Wrocław University of Environmental and Life Science
Faculty of Biology and Animal Science
Major: Bioinformatics
Full-time first degree

Paweł Tomkowski
121562

Creation of a tool for automatic assembling of mtDNA sequences from NGS using genome sequences of snails from genus Trochulus

Work created under the supervision of
PhD. Tomasz Strzała
Department of Genetics

Wrocław, 2022

Abstract

Creation of a tool for automatic assembling of mtDNA sequences from NGS using genome sequences of snails from genus Trochulus

As part of the project a tool has been created to allow more scientists access to recreation of mtDNA from readily accessible data. Gathering already existing programs and tools a script was created that quite effectively atomically processes Ilumina type NGS data to mitochondrial sequence.

Keywords: NGS, mtDNA, Trochulus, Bioinformatics, bash

Table of content:

1. Introduction.

The Human Genome Project and the technologies that came with it showed that large-scale sequencing of even large genomes with good accuracy was possible. In the years that followed, various companies and organisations took up the concept, creating ever faster, more efficient and, most importantly, cheaper sequencing equipment and techniques. Thus was born NGS - Next Generation Sequencing. NGS is an umbrella term for a collection of devices and technologies that allow the sequencing of an entire large genome (animal/plant) in a matter of days or even a few a day in the case of the most advanced devices and technologies[1]. This has become possible mainly due to the considerable automation and miniaturisation of the entire process. The most famous representative of this type of technology, and the benchmark to which all others have been compared, is the Illumina equipment. The main limitation of this technology is that the readout takes place in small elements (up to 200 base pairs), which must then be assembled into a complete genome, which, due to a large number of repeats of all kinds in eukaryotic genomes, makes it difficult to work with this data. One of the best solutions has been to sequence the same material several times to achieve an overlap between each short read. Over the last few years, new technologies have emerged that allow for much longer reads, mainly from Oxford Nanopore. The reads of their devices average several thousand base pairs in length, which solves many of the problems associated with short reads. The most advanced versions even allow reads over 2 million bp in length[2].

Over time, it has also been realised that since we are sequencing the whole genome from a eukaryotic cell, there will also be a mitochondrial genome and a chloroplast genome (in the case of plants) among the data obtained. Using certain algorithms, it is possible to attempt to reconstruct these genomes from whole cell DNA sequencing. Data from these genomes are valuable because they have a different structure and therefore evolve differently from and relatively independently of the nuclear genome (faster). This allows them to be used for systematic analysis of different organisms. Specialised tools have therefore emerged to reconstruct the mitochondrial genome from the overall sequence, however, these usually require appropriate steps before they can be used in sequencing machine data.

There are protocols[3], which describe step by step how existing tools, mainly MitoFinder and NOVOplasts, can be used to extract mitochondrial genome information from whole genome sequencing. They inspired the development of this tool, which would automate this process to make this type of analysis more accessible to more scientists. The script developed will be applicable to people who have less experience or skill in dealing with programming. I hope it will increase our knowledge of mitochondrial genomes, their evolution and the systematics of organisms.

2. Materials and methods

The data used in the project were reads of the genome of the snail from the genus Trochulus made by CeGaT GmbH. The library was created using the TruSeq DNA PCR-Free kit (Illumina). Reads were performed with a NovaSeq 6000 sequencer, using a 2 x 150 bp flow cell type. Q30 value ≥ 88.25%. Adapter sequences were trimmed using Skewer (version 0.2.2). No read quality-related clipping was performed. This gave two paired files totalling 798,324 thousand reads, 118,944 million base pairs.

The following programs and scripts were used in the course of the analysis:

- Linux-type system - despite the convenience and familiarity of Windows, in most specialist applications this system doesn't have needed functionalities, a Linux-type system is used, which have them and thus most specialist software is designed for it;
- bash - the shell of UNIX systems, an intermediary between the Linux system and installed programs, also the default language for controlling everything on the system, therefore very convenient for writing scripts to control other programs, most of the software is written in it;
    - sudo apt-get install - a function used to install programs mainly from online repositories, always with the 'sudo' option giving administrative privileges to be able to make changes to the system,
    - mkdir - creates a new directory with the specified name, in the specified location,
    - cd - changes the current directory to another directory, which changes the reference location of certain commands,
    - git clone - creates a copy of the specified repository from the github platform and downloads it to the specified folder,
    - sed - a function used to replace certain characters with other characters in text files, not very efficient on a large scale,
    - function - creates functions that can be called later
    - echo - a function that shows what it has been passed, used to display information or pass text for other functions,
    - grep - a command used to search for and display lines with a matching pattern in the text, in the program used with the -c option to specify only the number of lines with a matching pattern.
- AWK - an interpretable language created and used for various types of operations on text, mainly searching and processing specific patterns;
- cURL - a function for sending and receiving files from the Internet using the URL syntax;
- Perl - one of the programming languages, a lot of bioinformatics software has some scripting or is written entirely in this language;
- docker - a program that facilitates the inclusion of different programs by creating and including so-called containers - self-contained miniature operating systems with all the necessary functions and programs needed to activate the original program; this allows different programs to be used regardless of the existing operating system or installed functions on the system;
- fastqc[4] - quality control tool for sequencing data; included in the script as a useful tool to check the quality of the data you have;
- MitoFinder[5] - a program to extract mitochondrial data from next-generation sequencing; it was used because it was in the original protocol;
- NOVOplasty[6] - whole-genome organellar genome reconstruction program; was used because it was in the original protocol;
- fastp[7] - a program to clean and simply process sequencing files; it was used because a fast program was needed to clean sequencing data;
- seqkit[8] - a package for analysing and processing files from sequencing; it has been used, for rapid file size analysis;

- MITObim[9] - a program for reconstructing the mitochondrial genome from whole-genome sequences, in the program used in the docker; used by A.Hiley because it has a downsample.py function that allows sampling of data; in the script the program was added as another one for mtDNA extraction;
- BBmap[10] - a program for the global alignment of DNA and RNA sequences; in SUNmito.sh only a script is used that allows the file in which the paired reads have been stored together to be unspliced, this is used so that the analysis is carried out on two paired files;

Explanation of program operation:

The program is run via the command line in a Linux bash shell by typing the command ./SUNmito.sh with the appropriate variables. Link to the repository where the program is located: https://github.com/Jenlotis/praca_lic. The general command to run the program: $ ./SUNmito.sh -A -i path/to/folder -m path/to/reference/mitofinder.gb -n path/to/reference/novoplasty.fasta -b path/to/reference/mitobim.fasta -O number_of_organism_by_mitofinder -r amount_RAM

For the full run you will need to:

- specify the path to be used(-M MitoFinder, -N NOVOplasty, -B MITObim, -A all 3) this sets the value of the "alpha" variable as M/N/B/A depending on the path selected and it is the value of this variable that determines the path to be run(in this way the program flag can be anywhere);
- specify the full path to the folder (-i) with sequences in compressed fastq format (.fastq.gz extension) the program automatically searches for all files that meet these conditions and adds them to the list that will be used to run subsequent subprograms;
- specify the full path to the reference sequences (-m for MitoFinder (genbank format), -n for NOVOplasty (fasta format), -b for MITObim (fasta format)), which the programs will use to try to reconstruct the mitochondrial genome from, the program using the "grep" function with the "-c" flag checks the number of lines with characteristic marks for the required formats ("LOCUS" for genbank format and ">" for the fasta format) if the program finds one such line the file is considered compatible with the format (even if the file gets through this filter it will still be rejected when the assembly program is called, but even such a basic filter will get rid of simple errors more quickly);
- when using MitoFinder, you also need to specify the type of organism (-O) (and thus also the genetic code) from which the data are from, a full list is in the help for MitoFinder, for the model data it was 5 for invertebrates;
- amount of RAM to be used by NOVOplasty (-r), it is not directly needed but the authors suggest using it when you do not have the server amount of RAM (data size +~10%);

the other flags are:

- help (-h) information about the program and the meaning of flags, is a small function that is run when a flag appears in a program call, that shows text in the terminal. After the text is shown, the program is closed;

