of K-mer counts for each reference

The relevant parameters are ColdKmerF1 and HotKmerF1

Setting **ColdKmerF** to a non-zero value x will recuse all K-mers in the bottom x fraction as sorted by sample counts

So an example: Setting **ColdKmerF**= 0.2 will recuse the bottom 20% of K-mers by total count from each reference segment

Setting **HotKmerF** to a value <1 recuses all but the indicated fraction of K-mers sorted by frequency in samples

So setting **HotKmerF**=0.8 will recuse the top 20% of K-mers for each reference based on count

**ii. Count-relative-to-median based recusal**

The user can set a threshold that will be based on median k-mer counts for each reference

Setting **ColdKRelToMedian** to a value (generally <1) recuses all K-mers whose counts are below ColdKRelToMedian\*median\_kmer\_Coverage for that reference

Setting **HotKRelToMedian** to a value (generally >1) recuses all K-mers whose counts are above HotKRelToMedian\*median\_kmer\_Coverage for that reference

Note that gathering the recusal list entails a complete analysis of the data to obtain counts and identify k-mers for recusal

The recusal run is also done in single-core mode, so this will certainly slow the program down overall.

Note also that you can save the recused K-mer list with the WriteIndex=<myIndexFileName> option

(and use this recused index in a future run with ReadIndex=<myIndexFileName> option)

**HotKmerF**: Setting HotKmerF<1 instructs Rabbit to recuse any K-mer above this fractional rank amongst most abundant K-mers in each reference sequence (default=none). HotKmerF is of particular utility in cases where a small number of k-mers in a reference are mapping frequently to some other related (or unrelated) sequence. Example: Setting HotKmerF=0.98 will ignore the most abundant 2% of k-mers in every reference (default=none)

**ColdKmerF**: Sets a floor on K-mer counts based on rank

**HotKRelToMedian**: Sets a ceiling on K-mer counts based on median count over all datasets for K-mers in that reference sequence (default=none)

**ColdKRelToMedian:** Sets a floor on K-mer counts based on median count over all datasets for K-mers in that reference sequence (default=none)

**J. Settings for streamlined operation (compromise settings [some loss in output info but likely to be minimal in many applications])**

In some cases it may be advantageous to expidite the running of CountRabbit, to minimize the memory or processor usage footprints, or to simplify the outputs. Several setting enable different options that (under specific circumstances) may result in a streamlined operation. The first group of options (below) are not cost-free in terms of output in that there are details of output and/or specific complex positioning cases that may be more accurately dealt with by the default settings. A second group of options are cost-free in that they only tweak the internal memory and/or processor usage with no consequence to the eventual output. It is anticipated that these options will only be used in special cases where there are large datasets or large groups of datasets to be dealt with (e.g. searching an entire hard drive with data='~/').

**Express**: Setting this to True provides an express version of the program yielding only read counts (no k-mer counts) and with ambiguous reads assigned to first match (default=false)

**VerySimple**: Setting this to true runs RabbitCount in a paired down mode that just looks for unique K-mers and reports read counts (default=false)

**SingleCount**: Setting this to true runs RabbitCount in a minimal mode where all reads "belonging" to a reference (whether unique, local repeat, or public) are grouped in one category (derault=False)

**K-merThinning**

For very large reference sequence spaces it may be impractical to store all K-mers in memory, even in condensed form. In such cases, a "ReferenceCadence" can be set, which will use K-mers periodically sampled from the reference sequence. For very large datasets, the search itself may become CPU time intensive, in which cases a different type of K-mer thinning can be employed in which only a fraction of k-mers in each read are tested for identification. As an example, setting ReferenceCadence=8 will use only every 8th k-mer in searching for matches, setting DataCadence=8 will search only one in every 8th k-mer in each sequencing reads. Note that these two options are not useful together, and that cases where memory is not limiting and the majority of reads fail to match will benefit most substantially from the lossless built in 'Bialy' filtering below (either cadence option is not compatible with Bialy filtering-- so turning off Bialy filtering with BialyK=0 is recommended). Also note that either **Cadence** option can lose some ability to distinguish reads from highly related reference sequences, and could miss some reads.

Despite these caveats, for large reference sets or large datasets respectively, either K-mer thinning option provide a consistent and reflective set of metrics with a much small memory footprint and faster run times.

**ReferenceCadence**: Setting this to a value of n>1 will limit the k-mers index from the reference to only every n'th k-mer (default=every k-mer searched)

**DataCadence**: Setting this to a value of n>1 will limit the k-mers searched in the data to only every n'th k-mer (default=every k-mer searched)

**K. Settings for streamlining operation (Cost-free settings [no loss in output info, some of these may result in improvements under specific data conditions])**

**Using the PyPy version of Python**

Any version of python beyond 3.7 can be used. For speed, PyPy (version 3.9+) is highly recommended (can be downloaded at www.pypy.org and doesn't require a dedicated installation).

**Storing and retrieving prebuilt indexes**:

The index created by CountRabbit for searching is normally created as a temporary entity and thus lost when the program completes. For large input reference sets (on the order of gigabases), the time to create an index can be large. Similarly if a substantial amount of filtering (dust or incidence filtering) is to be applied, the eventual k-mer index can represent the result of considerable processing time. Thus it can be advantageous to store an index for use in future runs of CountRabbit.

