Universidade Federal do **Rio de Janeiro**
Instituto de Bioquímica Médica Leopoldo de Meis

Lampada
LAboratório Multidisciplinar Para Análise de Dados

mitoMaker 1.0

a simple script for mitochondrial genome assembly and annotation based in NGS data

Alex Schomaker
Francisco Prosdocimi

Rio de Janeiro, 2014

“The easiest way to assembly and annotate an animal mitochondria from NGS data is using mitoMaker software.”
from a mitoMaker user.

SUMMARY

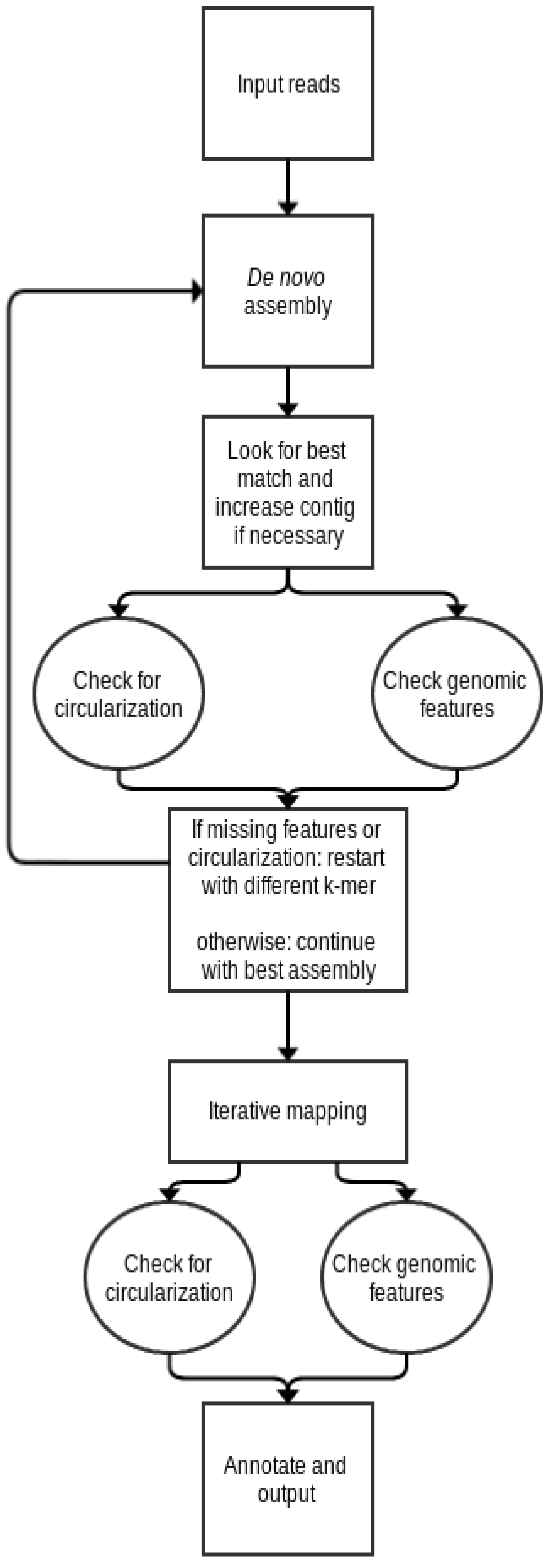
Thank you for downloading mitoMaker!
Good luck on your assembly endeavours!

1-Introduction:

mitoMaker is a pipeline script developed to simplify the assembly and automatic annotation of mitochondrial genomes, based on raw NGS reads and an optional target reference.

mitoMaker calls well known assemblers and algorithms, such as SOAPdenovo, MIRA and blast+ and parses their results providing easily readable outputs, such as FASTA, GENBANK, SEQUIN, PNG and others.

Input your animal NGS reads into mitoMaker and find your mitochondria assembled and annotated in the output files.


