The Karyon v1.1 manual
By Miguel Naranjo-Ortiz
April 20th 2022

Introduction

1. Installation

2. Basic usage

3. The config file

4. The Karyon pipeline

    4.a. Trimming

    4.b. Genome assembly

    4.c. Reduction

    4.d Variant calling

    4.e. Plot generation

5. How to interpret the results

    5.a. Report

    5.b. Scaffold length plots

    5.c. Scaffold versus coverage

    5.d. Variation vs. coverage

    5.e. Fair coin plot

    5.f. nQuire per scaffold plot

    5.g. K-mer distribution

6. Testing the pipeline

Introduction

Karyon is a pipeline for the assembly and analysis of highly heterozygous genomes. It uses redundans (Pryszcz & Gabaldón, 2016) to reduce heterozygosity during the assembly process, and then maps the original libraries against the reduced assembly to analyze the distribution of heterozygous regions. With this information, it generates a series of plots that can aid researchers to generate informed hypotheses with regard of the architecture of their genomes.

Scripts contained in this repository:

A. **karyon.py** -> complete pipeline, including genome assembly, assembly reduction, SNP calling and plot generation
B. **prepare_libraries.py** -> Karyon dependency. It makes a series of characterizations regarding the input libraries that are used by other parts of the script.
C. **trimming_libraries.py** -> Karyon dependency. It uses Trimmomatic (Bolger, Lohse & Usadel 2014) to trim input libraries before genome assembly.
D. **spades_recipee.py** -> Karyon dependency. It generates a file that launches SPAdes (Prjibelski et al. 2020) or dipSPAdes (Safonova, Bankevich & Pevzner 2015) with the input.
E. **varcall_recipee.py** -> Karyon dependency. It generates a file that launches all steps in the SNP calling pipeline.
F. **karyonplots.py** -> Karyon dependency. It generates all the plots as part of the Karyon pipeline.
G. **all_plots.py** -> Standalone version of karyonplots.py. It allows the user to input karyon results to generate the plots again.
H. **nQuire_plot.py** -> It allows the user to run the local ploidy plot alone.
I. **trimming_libraries_alone.py** -> Allows to run Trimmomatic as a standalone program.
J. **launch_bwa.py** -> Karyon dependency. It is used to configure BWA (Li & Durbin 2009).
K. **report.py** -> Karyon dependency. It is used to process the output of other parts of the script and generate a report providing some statistics and warning messages.
L. **fasta_stats.py** -> Karyon dependency. It is used to calculate some parameters of the generated fasta files.
M. **FastaIndex.py** -> Karyon dependency. It is used to calculate some parameters of the generated fasta files.
N. **create_config.py** -> It is used during the installation process to generate the configuration file, that the program uses to locate the dependencies.
O. **Dockerfile** -> Docker file involved in building the image to run Karyon
P. **installation.sh** -> Bash script required to install the remaining dependencies in the dockerfile
Q. **redundans_env.yml & busco_env.yml** -> Conda environments in YAML format required to install some of the trickiest dependencies

1. Installation

Before starting, you need to decide whether to use standard installation or to install Karyon in a Docker container.

For standard installation from GitHub:

```
git clone https://github.com/Gabaldonlab/karyon.git

cd karyon/scripts/

bash installation.sh
```

For Docker installation:

Prerequisites

In order to run this container you'll need docker installed.

- [Windows]
- [OS X]
- [Linux]

Note: When using docker, make sure you have enough space for its installation. If needed prune and clean unused docker images and containners

```
docker system prune
```

Download stable container

Pull `gabaldonlab/karyon` from the Docker repository:

```
docker pull cgenomics/karyon:1.1
```

Build your own container

Or build `gabaldonlab/karyon` from source:

```
git clone https://github.com/Gabaldonlab/karyon_docker.git

cd karyon

docker build -t cgenomics/karyon:1.1 .
```

Running the image

Every time you run a Docker container, it uses the Karyon Docker image to remake it. When the container is closed, everything in it goes away! Also, for every one of those containers you need to name it with an alias and include a volume to work with.

```
docker run -dit --name=your-alias -v
/your/karyon/location:/root/src/karyon/shared --rm
cgenomics/karyon:1.1
```

Install the necessary dependencies

Before launching the karyon script we need to install some necessary dependencies:

```
docker exec -it your-alias bash

cd /root/src/karyon/shared/

bash scripts/docker_install.sh
```

Launch Karyon script

```
docker exec -w /your/karyon/location/ your-alias python3
bin/karyon.py -l shared/your-data-file-one
shared/your-data-file-two... -d shared/output
```

Ports

This container exposes the following ports:

Port	Protocol	Service
8080	TCP	http

2. Basic usage

Once everything has been properly installed, you can run Karyon. In its most simple form, Karyon will take a set of Illumina libraries and a name for the output directory where it will generate everything. Be aware that Karyon will create several files that are typically quite large. Be sure you have plenty of space in your hard drive. If the full pipeline is run, it will generate a trimmed copy of the original sequencing libraries, as well as SAM and BAM files, that can take a lot of space. Karyon will remove some of these files after they are no longer needed, but the space needs to be available.