**WriteIndex**: Setting this to a filename will write the index that is compiled so it can be used for a future run. For large reference files (>100s of MB) using this file can save time in future runs (default=None [no index written])

**ReadIndex**: Setting this to a filename will read the index that is compiled. Note that the reference files used to generate the index must be the same as those for the current run, along with all settings except for the data file list.

**Data-derived Indexing**:

In cases where a reference sequence is larger and/or more complex that the data (sequence reads), the workflow and memory footprint of CountRabbit can be minimized with the setting PrimaryIndex='data'. With this setting, a modest dataset (up to several gigabases) can be compared to a much much larger reference or set of references, with essentially no limit.

**PrimaryIndex**: Default is to derive the primary index from the input reference. Setting PrimaryIndex=Reference will enable counting operations for reference datasets that are themselves much larger than could fit in memory

**MultiThreaded Operation**:

CountRabbit has an effective (although somewhat primitive) multithreading capability in which multiple CPU threads are handling different input files at any given time. To engage these, the user can set **Threaded=True** and CountRabbit will choose the number of threads to use based on some simple memory assessments (not perfect). To override the automatic processor count setting, the user can set Threads equal to a specific ask of course (e.g., 1, 2, 3, 20-- depending on the system)

**Threads**: Number of threads for multiprocessing (zero for single core execution, "Auto" or "True" to set this automatically, or provide a specific integer of how many cores to use)

**MaxMemPerThread**: Maximum fraction of total memory that will be allocated per Thread (default=0.5-- half of total)

**The "Bialy filter"**

The Bialy filter is a built in feature of CountRabbit that implements a pre-screen that triages reads that will not match any input k-mer. This option speeds up the program for datasets where only a small fraction of reads match the reference. The Bialy filter algorithm is similar to a Bloom filter (Bloom, B. doi:10.1145/362686.362692), but has been optimized for the filtration of reads (rather than K-mers) where overlapping internal words are non-independent. The name Bialy Filter is coined herein (thanks to S.U. Enam for pointing out the similarity between the filter used and a Bloom filter, and note that the linguistic association of Bialy and Bloom was derived from the work of M.Brooks, G.Wilder, and Z.Mostel; www.imdb.com/title/tt0063462/). While the BialyFilter is intrinsic to CountRabbit and will choose it's own K-mer length setting by default, these can be overriden by the user if they choose to.

**BialyK**: A K-mer length to be used for initial prescreening. This is particularly useful for datasets where most reads fail to match the reference (the default is for CountRabbit to set this)

**BialyLimit**: Setting this to a positive number n insists that there be a matching number within the first n bases of any read for it to be considered (default=none)

**EstimatedAverageReadLen**: Read length is one parameter in setting BialyK value (default="AutoSet"). For optimal execution,. this can be set to an expected read length (positive integer) if you have an expected value (e.g. 150 for HiSeq). Generally the speed/memory advantage from explicitly setting this will be small.

**L. Settings for output**

CountRabbit produces two output files by default:

1. A tab-delimited-text file with counts of k-mers and/or reads that can be normalized if the user chooses this.

2. A .svg graphics file which provides a preliminary graphic view of the results of the counting

Numerous command line parameters allow the customization of outputs for effective use in downstream applications.

**General Output Settings**

**OutFileBase**: By default, CountRabbit will generate a unique file name based on the current time and date. This option allows the user to specify a file base-name for the tab-separated-value and .svg ouput files (CountRabbit will still add .tsv and .svg to the user basename)

**SkipBlankRefs**: Setting this to True will mask output for reference sequences that are not detected (default=True)

**SkipBlankData**= True Setting this to True will mask output for datasets where no reference match is detected (default=True)

**Delimiter**: New line delimiter for output (default='\r')

**ExtendedHeader**: Setting this to True gives an extended descriptive header at the top of the tab delimited text output file (default=True)

**ReportGranularity**: How frequently to report updates during a run (default = 1000000; no effect on eventual tables or graphic... just a means to see if the program is still operating)

**Transpose**: Setting this to true provides an additional text file output that is rotated 90-degrees (rows are samples, columns are references)

**Some settings for graphic (.svg ouput)**

**vscale**: Vertical scale for graphic (default=14)

**hscale**: Horizontal scale for graphic (default=42)

**MinCountsForDisplay**: References with fewer average reads-per-sample will be skipped in display (default=1)

**SortReferencesByReadCount**: Setting this to true will sort references from low to higher counts (default=False)

**ClusterReferences**: Setting clusterferences to true implements a very rudimentary 'euclidian' reference clustering for the graphic display (default=False)

**MaxRefToDisplay**: Limits the number of references displayed on the illustrative graph (zero is unlimited). Priority for display is based on K-mer counts (read counts in express mode). (default=no\_limit)

**ClusterSamples**: Setting clustersamples to true implements a very rudimentary samples clustering for the graphic display (default=False)