Input reads
De novo assembly
Look for best match and increase contig if necessary
Check for circularization
Check genomic features
If missing features or circularization: restart with different k-mer
otherwise: continue with best assembly
Iterative mapping
Check for circularization
Check genomic features
Annotate and output

1.1. How it works

Mitomaker performs a number of different steps, each one implemented in a specific python module –some of them can be called as stand-alones.

The first step is a **De-novo assembly** that uses either SOAPdenovo-Trans, MIRA (v4) or SPAdes.

After the initial assembly is done, the resulting scaffolds/contigs are locally aligned to a reference mitogenome available in the package or provided by the user. This allows the software to choose the very best mitochondria representative in the dataset.

If the best mitochondrial genome assembled is too different from the target reference, mitoMaker will then try to find other contigs (based on the Blast+ results), that belong to uncovered regions of the target genome you are trying to build. mitoMaker will then concatenate those contigs according to their relative position (the more closely related the reference given is, the more accurate this search is).

The initial mitogenome assembly is then checked to find out how many mitochondrial features (protein-coding genes, tRNAs and rRNAs) are present. If the expected features could not be found, the assembly step is called again with a different k-mer, until all k-mers supplied have been used. Then, the best build of all assembly runs is selected.

A **referece-based assembly** using MIRA (either v3 or v4) is then executed to increase the build, close gaps and prepare results for further steps.

Once the initial mapping assembly is done, a third step (MITObim) is launched. MITObim starts a recursive baiting and mapping assembly to improve the resulting assembly until no more reads can be mapped to it. Thus, a complete mitochondria is exhaustively built.

The final assembly is then checked for **circularization**, number of **genomic features found** (tRNAs, rRNAs, and genes) and annotated. In the case of a mitochondrial build, the control region (d-loop) is also searched.

A number of flags can modify different parts of the default version. Please check section 4.

1.2. What to use Mitomaker for?

Mitomaker was developed for assembling mitochondrial genomes based in NGS data (mainly Illumina), though it could also be used for general transcriptome and genome assembly, in the case the user know particular ortholog genes of interest to be found. These genes must be provided as GenBank or FASTA file. mitoMaker has also been used successfully for assembling plastid genomes.

2-Installation:

In order to fully work, mitoMaker needs a working installation of **BioPython, SOAPdenovo-Trans, MIRA assembler, SPAdes assembler, tRNAscan-SE, Blast+** and **MITObim**.

Please note that working copies of these software are provided for LINUX 64bit versions and used by default for SOAPdenovo-Trans, SPAdes, MIRA4.0, Blast+, MITObim, BioPython and tRNAscan-SE.

mitoMaker installation has been tested in a Linux environments (Ubuntu 11.04, 11.10, 12.04, 14.04 & SUSE) and should work properly as long as you have a working python 2.7, the other programs properly installed and a 64bit environment.

All programs offered by default in mitoMaker's package are **pre-compiled for Linux 64bit usage**, so you might have to download different versions in order to try it in another OS.

Minimum mitoMaker installation will require at least SOAP (or MIRA) and Blast+. The other programs can be ignored with appropriate flags. Working packages for MIRA and MITObim are given within mitomaker's folder.

There is a configuration file called **generalMaker.config**, on which one can change the installation folder for the subprograms called by the main script. Further details on how to configure this file can be found inside its code.

You can test if all subprograms and routines are working properly in your system by running the provided script ***test.py***. It will try to assemble the testcase provided in the *testcase/* folder, first using SOAPdenovo-Trans then SPAdes and then MIRA4.0. Once it is finished it will inform whether any programs failed, or if they passed all tests.

You can also look at the *testcase/* folder for an example of an input file.

Working versions for each program listed can be found on the following links:
SPAdes: http://bioinf.spbau.ru/en/content/spades-download-0
MIRA: http://sourceforge.net/projects/mira-assembler/
MITObim: https://github.com/chrishah/MITObim (v1.7 is prefered)
SOAPdenovo-Trans: http://soap.genomics.org.cn/SOAPdenovo-Trans.html
tRNAscan-SE: http://lowelab.ucsc.edu/tRNAscan-SE/
(download source-code and compile, a pre-compiled version is offered in mitomaker's folder)
Blast: http://blast.ncbi.nlm.nih.gov/Blast.cgi?CMD=Web&PAGE_TYPE=BlastDocs&DOC_TYPE=Download

You will also need a working **BioPython package**, which you can get from http://biopython.org/wiki/Main_Page if the supplied version fail for your system.