- paired (-p) whether sequences are paired or unpaired (T(true)/F(false)), determines what subprograms will be called and how files from the input folder will be processed:
    - if the T option is specified, a variable that contains the names of the files in the folder is created by extracting the names of all the files in the specified directory using the "ls" command. This data is passed to the AWK, which searches for all files containing ".1.fastq.gz" in their name (this was the form of the file on which SUNmito.sh was tested, and probably how most of the paired reads will look like), this list is passed to another AWK, which discards the previously mentioned suffix (software assumption: the file is called NAME.1.fastq.gz if this NAME contains dots in it, the program will not be able to read this file correctly), so that the NAME itself remains, which is then passed as the main variable to the programs;
    - if the F option is specified, the AWK looks for all files containing ".fastq.gz", as this is a broad search - it will also find files used with the T option, it needs to reject them. This is done by passing the data to the next AWK, which checks if the file has a value of 1 or 2 between the NAME and ".fastq.gz" if so this file is discarded, and the next AWK leaves the NAME alone, which is then passed as the main variable to the programs;
- installs (-Z) all required programs and downloads all required repositories from the GitHub platform, this is created as a function that is run when the flag appears in the program call. The command used is "sudo apt-get install --assume-yes" - as super user/administrator install/update a package from a known repository, assume the user agrees(the command after finding a package asks the user if they are sure they want to install it, to make usage smoother this statement is used):
    - fastqc,
    - curl,
    - using the curl library, access to the brew repository is added,
    - seqkit is downloaded from the brew repository (no sudo required due to the difference in how seqkit will be installed),
    - fastp,
    - libidn11 - libraries used by MitoFinder, on newer versions of Linux they are not installed by default,
    - Perl interpreter is needed for NOVOplasty as the program is written in this language,
    - Python - many of the scripts in the programs used are written in Python so it is updated to the latest versions,
        - python2 - required by many scripts, newer Linux installations do not have it,
        - pip - package installation manager for Python,
        - python3 - the latest version of Python that exists,
        - python3-specific pip,
    - cmake - an open tool to help compile programs,
    - git - an open version control system and a lot of tools to help with this, in the program used as a 'clone' to download a copy of the repository with the required programs from the GitHub platform,
    - mawk - one of the interpreters for AWK,
    - BBmap,
    - automake autoconf - two packages responsible for making program compilation easier and simpler,

    - mkdir ./github - creates a folder named github in the current working folder,
    - cd ./github - changes the working folder to the newly created folder,
    - git clone - retrieves the repository from github to which the link has been provided,
        - MITObim,
        - NOVOplasty,
        - MitoFinder,
            - with the command "cd" the working folder is changed to the one where MitoFinder is located, "./instal.sh" installs this program, and "p=$(pwd)" "echo -e "\n#Path to MitoFinder image \nexport PATH=$PATH:$p" >> ~/.bashrc" "source ~/.bashrc" adds the program to the PATH variable, which contains the paths first checked when the bash command is invoked,
    - closes the program after everything has been done;
- number of cores(-t) that fastp and seqkit can use, if the flag is not used the programs will use their default values.

Paths are collections of different functions/subprograms gathered together to allow a fairly modular use of functions for each recognised file in a folder:

1. MitoFinder
    - pair end
        - clean_pair
            - using a conditional statement, the presence of cleaned files for this data is checked,
                - if not, they are created and cleaned using fastp,
                - if yes, the step will be skipped;
            - the "$input$i" structure allows the automatic reading of matching files from a folder; "-i" and "-I" are the input files, and "-o" and "-O" are the already cleaned data, which are saved to a separate folder for greater clarity;
            - the script uses the default values associated with file cleansing;
                - depending on the data, these values will be different and it is difficult to determine what values will be required for a particular file;
        - downsam_p2p
            - function completely created for MitoFinder, in the future, it will be possible to generalise its functionality;
            - the role of this function is to create a sample of the data using the downsample.py script from the MITObim package, and then, because the script saves the sample to a single file, to deinterleave the data using the reformat.sh script from the BBmap package;
            - The subprogram uses a conditional statement to check whether the input file samples already exist,