**GraphTitle**: Sets a graph title. Default will be OutputFile Name

**SuppressGraphDetails**: setting this to true suppresses the detailed listing of program parameters on the output graphic (default=False)

**DisplayGraphImmediately**: Setting this to false will suppress immediate graphic display of SVG file (default=False)

**M. Choice of output fields**

CountRabbit can provide either raw counts (of k-mers or reads) or normalized/processed counts reflecting a variety of possible downstream uses. The user can choose what output fields they want for table and graphic output files using the settings HeatMapTextFields, HeatMapColorField, and TSVFields. Each of these can be include one or more items from a list of raw counts and computed values related to what CountRabbit has discovered in the data. The options for different reported parameters and their corresponding id numbers are as follows (numbering goes from 7 to 19). Note that either the result name or the result ID can be used in specifying display preference (e.g. either HeatMapColorIndex=16 or HeatMapColorIndex='RPKM')

**ID Name Description**

7 KmerCount Raw K-mer Counts

8 KmerCount2 Sum of Raw K-mer counts squared

9 ReadCount Raw Read Counts

10 KmersCovered Number of K-mers covered

11 KmerDepthAve Average depth of coverage

12 KmerDepthStdDev Standard deviation of coverage depth

13 KmerCoverage Fraction of K-mers covered in data

14 KmerSampleFraction Fraction of pass-filter K-mers in sample assigned to this partition

15 ReadSampleFraction Fraction of pass-filter reads in sample assigned to this partition

16 KPKM K-mers mapped per thousand reference K-mers per million pass-filter K-mers

17 RPKM Reads mapped per thousand reference K-mers per million pass-filter reads

18 KmerEnrichment (K-mer fraction-this partition for this dataset) / (K-mer fraction-this partition for all datasets aggregated)

19 ReadEnrichment (Read fraction-this partition for this dataset) / (Read fraction-this partition for all datasets aggregated)

**TSVFields**: Specifying one or more of the measurement values above (#11-19 above) in this list expands the .tsv output to include these (default TSVFields='ReadCount,KmerCount,KmersCovered').

**HeatMapTextFields**. Specifying one or more fields will add the relevant values to the heatmap as text annotations in the relevant boxes. (default:HeatMapTextFields='ReadCount,KmerCount,KmersCovered'). Default setting will adapt as needed if other choices make K-mer counts unavailable or incomplete.

**HeatMapColorField**: Sets the raw or processed value above used to color rectangles in the summary heatmap. (default: HeatMapColorField='RPKM')

Note that K-mer counts are not calculated in Express mode, so any kmer-reliant value will be recorded as zero in Express mode

**VII.Description of matching criteria used to generate read counts**

The following discussion is a detailed description of the sausage making involved in assigning reads.

With a long-ish k-mer (e.g. the default, which is 31) and reference sequences that are not highly degererate, the following considerations are not substantial

For pools of references that are very similar to each other, the discussion becomes more relevant

The simplest category of K-mers are those that are present just once and in one reference ('Unique' k-mers)

Any single k-mer shared between a read and a reference nominates the read as matching that reference

In cases where there is no ambiguity (ie no match to any other reference from the read), the nomination is counted as a "private match"

This means that a read with any number of k-mer matches to a reference is counted as a match to the reference

Reads that match unique "private" k-mers from more than one reference are assigned arbitrarily to last matching reference in the list

K-mers present in more than one reference are a second category and are called 'public'

Reads with 'public' K-mers can still be assigned to an individual reference if there are 'private' k-mers matching a unique reference

In cases where all K-mers in a read match only public k-mers, the read is assigned to the public category

A public-category read will still be assigned to one or more references; so each reference can have its own "public" category

Assignment of a public read is based on the most restricted K-mer present in that read (K-mer present in fewest references)

In standard mode, public reads are assigned to every reference matching the assigning k-mer.

Turning on the option FirstDibs=True alters this so that only the first reference with the assigning k-mer is credited

Turning on the option PrivateOnly=True tells CountRabbit to ignore public K-mers and reads (focusing only on k-mers belonging to a single reference)

-

A third category of k-mers ("LocalRepeat" K-mers) are those present more than once but only in a single reference.

These are reported as a separate category in K-mer counts

Reads that only match LocalRepeat K-mers are reported also as in the LocalRepeat reads category

Reference Assignment based on a unique k-mer takes precedence over assignment based on a local repeat k-mer

Setting the parameter AggregateUniqueAndLocalRepeats=True instructs CountRabbit to group local repeat and unique k-mers together

The combined Unique+LocalRepeat category is then designated "Private"

*Documentation First Version, Stanford-CA-USA July 21, 2024. This Document, and CountRabbit and VSG\_Module Code ©2024, Andrew Fire and Stanford University*

*Thanks to all colleagues in the past-present FireLab for their help in making this happen... with valued contributions from Karen Artiles, Drew Galls, Usman Enam, Ivan*

*Zheludev, Massa Shoura, Matt McCoy, Emily Greenwald, Dae Eun Jeong, Orkan Ilbay, and Collette Benko in aspiration and execution of these algorithms and code.*

*There is a 99.999999% chance of bugs in code and documentation (or both). Please report these to afire-at-stanford-dot-edu*