2.1. Installation information for the impatient

Simple installation instructions for LINUX users: First access http://sourceforge.net/projects/mitomaker/files/ and download the file named **mitoMaker_1.0rc4.tar.gz**

In the LINUX terminal, access the folder containing the file and type:

```
$> gunzip mitoMaker_1.0rc4.tar.gz

$> tar -xvf mitoMaker_1.0rc4.tar
```

Congratulations!
Your version of mitoMaker should already be installed!

2.2. Testing the script

Open the terminal, enter mitoMaker's folder and run *test.py* script. *test.py* will try to assemble a known mitochondria using both SOAPdenovo-Trans and with MIRA. This will test all provided features.

```
$> python test.py
```

If you get an error message, check it to see which program failed and open mitoMaker_manual.pdf for instructions on installing each program to your environment.

3. Preparing input data:

3.1. Introduction

Mitomaker will use your available reads as a starting point, which can be originated from any technology supported by SOAPdenovo-Trans and Mira 4.0, such as:

(i) Sanger;
(ii) 454;
(iii) Illumina;
(iv) PacBio;
(v) Ion-torrent

You should only use Illumina data when using SOAPdenovo-Trans mode though. It uses a simple, yet informative, input file format.

Any type of trimming or filtering should be done before running the script, although mitoMaker normally works well with standard quality data.

If you are running in MIRA mode, MIRA does a great job trimming and filtering usual Illumina sequences on it's own.

3.2. The input file:

The input file is basically a **SOAPdenovo-Trans input file**, with additional flags available.

You'll need to provide the **complete** path to your reads for all the libraries you intend on using and inform if they have qualities for them. You'll also need to inform the size of the reads you are using.

Example of input file for a MIRA De-Novo assembly:

```
$> more input.mira
[LIB]
avg_ins=300
q1=/../../../YourReads_R1.fastq
q2=/../../../YourReads_R2.fastq
technology=solexa
orientation=---> --->
```

Example of input file for a SOAP De-Novo assembly:

```
$> more test.input
max_rd_len=100

[LIB]
rank=1
avg_ins=300
asm_flag=3
q1=/../../../YourReads_R1.fastq
q2=/../../../YourReads_R2.fastq
reverse_seq=1
```

Flag description:

max_rd_len (SOAP only): reads bigger than this value will not be used (works only in SOAP mode).

[LIB]: marks the beginning of a read library (multiple libraries can be used).

rank (SOAP only): in what order should this library be assembled? (required only in SOAP mode).

avg_ins: the average insert size (if ommited, MIRA will try to auto-guess this; required for SOAP)

reverse_seq (SOAP only): This option takes value 0 or 1. It tells the assembler if the read sequences need to be complementarily reversed. Illumima GA produces two types of paired-end libraries: a) forward-reverse, generated from fragmented DNA ends with typical insert size less than 500 bp; b) reverse-forward, generated from circularizing libraries with typical insert size greater than 2 Kb. The parameter “reverse_seq” should be set to indicate this: 0, forward-reverse; 1, reverse-forward.

asm_flag (SOAP only): tells in which parts (contigation, scaffolding or both of them) should this library be used (required for SOAP, not needed for MIRA).

1 = contigation only

2 = scaffolding only

3 = both phases

q1, q2, q, f1, f2, f: the files containing the reads (1 and 2 for paired-end/mate-pair configurations). There is no need to use 1 or 2 after the 'q' or 'f' if it is not paired.

Use 'f' for fasta file without quality and 'q' for files with quality.

If you are going to assemble in De-Novo mode with MIRA, change the ending of the file from **.fasta** to **.fna** if you wish to ignore qualities at all. Otherwise, the programs will look for a **.fasta.qual** file.