        - if so, it checks with the ls function(and some formatting with grep and AWK) which percentages have been sampled, then informs the user of the existing files and asks whether the user wants to use one of the existing files or create a new one.
            - if they want to use an existing file, a list of all existing versions is shown, after which the user is asked to enter one of the listed values. Once this is selected, the program saves it as a variable that can be used by subsequent programs;
            - if they want to create a new file, the program proceeds to sampling;
        - If not, it goes directly to sampling;
    - MitoFinder does not perform optimally when it has to analyse more than 7,000,000 paired reads. Therefore, a necessary step is to create a sample from the cleaned data using the appropriate function, but before this can be done it is necessary to check what sample size is needed:
        - this is done using seqkit, which calculates basic statistics from one of the files that have already been cleaned,
        - statistics are passed to AWK, which extracts the length of the file from the table,
        - because the format used by seqkit has commas as thousands separator, sed has to be used to replace all commas with nothing, allowing arithmetic operations to be performed on the data,
        - using AWK, the extracted value is compared to the maximum and converted to a percentage. The program tells the user what percentage came out of the analysis, then gives a recommended value (rejecting decimals as the value taken by the next program can only be an integer and rounding, especially with a large amount of data, can deviate greatly from the maximum value) and asks what value the user wants to use. Once the value has been entered, the program informs the user of the value they have entered and proceeds to the next step.
    - using the downsample.py function from the MITObim package, a data sample of the size specified by the user is taken from the cleaned files. As the output of the function is only one file, the '--interleave' option is used to save it correctly, it is passed to the gzip function which will compress it to save space,
    - using the reformat.sh script from the BBmap package, the file created in the previous step is deinterleaved into two separate paired files; the "int=t" flag ensures that the file will be interpreted as two files mixed together,
- mitfi_pair
    - once the data has been properly prepared, the MitoFinder itself can be run by entering these flags:

        - project name "-j" (for easier identification) is the file name along with the sample percentage, "-1" and "-2" are the deinterleaved files from the previous subprogram, "-r" is the reference file (as mentioned earlier in genbank format), and "-o" the organism type entered when starting SUNmito.sh, the output file which is the most important in this script is "[Seq_ID]_mtDNA_contig.fasta" which contains the reconstructed mitochondrial genome in fasta format, the other output files are: information about the build of the reconstructed sequence and what information was used to reconstruct the sequence;
- single end
    - clean_sing
        - using conditional statement, the presence of a cleaned file for this data is checked,
            - if not, it is created and cleaned up with fastp,
            - if yes, the step will be skipped;
        - the "$input$i" structure allows matching files to be automatically read from a folder,
            - "-i" is the input file and "-o" is the already cleaned data which is saved to a separate folder for greater clarity;
        - the script uses the default values associated with file cleansing;
            - depending on the data, these values will be different and it is difficult to determine what will be required for a particular file;
    - downsam_s2s
        - function completely developed under MitoFinder, in the future, it will be possible to generalise its functionality;
        - the role of this function is to create a data sample using the downsample.py script from the MITObim package;
        - The subroutine uses a conditional statement to check whether a sample of the input file already exists,
            - if yes, it checks with the ls function(and some formatting with grep and AWK) which percentages have been sampled, then informs the user of the existing files and asks whether the user wants to use one of the existing files or create a new file.
                - if they want to use an existing file, a list of all existing versions is shown, after which the user is asked to enter one of the listed values. Once this is selected, the program saves it as a variable that can be used by subsequent programs;
                - if they want to create a new file, the program proceeds to sampling;
            - if not, it goes directly to sampling;

- MitoFinder does not perform optimally when it has to carry out an analysis on more than 7,000,000 paired reads(14,000,000 unpaired) therefore a necessary step is to create a sample from the cleaned data using the appropriate function, but before this can be done it is necessary to check what sample size is needed,
    - this is done using seqkit, which calculates the basic statistics of the cleaned file,
    - statistics are passed to AWK, which extracts the length of the file from the table,
    - as the format used by seqkit has commas as thousands separator, sed must be used to replace all commas with nothing, which allows arithmetic operations to be performed on the data,
    - using AWK, the extracted value is compared with the maximum and converted into a percentage. The program tells the user what percentage came out of the analysis, then gives a recommended value (rejecting decimals as the value taken by the next program can only be an integer and rounding, especially with a large amount of data, can deviate greatly from the maximum value) and asks what value the user wants to use. Once the value has been entered, the program informs the user of the value they have entered and proceeds to the next step.
- using the downsample.py function from the MITObim package, a data sample of the size specified by the user is taken from the cleaned file, and a sample is compressed to save space using the gzip function;

■ mitfi_sing
- once the data has been properly prepared, it is now possible to start the MitoFinder itself,
- project name "-j" (for easier identification) is the file name along with the sample percentage, "-s" is the file from the previous subroutine, "-r" is the reference file(as mentioned earlier in genbank format), and "-o" the organism type given when calling the program, the output file most important for this script is "[Seq_ID]_mtDNA_contig.fasta", which contains the reconstructed mitochondrial genome in fasta format, the other output files are: information about the structure of the reconstructed sequence and what information was used to reconstruct the sequence.

