mitopipeline Documentation

Release 0.1

Timothy Kuo

Aug 24, 2019

Contents

CHAPTER 1

Getting Started

This project can be downloaded through `pip install mitopipeline` to install the latest version of mitopipeline. Or, you can directly clone the repository through `git clone https://github.com/timmykuo/mitopipeline.git`, navigate to the cloned repository, and run `python3 setup.py install`. Documentation is hosted through readthedocs.

Users will have to download samtools and bwa, `make` them, and include the path to the executable in their $PATH variable or copy them to their `bin` folder so that they can be run from the command line. Download links for samtools and bwa. In addition, 3rd party software packages that they wish to use on their own (such as GATK, ANNOVAR, etc.) need to be downloaded before being able to use them in the pipeline.

CHAPTER 2

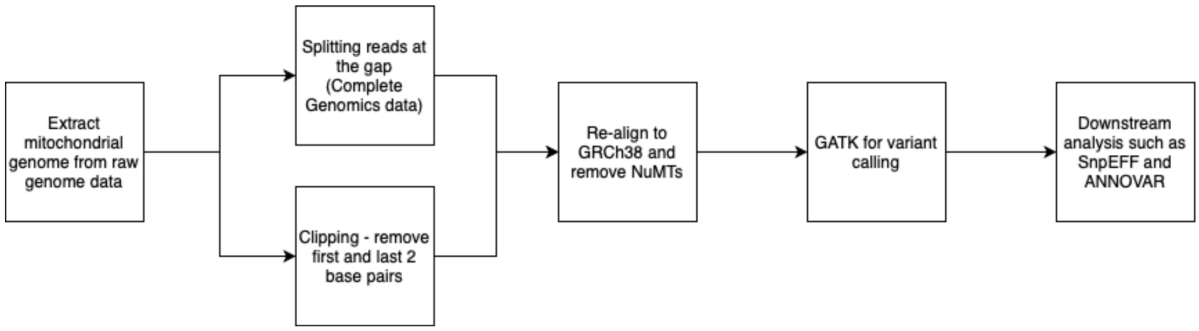
Purpose

In genetics research, researchers often require raw genome data to be run on different softwares first to extract useful information such as variant information before it can be analyzed. Thus, we want to provide a pipeline that allows users to streamline their processing automatically for them.

The steps included within this pipeline include extracting mitochondrial genome from human genome, clipping, splitting the gap for Complete Genomics data, alignment into NuMT removals, and other software packages such as GATK, SNPEFF, ANNOVAR.

CHAPTER 3

Pipeline



The steps shown in this figure are described further in the Pipeline Steps page. Each step can be omitted if it's unncessary for your pipeline.

CHAPTER 4

Credits

The dependency management is controlled through a python package called Luigi. In addition, many of the steps in the pipeline were adapted from scripts written by Sneha Grandhi. Check out her Github, LinkedIn, or contact her through her email: sneha_grandhi@hms.harvard.edu. In addition, Janet Wang helped with getting mitopipeline set up on the Case HPC Server. Check out her LinkedIn!

CHAPTER 5

Table of Contents

5.1 Example - One TARGET Genome File

This section will go through an example of how to run the pipeline step by step assuming that you've gone ahead and installed it through pip or cloned and setup through git.

5.1.1 Step1 - Establish subdirectories such as starting directory, out directory, REFs, and Tools

In the example case, the file we will be using is located within <path/to/mitopipeline/example>, so depending on where your mitopipeline package is installed, you will have to specify that directory.

In addition, the REFs directory will need to be referenced based on where you have the human reference genomes. The specific steps for when the reference genomes are needed can be found in the Pipeline Steps page.

It's also necessary to specify where the tools directory that contains the softwares needed for each individual step. In my case, the software package's executables are located in the same folder as the start and out folders. Thus, the directories are as follows:

Thus, our following directories to be specified can be relative to the current directory you are in or they can be the full path. In my case, the directories are:

- start directory == './example'
- out directory == '../test/out'
- tools directory == './tools'
- genomes directory == '/Users/Timmy/Documents/CWRU/Thesis/mitopipeline/mitopipeline/REFs'

Since the example genome file is from Complete Genomics, we won't be clipping the bam files and thus we won't need bam2fastq and seqtk.