Obtained and modified from: http://soap.genomics.org.cn/soapdenovo.html

orientation (MIRA only): what is the orientation of paired reads? ---> <--- / ---> ---> / <--- <--- / <--- ---> (usable with MIRA mode. If ommited MIRA will automatically guess this information).

technology (MIRA only): what kind of technology was used to make these reads. Important for MIRA.

Available technologies (as should be written in input file):
454, solexa, pacbio, iontor, sanger.

Solexa is the default value if this field is ommited.

default_qual (MIRA only): if a read presents no quality, use this quality instead of throwing an error.

3.3. The mitogenome used as reference

The user should always try to provide a reference mitogenome **as closely related as possible** to the target to be assembled.

This will allow the accurate comparisons and annotations. If the user cannot find a good reference, we provide some for various phylogenetic groups in the *references/* folder.

A reference mitogenome is required in two steps: (i) when checking for a possible mitogenome in the De Novo assemblies and, (ii) when annotating and performing feature checking and automated annotation (should be provided in GenBank format).

If the user cannot provide a mitogenome annotated in GenBank format or in the case one has only a FASTA reference, mitoMaker can check for the features and annotations based on a provided reference with the flag -o (explained in more detail in Section 4).

If the user absolutely does not want to use any reference, a range of complete mitogenome sizes can be defined for searching. For example, if the user wants to build a vertebrate mitochondria, he/she can tell mitoMaker to filter scaffolds of sizes varying between 500 and 18000 and get the closest one to 16000, as common in these organelles from vertebrates.

The user can also tell the script which annotated molecule should be used as a reference with the -r flag. The usage is *-r* *reference_file*. If it terminates in *.fasta* it will be interpreted as a FASTA sequence. Otherwise, it will be interpreted as a GenBank file (so make sure it is properly formatted to your needs).

4 – Flags used to alter the mitoMaker behavior

4.1 – The -j flag:

This flag names the main job, basically telling the script what is the name that should be used for assemblies, sequences and read strains.

Usage: -j test

4.2 – The -r flag:

Tells the program which file should be used as reference mitogenome. If the file is given with *.fasta* extension, mitoMaker will assume it to be a FASTA formatted file sequence, otherwise it will be considered as a **Genbank flat file**. Thus it is important to make sure you have a properly formatted file.

If the user provides a genbank reference, it will be used during the BLAST steps when trying to find the best assembly matching the mitogenome reference, and also during the automatic annotation step. If the file provided is a FASTA reference, a default Genbank reference will be used during the automatic annotation according to the -*o* flag.

When using a genbank reference, please make sure the *CDS* feature for the target genes of interest are provided.

Usage: -r my_reference.gb

4.3 – The -i flag:

Tells the script which file should be used as the input file (please read section 3 for more info about the input file).

Usage: -i my_input.input

4.4 – The -k flag:

This flag will describe a list of k-mers that should be used when trying to find the best assembly possible during the De-Novo assembly phase. It can be provided in any order and need to be separated by commas.

Usage: -k 53,31,71,23

Special case: When using MIRA mode for De-Novo assembly, you can specify the *default* value in the k-mers list. This value means MIRA will run the assembly with its default values, which varies according to the technology being used (see section 3.2 for more info).

Example: -k 31,default,71

4.5 – The -p flag:

Controls how many threads mitoMaker can use at maximum, when multi-threading is available for the assembly programs. Higher values usually mean faster assemblies. Before using this flag, please certify with your system administrator how many cores are available in your machine and how many you can use.

Default value: 4

Usage: -p 8

4.6 – The -l flag:

Contigs lower than this size will not be used during the scaffolding phase of SOAPdenovo-Trans.

Default value: 100

Usage: -l 300

4.7 – The -op flag:

When the user is not using any reference mitogenome, mitoMaker grabs the contig/scaffold that is closest to the *-op* value.

This is important if the user absolutely do not have any reference to use, or in the case the user is assembling a very specific data to a given study on which sequence size may be used as a good indicative of a proper build. If the user does not specify this, the script will try to auto-guess it based on other parameters.