2. NOVOplasty
    - paired end
        - novpla
            - for reasons of convenience and to prevent the use of a bad master file for modification, the function has an entire configuration file in it, which is created each time before the programme is switched on and is used to call it,
            - lines that are modified in the file:
                - “Project name” - project name used to name files,

        - “Forward reads” and “Reverse reads” - path to the input files,
        - “Seed Input” - path to the reference file,
        - “Output path” - the path where the output files are to be saved,
        - others remain unchanged,
      - The programme is started by calling NOVOPlastyVERSION.pl script using a Perl interpreter, with the option -c pointing to the created configuration file.
  - single end
    - novoplasty does not support the option of using single end files(the option does, however exist in the configuration file);
3. MITObim
  - paired end
    - interleave
      - because mitobim has a problem with multiple files, the paired files have to be merged into a single file, in a way that preserves this information,
      - using the reformat.sh function from the bbmap package, the data from both paired files are written into one,
    - mitobim_pair
      - an active container is created with the entire MITObim in a mode that allows commands to be passed to an already running container(the ability to communicate with the container from the script) when a command is first run, the container is downloaded from the relevant repository; synchronisations are also created to allow data to be passed to the container and later extraction,
      - the name of the container is written to the variable (information is needed to be able to pass commands to the corresponding container),
      - using docker exec command MITObim is activated in the container,
      - the output in the container is copied to the corresponding synchronised folder,
      - the container is no longer needed so it is stopped and removed,
  - single end
    - mitobim_sing
      - an active container is created with the entire MITObim in a mode that allows commands to be passed to an already running container(the ability to communicate with the container from the script) when a command is first run, the container is downloaded from the relevant repository; synchronisations are also created to allow data to be passed to the container and later extraction,
      - the name of the container is written to the variable (information is needed to be able to pass commands to the corresponding container),

- using docker exec command MITObim is activated in the container,
- the output in the container is copied to the corresponding synchronised folder,
- the container is no longer needed so it is stopped and removed

If the -A option is selected, all listed programmes are run in sequence, depending on the -p T/F flag in either single or pair end version.

3. Results

The final output for each of the programmes is a reconstructed mitochondrial genome available for use in further analyses. Dependzing on the program run, in addition to the reconstructed genome, other files containing more information about the analysis will be found in the defined locations:

- MitoFinder
  - NAMES.PERCENT_final_genes_NT.fasta - contains the final nucleotide sequences of genes found in reconstructed mitochondrial genomes
  - NAMES.PERCENT_final_genes_AA.fasta - contains the final amino acid sequence of the genes found in the reconstructed mitochondrial genomes
  - NAMES.PERCENT_mtDNA_contig.fasta - reconstructed mitochondrial genome in fasta format
  - NAMES.PERCENT_mtDNA_contig.gff - reconstructed mitochondrial genome with predicted gene placement in GFF3 format
  - NAMES.PERCENT_mtDNA_contig.tbl - reconstructed mitochondrial genome with predicted gene placement in a format suitable for publication in Genbank
  - NAMES.PERCENT_mtDNA_contig.gb - reconstructed mitochondrial genome with predicted gene placement in Genebank's visualization format
  - NAMES.PERCENT_mtDNA_contig_genes_NT.fasta - contains the nucleotide sequences of the predicted genes
  - NAMES.PERCENT_mtDNA_contig_genes_AA.fasta - contains the amino acid sequences of the predicted genes
  - NAMES.PERCENT_mtDNA_contig.png - a diagram showing the location of predicted genes on the reconstructed mitochondrial genome
  - NAMES.PERCENT_mtDNA_contig.infos - contains data on the reconstructed genome: length, name and GC content
- NOVOplasty
  - Contigs_NAMES.txt - all possible reconstructed genomes: whole or fragmented
  - Circularized_assembly_NAMES.fasta - an attempt to reconstruct a circular genome from one of the reconstructed: whole or fragmented
  - Merged_contigs_NAMES.txt - if none of the reconstructed fragments are suitable to reconstruct a circular genome, NOVOPlasty will try to merge them together to create one
- MITObim
  - There are two folders called iteration* in the folder ./NAMESY/output in these folders there are
    - folder NAMES_assembly