Let us assume we are in a directory that contains the sequencing libraries L1_1.fastq, L1_2.fastq, L2_1.fastq and L2_2.fastq; and we want Karyon to generate the results in a directory called “karyon_results”. The command for that is:

```
python3 karyon.py -l L1_1.fastq L1_2.fastq L2_1.fastq L2_2.fastq -d
karyon_results
```

By default, it will create a random string of characters to use as a prefix for all the output files. If we want to set a name, we can do it with the flag `-o` or `--output_name`. For example, if we want to give the prefix “example”, we can do it with:

```
python3 karyon.py -l L1_1.fastq L1_2.fastq L2_1.fastq L2_2.fastq -d
karyon_results -o example
```

3. The config file

The installation generates a file in the main directory called configuration.txt. This is read by Karyon to figure out where the dependencies are located and the system, as well as introducing non-default parameters. The file contains several lines that start with "#", which are ignored.

For each of the programs used by Karyon we have three lines:

- The name of the program starts with "+"
- The location of the program starts with "@". An empty line implies that the program can be accessed directly from the environment.
- Any additional flag can be introduced with a line starting with ">". Each additional flag can be inputted with an additional line.

For example, let's imagine we want to configure the location of SPADes, one of the programs Karyon can use to perform *de novo* genome assembly. If our SPADes is installed in our main directory, the configuration file should contain something like this:

```
+SPAdes

@/home/SPAdes-3.9.0-Linux/
```

Now we decide that this installation of the program is not the one we want to use. We have this software configured in our environmental variable. In that case, the configuration file should include this:

```
+SPAdes

@
```

Now we have found that our genome has an extreme value of GC%. Our recommendation in such a case is to run SPADes with the flag --sc, which is an option that tolerates a wider range of *k*-mer frequencies. To run Karyon with this flag active, we need to modify the configuration and add the following:

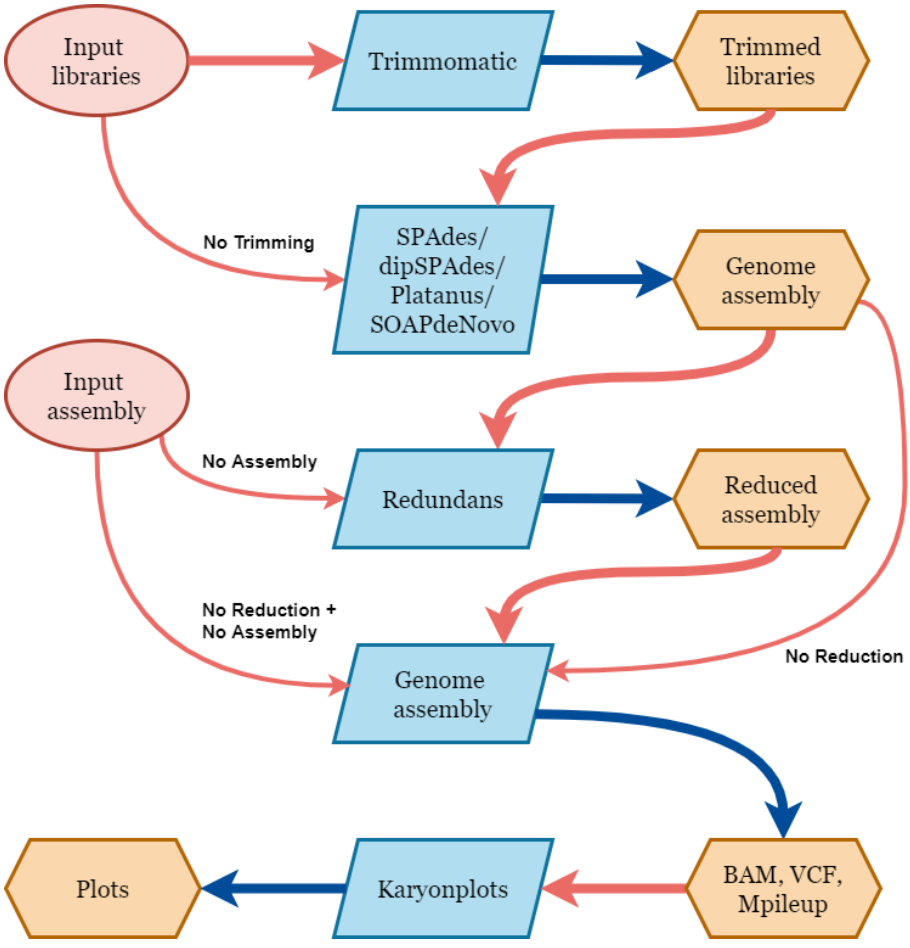
```
+SPAdes

@

> --sc
```

You can define which config file to use with the flag `-c` or `–configuration`. We recommend that you keep the original configuration file and make modified copies whenever you need to run Karyon with non-default configuration.