Usage: -op 16000

4.8 – The -c flag:

After the *de novo* assembly, scaffolds/contigs lower than this size will be filtered out when searching for a best match to the target.

Usage: -c 500

4.9 – The -mc flag:

During initial filtering of scaffold/contigs, assemblies bigger than this value will be filtered out.

Usage: -mc 19000

4.10 – The -e flag:

e-Value cutoff to be used by Blast when trying to find a build that resembles the mitogenome reference provided. Lower values will make the search more strict.

Default value: 10.0

Usage: -e 5.0

4.11 – The -ceq flag:

The percentage (in decimals) of similarity used during feature check to conclude that a feature has been really found when comparing to the Genbank reference. Lower values will make this check more relaxed.

Usage: -ceq 0.45

4.12 – The -o flag:

This flag should be set to tell mitoMaker what kind of organism (genetic code) you are trying to assemble. It is absolutely required for gene annotation step. By default, mitoMaker uses 2 (vertebrate mitochondria). The possible values are the same of NCBI's blastx table.

In case you are not using a Genbank reference, the program will get a default reference based on this value. **Thus, it is deeply recommended that you use a genbank reference as close as possible to your target DNA.**

Usage: -o 2 → vertebrate mitochondria (default)

NCBI's table:
1. The Standard Code
2. The Vertebrate Mitochondrial Code
3. The Yeast Mitochondrial Code
4. The Mold, Protozoan, and Coelenterate Mitochondrial Code and the Mycoplasma/Spiroplasma Code
5. The Invertebrate Mitochondrial Code
6. The Ciliate, Dasycladacean and Hexamita Nuclear Code
9. The Echinoderm and Flatworm Mitochondrial Code
10. The Euplotid Nuclear Code
11. The Bacterial, Archaeal and Plant Plastid Code
12. The Alternative Yeast Nuclear Code
13. The Ascidian Mitochondrial Code
14. The Alternative Flatworm Mitochondrial Code
16. Chlorophycean Mitochondrial Code
21. Trematode Mitochondrial Code
22. Scenedesmus obliquus Mitochondrial Code
23. Thraustochytrium Mitochondrial Code
24. Pterobranchia Mitochondrial Code

25. Candidate Division SR1 and Gracilibacteria Code

4.13 – The -d flag:

During automatic annotation, a region with at least *-d* size that does not match any gene, rRNA or tRNA will be annotated as D-Loop. User should set this variable mainly if he/she is not building a mitogenome.

Default value: 900
Usage: -d 1300

4.14 – The --chloroplast flag:

Tell the script you are trying to build a chloroplast. This will automatically change some default behaviors and, if a reference is not given, will tell the automatic annotation script to use the base chloroplast reference located in the references/ folder.

4.15 – The --skipmitobim flag:

Tells the script to skip MITObim phase. Should be used whether the user just want to go as far as the mapping phase after the De-Novo assembly.

Default value: False
Usage: --skipmitobim

4.16 – The --relaxed flag:

When turned on, mitoMaker will run neither automatic annotation (geneChecker) nor tRNAscan-SE.

Default value: False
Usage: --relaxed

4.17 – The --skiptrna flag:

Makes the program skip running tRNAscan-SE checks. Turned off by default.

Default value: False
Usage: --skiptrna

4.18 – The --nocopykmers flag:

This flag makes MIRA **DOES NOT** use the same k-mer in the mapping phase as the best result from the De-Novo phase. It uses therefore MIRA standard kmer size.

Default value: False
Usage: --nocopykmers

4.19 – The --mitobimiter flag:

This flags tells the script how many MITObim iterations to run at maximum.

Default value: 20
Usage: --mitobimiter 30

4.20 – The --mapmira3 flag:

Tells mitoMaker to use MIRA3.4 instead of MIRA4.0. Not recommended, but in case you really want to use the older MIRA version, you can turn this flag on.

Usage: --mapmira3

4.21 – The --miratech flag:

Required if using --mapmira3 and not using a solexa read library. See section 3 for more technology options.