        - NAMES_d_results - contains the reconstructed mitochondrial genome in various formats,
        - NAMES_d_info - various technical information about how the programme worked,
        - NAMES_d_tmp - logs and temporary intermediate files of the reconstructed genome,
        - NAMES_d_chkpt - all files needed to resume the analysis in the event of programme failure or to extend it after completion if results are not satisfactory.
    - backbone_it*_initial_NAMES.fna - contains the selected reference sequence,
    - baitfile.fasta - reference sequence in a more convenient arrangement for the programme,
    - hashstat.bin - probably statistics on the hash table in the binary form,
    - manifest.conf - configuration input file,
    - NAMES-readpool-it*.fastq - sequences from the input file deemed suitable for mitochondrial genome reconstruction attempts,
    - NAMES-NAMES-it*_noIUPAC.fasta - consensus sequence without IUPAC convention;

4. Summary

The aim of the project was to develop a programme that will make the work of many researchers easier and more efficient and, based on the tests carried out, it can be concluded that the programme created works well. It is planned to develop the programme as part of a master's thesis: to introduce support for longer reads, and more programmes and to compare the quality of the reconstructed sequences depending on the programme and type of input data. The presence of this tool will facilitate the work and data acquisition of many scientists and will allow us to expand our knowledge of mitochondrial DNA using already existing NGS data.

5. References

[1] Berglund, E.C., Kiialainen, A. & Syvänen, AC. Next-generation sequencing technologies and applications for human genetic history and forensics. Investig Genet 2, 23 (2011). https://doi.org/10.1186/2041-2223-2-23

[2] Payne, A., Holmes, N., Rakyan, V.K., & Loose, M.W. BulkVis: a graphical viewer for Oxford nanopore bulk FAST5 files. Bioinformatics, 35, 2193 - 2198 (2019).

[3] https://www.protocols.io/view/mitogenome-assembly-from-ngs-genome-skimming-data-5qpvo5j3xl4o/v1

[4] Andrews, S. (2010). FastQC: A Quality Control Tool for High Throughput Sequence Data [Online]. Available online at: http://www.bioinformatics.babraham.ac.uk/projects/fastqc/

[5] Allio, R, Schomaker-Bastos, A, Romiguier, J, Prosdocimi, F, Nabholz, B, Delsuc, F. MitoFinder: Efficient automated large-scale extraction of mitogenomic data in target enrichment phylogenomics. Mol Ecol Resour. 20, 892– 905 (2020). https://doi.org/10.1111/1755-0998.13160

[6] Dierckxsens N., Mardulyn P. and Smits G. NOVOPlasty: De novo assembly of organelle genomes from whole genome data. Nucleic Acids Research, (2016). doi: 10.1093/nar/gkw955

[7] Shifu Chen, Yanqing Zhou, Yaru Chen, Jia Gu; fastp: an ultra-fast all-in-one FASTQ preprocessor. Bioinformatics, 34, 17, i884–i890 (2018). https://doi.org/10.1093/bioinformatics/bty560

[8] Shen W, Le S, Li Y, Hu F (2016) SeqKit: A Cross-Platform and Ultrafast Toolkit for FASTA/Q File Manipulation. PLoS ONE 11(10): e0163962. https://doi.org/10.1371/journal.pone.0163962
[9] Hahn C., Bachmann L., Chevreux B. Reconstructing mitochondrial genomes directly from genomic next-generation sequencing reads—a baiting and iterative mapping approach. Nucleic Acids Research, 41, 13, e129 (2013). https://doi.org/10.1093/nar/gkt371
[10] https://jgi.doe.gov/data-and-tools/software-tools/bbtools/bb-tools-user-guide/bbmap-guide/