4. The Karyon pipeline

The Karyon pipeline consists on the following steps:

- **Trimming**: Remove adapters and low quality positions from Illumina reads.
- **Genome assembly**: Generate a *de novo* assembly using the cleaned libraries.
- **Reduction**: Karyon uses Redundans to improve genome assembly quality by collapsing contigs with high similarity. This produces an assembly with reduced heterozygosity that will be used as reference for variant calling.
- **Variant calling**: One of the original libraries will be mapped against the reference in a standard variant calling pipeline.
- **Plot generation**: Using the results of the previous steps, Karyon will generate a series of plots that can help to understand the genomic architecture of the genome.



4.a. Trimming

By default, Karyon will apply Trimmomatic (http://www.usadellab.org/cms/?page=trimmomatic) to the libraries selected as input. It might happen that our libraries are already trimmed, and we want to skip that step. This can be done with the flag `-T` or `--no_trimming`.

```
python3 karyon.py -l L1_1.fastq L1_2.fastq L2_1.fastq L2_2.fastq -d karyon_results -T
```

4.b. Genome assembly

Karyon is able to use several popular genome assemblers for the creation of the initial assembly. By default it will use dipSPAdes, which is an implementation of the SPAdes genome assembler for heterozygous diploid genomes (https://github.com/ablab/spades/tree/spades_3.9.0). You can run Karyon with the standard SPAdes, Platanus (http://platanus.bio.titech.ac.jp/platanus-assembler) or SOAPdeNovo2 (https://github.com/aquaskyline/SOAPdenovo2). You can specify which program to use with the flag `-g` or `–genome_assembler`. Accepted values are 'dipspades', 'dipSPAdes', 'spades', 'SPAdes', 'platanus', 'Platanus', 'soapdenovo' and 'SOAPdenovo'.

For example, if we want to run the previous example using Platanus as our genome assembler, we can use the following command:

```
python3 karyon.py -l L1_1.fastq L1_2.fastq L2_1.fastq L2_2.fastq -d karyon_results -g platanus
```

Alternatively, we might already have a genome assembly that we want to use. In that case, we can provide the flag `-A` or `–no_assembly`. This will tell Karyon not to run the genome assembly, and requires a fasta file that will be used for subsequent steps. For the previous example, imagine that we already have a genome assembly in a file called example.fasta. We can use it instead of running the assembly with the following command:

```
python3 karyon.py -l L1_1.fastq L1_2.fastq L2_1.fastq L2_2.fastq -d karyon_results -A example.fasta
```

4.c. Reduction

Karyon uses Redundans (https://github.com/lpryszcz/redundans) to generate a reduced assembly that will be used as reference in the variant calling protocol. You can skip this step with the flag `-R` or `--no_reduction`. If you skip it, the reference genome will be the result of the genome assembly step. If the genome assembly step is also omitted, the reference will be the fasta file provided as input.

4.d. Variant calling

Karyon will use the assembly generated or provided by previous steps as reference for a variant calling pipeline using BWA-MEM (https://github.com/lh3/bwa), GATK (https://github.com/broadinstitute/gatk) and Bedtools (https://github.com/arq5x/bedtools2). This part of the pipeline can be skipped with the flag `-V` or `--no_var_call`. However, doing so will also skip plot generation, as the plots require the results of this step.

In order to save time, Karyon will only use one library for the variant calling protocol, which is internally referred to as the "favorite" library. The default behavior is that the favorite will be to the library with higher depth of sequencing. You can manually select which library to use as favourite with the flag `-F` or `--favourite`. Let's imagine that we want to use L2_1.fastq as favourite in the previous example:

```
python3 karyon.py -l L1_1.fastq L1_2.fastq L2_1.fastq L2_2.fastq -d
karyon_results -F L2_1.fastq
```

4.e. Plot generation

Karyon will automatically use the results of the variant calling pipeline to create the plots. Karyon allows to configure some of the parameters used for plotting:

- Window size (`-w` or `--wsize`): Defined the length of the genome pieces used for the plots. Default is 1000. Low values will generate very noisy results, while large values will produce plots with lower resolution. We recommend raising the value for genomes with lower expected heterozygosity.
- Maximum number of scaffolds to plot (`-x` or `–max_scaf2plot`). Sets a limit to the number of scaffolds to be represented in all plots that represent individual scaffolds. Default is set to 20 and will prioritize the longest scaffolds.
- Minimum scaffold length (`-s` or `--scafminsize`). Scaffolds with a length below a threshold will be ignored. If not defined, it will apply no filter. The value is a number. To exclude scaffolds shorter than 5Kbp, the command is `-s 5000`.
- Maximum scaffold length (`-S` or `--scafmaxsize`). Scaffolds with a length above the threshold will be ignored. If not defined, it will apply no filter. The value is a number. To exclude scaffolds longer than 50Kbp, the command is `-s 50000`.

Let's imagine we want to run Karyon and we want our plots to use a window size of 2Kbp and up to 50 scaffolds represented in the plots that must be longer than 10Kbp. The command will be:

```
python3 karyon.py -l L1_1.fastq L1_2.fastq L2_1.fastq L2_2.fastq -d
karyon_results -w 2000 -x 50 -s 10000
```

You can also generate the plots as a standalone script. For that, you have to call the script allplots.py, which requires the user to provide a fasta file, a bam file, a vcf file, a mpileup file and the fastq library used to create it. Of course, allplots.py has the same options for configuring the plots as karyon.py. Let's assume we are using the previous example, where we have run Karyon with the prefix "example" for the output files:

```
python3 allplots.py -f example.fasta -b example.bam -v example.vcf -p
example.pileup -l library.fastq
```

Where:

1) `example.fasta` is the genome assembly to use as reference.

2) `example.bam` is a bam file, which is a binary conversion of a sam file. The bam file needs to be sorted and indexed. For additional information about this format, check the following link: https://samtools.github.io/hts-specs/SAMv1.pdf
3) `example.vcf` is a Variant Calling Format, as generated by GATK. This file must have been generated using the `example.fasta` as reference and `example.bam` to obtain variant information. For additional information about this format, check the following link: https://gatk.broadinstitute.org/hc/en-us/articles/360035531692-VCF-Variant-Call-Format
4) `example.pileup` is a pileup file generated by the command mpileup of the package BCFTools. For additional information about this format, please check the following link: https://samtools.github.io/bcftools/bcftools.html
5) `library.fastq` is one of the libraries used to generate `example.bam` by mapping against `example.fasta`. The format must be FASTQ, but it can be compressed. In the case of paired libraries, you can use just one from the pair. For additional information regarding the format, please check this link: https://support.illumina.com/bulletins/2016/04/fastq-files-explained.html

5. How to interpret the results

5.a. Report

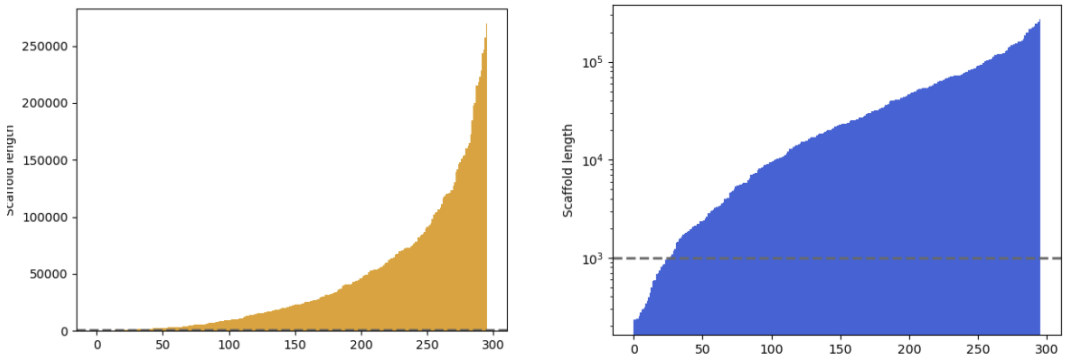
Karyon generates a report file that summarizes a series of statistics aimed to help the user to interpret genomic characteristics of the analyzed genome. The report is a text document that is organized in several sections:

- **Global stats**: Includes basic statistics regarding the assembly, including number of scaffolds, N50, N90, Busco completeness (https://busco.ezlab.org/) , GC%. It will raise a warning message if Busco completeness is low (>90%), or if GC% is too high or too low. Low BUSCO completeness might suggest that the assembly is incomplete, although this value might be naturally low for certain groups, especially parasitic lineages with reductive evolution. Taxonomic assignment is automatic (eukaryotes), and thus might be incorrect for lineages with low genomic representation. This option can be changed in the configuration file. For genomes with extreme values of GC% we recommend rerunning the whole assembly using SPAdes with the option `--sc` in the configuration file.
- **Alignment stats**: Includes statistics regarding the alignment of reads against the reference genome. These stats are derived from samtools flagstat (http://www.htslib.org/doc/samtools-flagstat.html). It includes the % of reads in the library that align against the reference and the % of reads that are properly paired. It raises a warning if either stat is low, which might suggest that the library has low quality or that the reduction step was too aggressive.
- **Reduction stats**: Reports changes in basic stats caused during the reduction step. This section is only available if the reduction step is enabled.
- **Ploidy analysis:** Reports the scores generated with nQuire lrdtest (https://github.com/clwgg/nQuire). Diploid score, triploid score and tetraploid score indicates which ploidy is the most likely for the genome. For genomes that have extensive loss of heterozygosity, aneuploidies, uneven coverage, this measurement might be inaccurate. Karyon will run the lrdtest in windows across the genome with a few modifications, and report the results. Assignment of a window to a ploidy level takes into

account the distribution of coverage across the genome, which allows to curate putative tetraploid regions (diploid score and tetraploid score tend to be very similar) and assigns haploid windows, which is not possible by nQuire. While the individual accuracy of each of these windows is relatively low, this localized analysis gives a more detailed view of the ploidy distribution across the genome and we suggest taking the most abundant ploidy configuration as the true ploidy of the genome.

- **Per scaffold analysis:** Reports all scaffolds in the assembly for which >50% of the windows deviate from the most abundant ploidy. These scaffolds are most likely aneuploid.

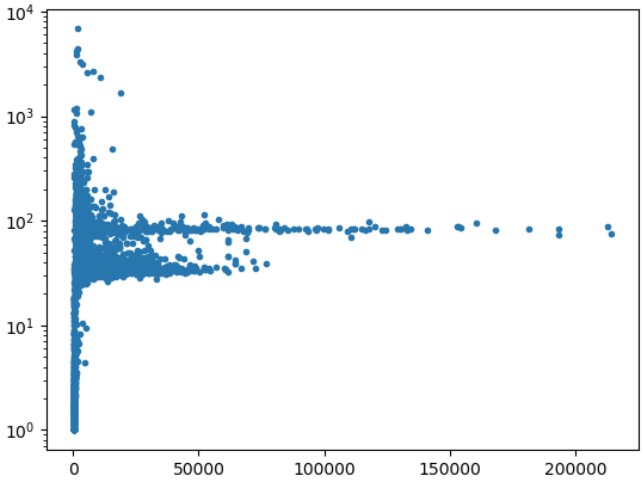
5.b. Scaffold length plots



These plots represent the distribution of scaffold lengths though a barplot, where each value represents a single scaffold versus its length, with scaffolds sorted from shorter to longest. Karyon generates the results in linear and logarithmic distribution. Very short scaffolds (shorter than 1Kbp) might introduce noise in the analyses and might be interesting to just filter them out. Dashed line is equal to window size, and can be configured with the flag `-w` or `--window_size`. Karyon will generate two plots, one named [prefix].lenlin.png and [lenlog.png], which correspond to the length distribution in linear format (yellow) and logarithmic format (blue).

5.c. Scaffold versus coverage

Karyon will generate a scatter plot representing the average coverage versus length for each scaffold in the assembly. Quite often short scaffolds have different coverage from the majority of the genome, which might be indicative of contamination or repetitive regions. Even more, long scaffolds might present a range of coverage, which might be indicative of aneuploidy. This is generated in the file [prefix].len_v_cov.png.