Default value: solexa

Usage: --miratech 454

4.22 – The --miraest flag:

When this flag is turned on, MIRA will run in EST mode during the De Novo phase. Please refer to MIRA Manual for more information: http://mira-assembler.sourceforge.net/docs/DefinitiveGuideToMIRA.html#chap_est

Usage: --miraest

4.23 – The --recursivemira flag:

This flag tells mitoMaker to use MIRA4.0 in the De Novo phase instead of SOAPdenovo-Trans. MIRA will work differently, aside from various software level differences, in the sense that it will make a De Novo genomic assembly and not a De Novo transcriptomic assembly. Both should work fine for building mitochondria, but our usual approach using SOAPtrans tends to be faster and works well enough for building mitochondria and chloroplasts, since they are present in high levels in the read libraries.

See section 3 for information on what might change in the input file when using this mode.

Usage: --recursivemira

4.24 – The --noextension flag:

If turned on, mitoMaker will not try to extend the best DeNovo assembly by concatenating complementary contigs/scaffolds.

Usage: --noextension

4.25 – The -cove flag:

Defines the cutoff value to be used by tRNAscan-SE. Default is 7. The higher the value, the more strict becomes the tRNA serch perfomed by tRNAscan-SE, meaning it could skip real tRNAs. On the other hand, if it is too low, you could start to get false positives.

Usage: -cove 10

4.26 – The --bacteria & --archea flag:

These flags should be used whether the user is trying to build a bacterial, or archea, target DNA, such as a specific gene of *E. coli* or the whole genome. This will change the behavior of programs such as tRNAscan-SE and blastx so that they look for features based on the appropriate genetic code.

If the user is not providing any genbank reference, this flag will also choose one of the default references in the *references/* folder, accordingly. We advice that, although mitoMaker can build and annotated bacterial genomes, it was not developed for it.

Usage: --archea
--bacteria

4.27 – The --circularsize flag:

Tells mitoMaker how big should the circularized region be for it to be considered a proper circularization.

Example: If a region at the end of the sequence blasts against the beginning of the sequence, how large should it be?

Default: 45

Usage: --circularsize 40

4.28 – The --skipdenovo flag:

Tells mitoMaker to skip the DeNovo assembly step. In order to do so, the user will need to provide as argument to this flag a *fasta* (.fasta or .fa) or a *genbank* file containing an already built sequence: *--skipdenovo MyAssembly.fasta*

The program will consider the first k-mer in the *-k* flag as the k-mer used to assemble the provided sequence. So remember to change the *-k* flag if it is needed.

For example, if you have already done a *de novo* assembly using a different program than the ones provided by mitoMaker, the user can give that sequence fasta file as argument to this flag and mitoMaker will consider it as the reference for the mapping phase.

4.29 – The --forcedenovo flag:

Tells generalMaker.py to use SOAPdenovo, even when doing a mitochondrial or chloroplast build. By default, mitoMaker and chloroplastMaker will use SOAPdenovo-Trans.

When running with SOAPdenovo the default k-mers change to 23 and 31.

Usage: *--forcedenovo*

4.30 – The --soaptrans flag:

Tells generalMaker.py to use SOAPdenovo-Trans. Turned on by default when using mitoMaker or chloroplastMaker. If --forcedenovo is on, this will be ignored.

4.31 – The --spades flag:

Tells generalMaker.py to use SPAdes for genome assembly, without error correction. Any read error correction should be done prior to running mitoMaker.

4.32 – The --keepfolders flag:

Tells generalMaker.py to keep temporary folders created during assembly. Such as the various k-mer De Novo assembly folders, the MIRA mapping folder and the last two MITObim iteration folders.

4.33– The --help flag:
Displays the help menu for mitoMaker.

5 – Running the script:

In order to execute mitoMaker, the user will need to run only the main script, ***mitoMaker.py***, which can be given proper permissions to be run as an executable. Otherwise, the user should call it through the python interpreter.

The user **should not** change the flags available, once we chose the best set of parameter to be the default set.

But in case the user is trying to build something very specific, he/she might need to play around with them. Please read section 4 carefully for information on each flag available or access most of the available parameter running the script with –help.