- './tools/gatk/GenomeAnalysisTK.jar'
- './tools/snpeff/snpEff.jar'

- './tools/annovar/table_annovar.pl'
- './tools/annovar/convert2annovar.pl'
- '<path/to/REFs>/hg38-nochr.fa'
- '<path/to/REFs>/hg38.fa'
- '<path/to/REFs>/rCRS-MT.fa'

Details about each step's requirements can be found in the Pipeline Steps page.

5.1.2 Step2 - Run pipeline

Now that we have our directories, we can run the pipeline.

```
$ mitopipeline -s ./example -o ../test/out -g /Users/Timmy/Documents/CWRU/Thesis/
↪mitopipeline/mitopipeline/REFs/ -t ./tools -r clipping extractmito
```

One thing to note is that since this example file is from the TARGET database and was sequenced by Complete Genomics, we will be using the split gap step but without clipping since our reads are relatively short already. Also, since our starting file is already the mitochondrial genome, we can also skip the extract mito step.

It's also worth noting that we do not specify the number of workers. Since we only have one file in our example folder, it's not necessary to have multiple workers and so we can use the default of 1.

5.1.3 Step3 - View Output

Once the pipeline has finished running, you will see either a successful or error. You can read the log output to see what parts may have failed and which have succeeded. All output files can be viewed within the specified out directory with subdirectories created for each step. Here is an example of what a successful output would look like:

```
===== Luigi Execution Summary =====

Scheduled 13 tasks of which:
* 2 complete ones were encountered:
    - 2 ExtractMito(id=AML_TARGET-20-PANPKN_14_mito,NBL_TARGET-30-PAPTDH_10_mito)
* 11 ran successfully:
    - 2 ANNOVAR(id=AML_TARGET-20-PANPKN_14_mito,NBL_TARGET-30-PAPTDH_10_mito)
    - 2 GATK(id=AML_TARGET-20-PANPKN_14_mito,NBL_TARGET-30-PAPTDH_10_mito)
    - 1 PipelineRunner()
    - 2 RemoveNuMTs(id=AML_TARGET-20-PANPKN_14_mito,NBL_TARGET-30-PAPTDH_10_mito)
    - 2 SNPEFF(id=AML_TARGET-20-PANPKN_14_mito,NBL_TARGET-30-PAPTDH_10_mito)
    ...

This progress looks :) because there were no failed tasks or missing dependencies

===== Luigi Execution Summary =====
```

5.2 Pipeline Steps

A number of steps are able to be included within the pipeline, and you can control which steps to include through the `-r` option on the command line. All of them utilize samtools, bwa or both and thus are necessary to download. In addition, if you decide to use the `-l` (slurm) option, then each file that is run on each step of the pipeline will be submitted as a slurm job, provided your server uses lmod.

This section will explain each step in further detail so you can determine if it's necessary to include within your pipeline.

5.2.1 Extract Mito

REQUIRED
Tools from command line: samtools
Input: bam file
Output: bam file

```
samtools index $START/$FILENAME.bam
samtools view -H $START/$FILENAME.bam > $START/$FILENAME.header.bam
grep -i SN:MT $START/$FILENAME.header.bam > $START/$FILENAME.MT.bam
grep -i SN:chrM_rCRS $START/$FILENAME.header.bam > $START/$FILENAME.chrM_rCRS.bam
grep -i SN:chrM $START/$FILENAME.header.bam > $START/$FILENAME.chrM.bam
grep -i SN:M $START/$FILENAME.header.bam > $START/$FILENAME.M.bam
```

This snippet extracts the mitochondrial genome from the bam file that was passed in where `$START` is the directory that contains the bam files and `$FILENAME` is the filename. Since the mitochondrial genome can have different notations based on which reference genome it was aligned to, the script takes into account all the different mitochondrial genome tags. Output is a bam file.