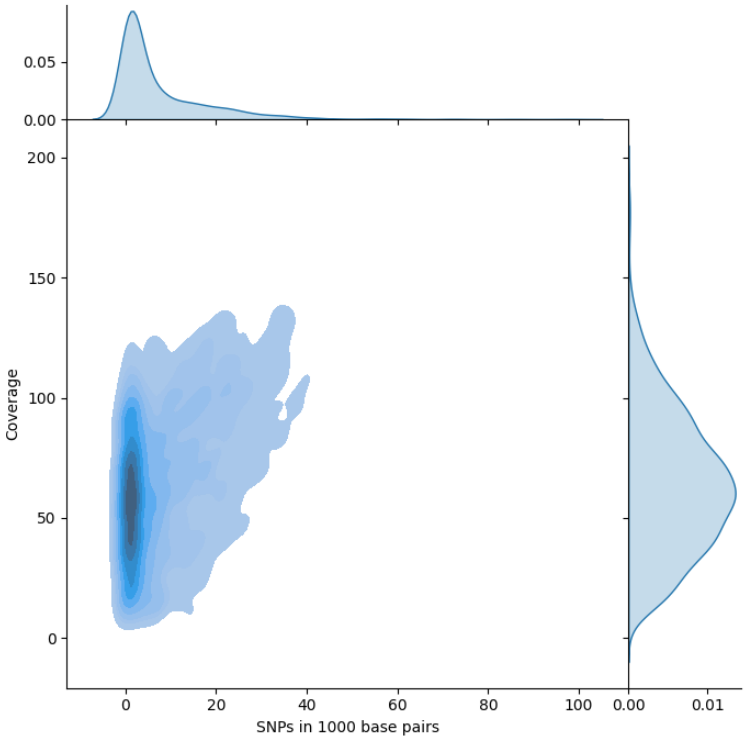


In this example, assembly is highly fragmented, but it clearly shows two populations with different coverage.

5.d. Variation vs. coverage

This plot allows the user to observe overall patterns across the whole genome. It uses a Kernel Density Estimation over a cloud of dots. Each dot represents the number of SNPs (X axis) versus the average coverage (Y axis) in a window of the genome, typically 1Kb. These dots will tend to form one or more populations. Here we compile some of the most common simple patterns:

1. **Cloud clustered towards 0 in the X axis:**
   This is the common pattern observed for a haploid or highly homozygous organism. The average Y value of the cloud corresponds to the average coverage of the genome.
2. **Cloud that diffuses across the X axis, but with a stable Y value.**
   This situation is typical of diploids and heterokaryons. Heterozygosity is rarely evenly distributed across the genome. The average Y value of the cloud corresponds to the average coverage of the genome. If part of the cloud clusters at zero heterozygosity and presents the same coverage, that represents the fraction of the genome that is homozygous, which might arise either from recombination or correspond to slowly evolving regions.
3. **Diffuse cloud across both X and Y axes.**



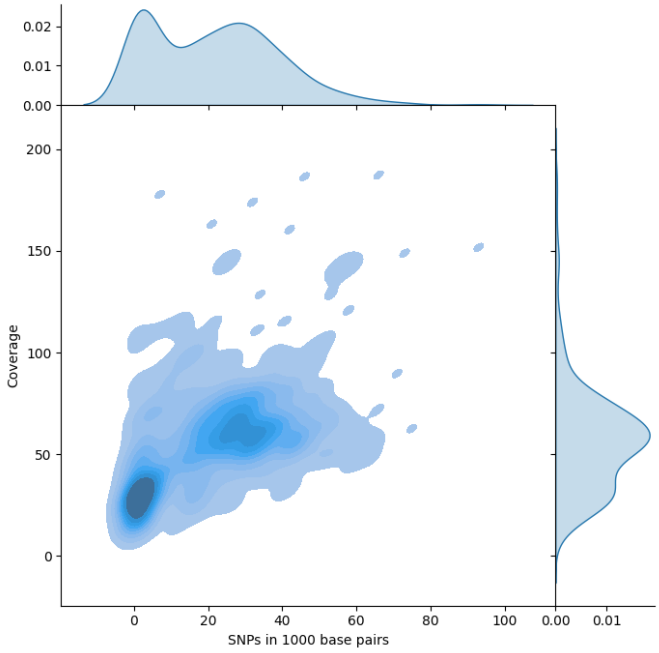
This type of distribution is often caused by the presence of contaminating sequences, especially if such contamination is heterogeneous in nature. If these patterns are observed, we recommend running BlobTools (https://github.com/blobtoolkit/blobtools2) on the sample. Alternatively, similar patterns might be caused by partial collapse of

paralog regions during the reduction step. If contamination is ruled out, we suggest changing the options for redundans and increase the identity threshold for contig removal using the configuration file (default is `--identity 0.51`). If the observed pattern is caused by an overaggressive reduction step, applying a more restrictive threshold should help.

4. **Cloud clustered towards 0 in the Y axis:**
   This pattern might also indicate the presence of contaminating sequences in the genome assembly. Contamination might affect only some of the libraries used. If the graphics are generated using one of the non-contaminated libraries, regions of the assembly that were assembled from contaminating reads will not appear. If these patterns are observed, we recommend using BlobTools to identify the contaminating libraries, and then try assembling the genome without those problematic libraries.