Two examples of simple usage:

5.1. Generic command line example

```
>python mitoMaker.py -j test -i test.input -p
8 -r my_ref.gb
```

This command line will run mitoMaker with job *test*, using test.input as the input file (please read section 3 for more info on how to build this file) using 8 threads whenever possible during assembly programs.

5.2. Another command line example

```
>python mitoMaker.py -j test -i test.input -p
64 -r my_ref.gb --skipmitobim --skiptrna
```

This command will run mitoMaker with job *test*, using test.input as the input file, using 64 threads whenever possible. It will also skip MITObim phase and the program tRNAscan-SE.

6 – Results and output files

Inside the main folder the user chose to run the job, various files will be created and a final **result** folder. Here's a summary of the most important files created:

6.1. 'your_job_name'_Final_Results folder:

This folder will hold the most important results of mitoMaker run.

6.1.1. The genbank annotated file:

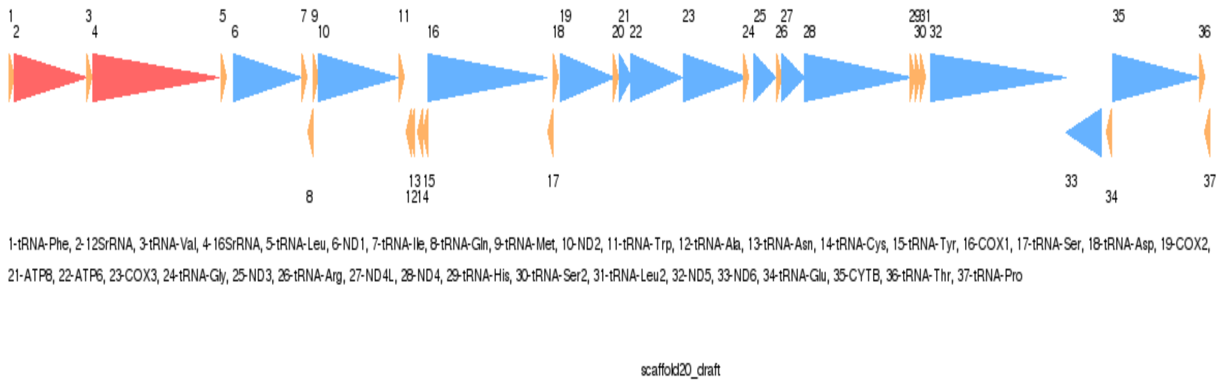
Contains your mitochondrial genome completely built and annotated.

In the case all features have been found, mitoMaker will also provide an **ordered genbank file** that starts with the *tRNA-Phe*, as common in the description of mitogenome.

The program will also create a *.tbl* file, which is in the Sequin format, ready to submit to GenBank.

We encourage the user, however, to perform careful manual curation in the finished sequence provided by mitoMaker. The software is provided AS IS and we will not be responsible for any mistakes happening in the mitochondrial assembly and/or annotation. In our experience, most mitogenomes are very well annotated by mitoMaker, but we cannot be sure what is going to happen with spurious, contaminated data and/or highly divergent sequences.

6.1.2. The png annotation image file:


1-tRNA-Phe, 2-12SrRNA, 3-tRNA-Val, 4-16SrRNA, 5-tRNA-Leu, 6-ND1, 7-tRNA-Ile, 8-tRNA-Gln, 9-tRNA-Met, 10-ND2, 11-tRNA-Trp, 12-tRNA-Ala, 13-tRNA-Asn, 14-tRNA-Cys, 15-tRNA-Tyr, 16-COX1, 17-tRNA-Ser, 18-tRNA-Asp, 19-COX2,
21-ATP8, 22-ATP6, 23-COX3, 24-tRNA-Gly, 25-ND3, 26-tRNA-Arg, 27-ND4L, 28-ND4, 29-tRNA-His, 30-tRNA-Ser2, 31-tRNA-Leu2, 32-ND5, 33-ND6, 34-tRNA-Glu, 35-CYTB, 36-tRNA-Thr, 37-tRNA-Pro
scaffold20_draft

Contains a graphical representation of the mitogenome built.