It is important to note that quite often more than one cloud will appear, which creates composite distributions. Here we comment some of the most common ones:

1. **Homozygous cloud + Heterozygous cloud clustered across the same Y values:**
   This pattern indicates that some regions in the genome have low heterozygosity. Depending on the relative size of each cloud, these homozygous regions might correspond to slowly evolving sites. However, if the two clouds are well defined and separated, it typically indicates loss of heterozygosity. This means that the organism is a diploid formed by the combination of two divergent parentals,that has secondarily lost regions corresponding to one or the other parental due to chromosome recombination. This pattern is common in species that formed due to hybridization events.
2. **Two or more clouds clustered at different Y values:**

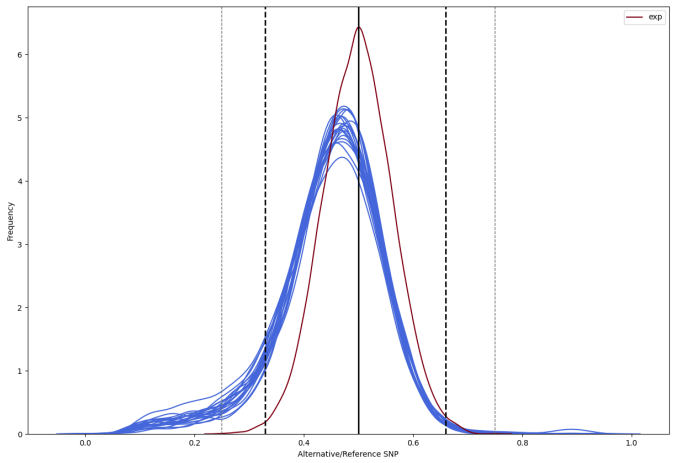


In these situations the important factor to identify is the relative Y values of these two clouds. Often, the highest Y values will be close to double or triple the smallest, which is indicative of different ploidy levels. This situation is common in aneuploids, and we recommend checking the plots per scaffold to certify that some scaffolds are driving the observed behavior. Typically, clouds with higher Y values will tend to have higher and more diffuse X values, since higher ploidy levels contain more sites that can vary. If the relative proportions of the Y peaks are incompatible with aneuploid levels, it is likely that the dual

behavior is caused by a single, highly abundant second source of DNA, such as a symbiont. Such scenarios can be identified easily with BlobTools. Alternatively, diffuse clouds with very high coverage might be indicative of a high fraction of repetitive elements.

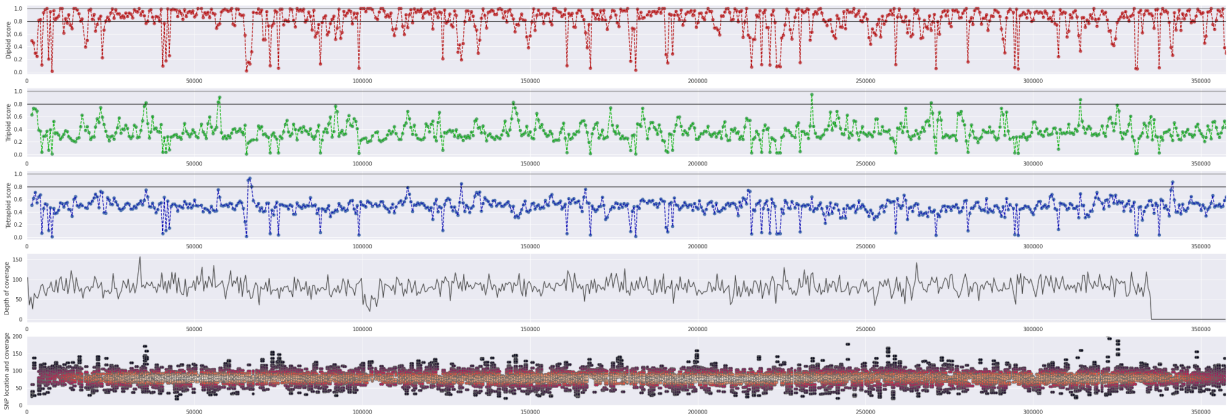
5.e. Fair coin plot

This plot represents the proportion of alternative vs. reference SNP for the whole genome and for each individual scaffold. Vertical lines indicate expected frequencies of 0.5, 0.33 and 0.25; corresponding to ideal diploids, triploids and tetraploids, respectively. An expected frequency is drawn, which is based on a per-site simulation of proportions assuming ideal 0.5 relative frequencies and random sampling equal to the coverage of the site.



In a diploid or heterokaryotic organism, most of the scaffolds will present a frequency maximum close to 0.5 and the curve will approach the theoretical ideal. Repetitive elements, stochastic calling errors (variant calling algorithms have a bias against the alternative SNP) and other methodological limitations might slightly deviate from the ideal curve. Triploid regions will present two maxima around 0.33 and 0.66. While tetraploid regions could theoretically present three maxima, at 0.25, 0.5 and 0.75, in practice the relative signal will depend greatly on the composition of the subgenomes. For example, an allotetraploid formed by hybridization of two highly homozygous diploids might be indistinguishable from a regular diploid and present most signal at 0.5. Analogously, a second generation allotetraploid hybrid between an AB hybrid and an AA homozygous will present most of its heterozygosity in the 0.25 and 0.75 mark. It is important to note that the different proportion of reads for each subgenome might cause biases against the minority subgenomes during genome assembly. Finally, some heterokaryons might present systematic deviations from the expected distributions that might be visible in this plot as slight, mostly symmetric deviations from the theoretical distribution. Detection of heterokaryosis is far from trivial, and experimental approaches might be necessary to verify it.

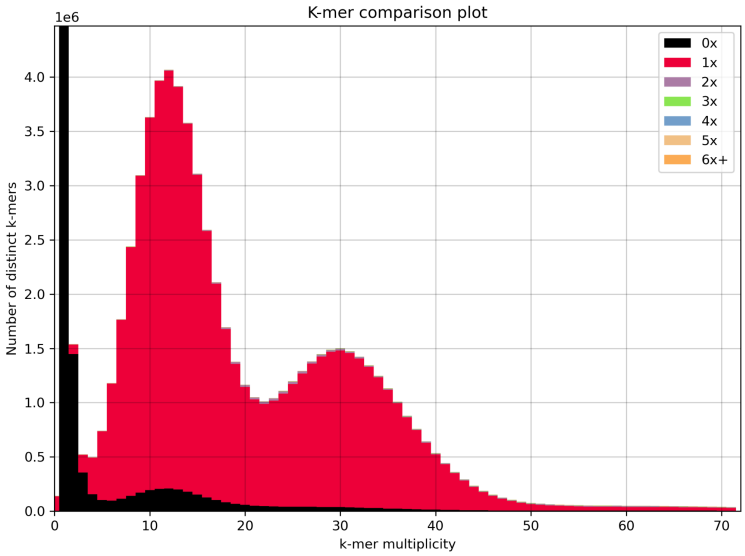
5.f. nQuire per scaffold plot



Karyon will run nQuire (https://github.com/clwgg/nQuire) across sliding windows of defined length (by default 1Kbp) across different scaffolds. The plot for each scaffold contains five subplots. The first three represent the nQuire score for diploid, triploid and tetraploid for that particular window. This allows the user to visualize patterns of aneuploidy per scaffold, especially with regards to diploid and triploid regions. The fourth and fifth subplot represents the location and coverage of SNPs across the scaffold. A color code is assigned to represent the density. Since nQuire requires information of SNPs, homozygous regions cannot be assessed and will appear as missing data.

The user can generate this plot alone by running the script `nQuire_plot.py`.

5.g. K-mer distribution



Karyon is able to call KAT (https://kat.readthedocs.io/en/latest/) to analyze the distribution of unique k-mers in the favourite library. The shape of the curve is informative about the ploidy

level of the sample and the number of peaks typically corresponds with the maximum ploidy level. One of the main factors that affect the relative size of the peaks is loss of heterozygosity, as this reduces the amount of heterozygous positions. Additionally, aneuploidies might appear as peaks if they affect a large enough proportion of the genome. When interpreting these results it is important to check the proportion of mapping reads. A low proportion of mapping reads might be indicative of incomplete assembly, which might arise due to some artifacts. Karyon will automatically generate a report using samtools flagstats. Ideally, the percent of mapping reads should be above 90%.

Due to the reduction step, k-mer distribution plots will often have a significant portion in black. This corresponds to k-mers that appear 0 times in the reference genome, which would be the genomic fraction reduced by Redundans. We can use this to have a visual representation of how much reduction has occurred in the genome, but only if the percent of mapped reads is large enough (samtools flagstats report). A low percent of mapped reads might indicate incomplete assembly or presence of contaminants (i. e. mitochondrial DNA).

6. Testing the pipeline

If you want to test the pipeline with a known dataset, you can use the following commands.

```
fastq-dump --split-files SRR974799 SRR974800
```

This will download two sequencing libraries from NCBI SRA corresponding to *Lichtheimia ramosa* B5399, one of the strains analyzed in the main publication.

If you are using docker images, first execute interactively the container, then export the SRA tools path to PATH, configure sra-tools and then download the reads.

```
docker exec -it our-alias bash

cd /src/karyon/shared/dependencies/

export PATH=$PATH:$PWD/sratoolkit.3.0.0-ubuntu64/bin

cd ..

fastq-dump --split-files SRR974799 SRR974800
```

Run the test

```
python3 karyon.py -l SRR974799_1.fastq SRR974799_2.fastq
SRR974800_1.fastq SRR974800_2.fastq -d karyontest
```