6.1.3. Other files:

A FASTA file containing the complete sequence of the mitogenome built.

We provide a *maf* and *caf* files of the resulting sequence alignment. These files can be used to check for sequence coverage in each part of the assembled molecule and to provide general verification of the assembly.

There will also be a *.stats* file with some basic statistics, such as: GC%, length, length without Ns, evidence of circularization, number of features found, missing features and features split or frame shifted.

6.2. best_query.fasta

The contig or scaffold that was considered the best hit against your reference target and had a minimum percentage of its size spanning the reference. This was used as a reference for the mapping phase.

6.3. all_hits_except_best.fasta

File containing all contigs and scaffolds assembled except the best_query.

6.4. circularization_check.blast.xml

XML result of a blast of the last build assembled (if de novo phase is done, it will be from the best_query result; if mapping has been done, it will be from the mapping result, etc) against itself to check for edges matching as a parameter to verify if the build is complete and circular.

6.5. important_features.fasta

File containing the sequences of all features extracted from a the Genbank reference provided. This is used during the feature checking phase and automatic annotation.

6.6. possible_hits.fasta

Contigs and scaffolds filtered after the De Novo phase according to the -c and -mc flags (see section 4 for more information).

6.7. 'your_job_name'.draft.fasta

Non-annotated semi-final result. Basically this is the sequence generated by the last MITObim iteration or the mapping phase (if you used --skipmitobim).

6.8. Other files and folders

At the end of each run, mitoMaker will create various folders and files for each phase, including: (i) folders for each k-mer used, with logs from the main assembly program used (MIRA or SOAPdenovo-Trans), (ii) a copy of the input file used by that program and its various possible results (such as assembly statistics, debri files and the actual contigs and scaffolds built).

There will also be a folder created for the mapping phase, on which the user will find the input file used, the sequencing reads and a MIRA log file. The user will also find MIRA's folder containing assembly statistics, information and results in various formats.

Moreover, folders containing each MITObim iterations will be created, including also assembly results and informations. If the user happen to run mitoMaker again from within the folder where these MITObim folders are, the files will be deleted and overwritten, as it is a requirement for MITObim to run properly.

7 – Standalone modules

The script for circularization checking and for automatic annotation used by mitoMaker can also be called as standalones for expert users. This will be helpful if the user has some assemblies already performed that need to be annotated. Or in the case the user you prefer to use another method to assembly the reads.

7.1. circularizationCheck.py:

This script is used to check if there is evidence of circularization in a given assembly. In order to call it as a standalone, the user will need to provide the FASTA file on which you want to check for circularization and the script will print the results as a python tuple:

Usage: python circularizationCheck.py fasta_file
Outputs: (True, x_coordinate, y_coordinate) or *(False, x_coordinate, y_coordinate)*
x and y coordinates mean the starting position and ending position of circularization, respectively

7.2. geneChecker.py:

This script should be called to assess the presence of features into a FASTA file. In order to call it as a standalone, the user will need to provide a Genbank reference file, a FASTA file with the sequence you want annotated and the output file wanted.

Usage:

```
$> python geneChecker.py genbank_reference
fasta_file outputfile_wanted organism_type
```

```
$> python geneChecker.py ancistrus.gb
ancistrus_2.fasta ancistrus_2.gb
```

```
$> python geneChecker.py ecoli.gb
my_target_ecoli.fasta target.gb –o 11
```

Outputs:

a genbank formatted file with your final build named as *outputfile_wanted*

a Sequin formatted file, with the *.tbl* terminology

In the case all features have been found, the script will also create an ordered genbank file starting with tRNA-Phe and named as *outputfile_wanted.ordered* and its respective *.tbl* file.

8 – Citation

Rio de Janeiro, September 2014

Thanks for using **mitoMaker!**

Please cite us:

Schomaker and Prosdocimi, 2014.

mitoMaker: a pipeline for automatic assembly and annotation of animal mitochondria in partial genome sequencing projects. *Submitted to Bioinformatics Journal*.