5.2.2 SplitGap

REQUIRED
Tools from command line: samtools
Input: bam file
Output: fastq file(s), "_1/_2" naming convention if paired end

This step is particularly for Complete Genomics data. Complete Genomics uses a technique called DNA Nanoball Sequencing.

The downside to this technique is that each read ends up having a 'gap' in the middle of the read that doesn't match to the reference genome. For example, a reference genome read of 'ATCGA' on a DNA Nanoball Sequencing read could be 'ATA' where the cigar string would be 2M2N1M. To fix this, we use the sam file's format cigar string to split the read where the matches are into two reads. So the previous examples reads would be split into 'AT' and 'A' with 2M and 1M being their respective cigar strings.

```
#regex matches any amount of numbers followed by one capital letter
matches = re.findall(r'[0-9]+[A-Z]{1}', cigar)
curr_num = 0
temp_idx = 0
let = ""
```

(continues on next page)

(continued from previous page)

```
for match in matches:
    num = int(match[:-1])
    let = match[-1]
    #append next reads and quals if not an M or an I
    if let != 'M' or let != 'I' and curr_num != 0:
        add_next_seq(temp_idx, curr_num, new_reads, reads, new_quals, quals)
        temp_idx += curr_num
        curr_num = 0
    else:
        curr_num += num
```

The above code block finds all cigar blocks, i.e. any number followed by a capital letter like 1M, 5N, 40I, etc. It then appends on the next reads only if the letter is not an M or an I, which are the reads that match up to the reference genome.

Although this does decrease the read size since they are being split, the quality of the reads drastically improve. Output is a bam file.

5.2.3 Clipping

REQUIRED
Tools from command line: samtools, bwa
Tools from tools directory: bam2fastq, seqtk
Input: bam file
Output: fastq file(s), "_1/_2" naming convention if paired end

The clipping step takes in as input a bam file, either from the mitochondrial bam file extracted from the previous step, or as a starting point. First, it changes the bam file into a fastq format:

```
$TOOLS/bam2fastq/bam2fastq -f -o $OUT/$1_bam2fastq_#.fastq $START/$1$filetype.bam
```

Then, depending on if it's a paired end or single end, will trim the first and last two pairs from every read.

```
if [ -e "$OUT/$1_bam2fastq_1.fastq" ]
then
echo "PAIRED-END"
        echo "--CLIPPED: Removing first and last 2 base pairs from every read"
        $TOOLS/seqtk/seqtk trimfq -b 2 -e 2 $OUT/$1_bam2fastq_1.fastq > $OUT/$1_1.
↪fastq
        $TOOLS/seqtk/seqtk trimfq -b 2 -e 2 $OUT/$1_bam2fastq_2.fastq > $OUT/$1_2.
↪fastq
else
echo "SINGLE-END"
        echo "--CLIPPED: Removing first and last 2 base pairs from every read"
        $TOOLS/seqtk/seqtk trimfq -b 2 -e 2 $OUT/$1_bam2fastq.fastq > $OUT/$1.fastq
fi
```

The output of this step will be $filename.fastq or $filename_1.fastq and $filename_2.fastq, depending on if it's a paired end read or not.

5.2.4 Remove NuMTs

REQUIRED
Tools from command line: samtools, bwa
Reference genomes from genome directory: hg38 mitochondrial reference genome (rCRS-MT.fa), hg38 human genome without mitochondrial genome (hg38-norcrs.fa), and hg38 human genome (hg38.fa)
Input: fastq file(s) "_1/_2" naming convention if paired end
Output: bam file

NuMTs are DNA sequences harbored in the nuclear genome, but closely resemble sequences in the mitochondrial genome. We remove these as quality control and to reduce noise in the following steps. The output of this step is a bam file with NuMTs removed

To do this, we first align our input fastq files to both the mitochondrial genome and hg38 without the mitochondrial genome to find any close matches. Then, we extract the perfect matches to the nuclear genome, realign the resulting fastq file back to hg38 reference genome, and extract the mitochondrial genome. Output is bam file.

5.2.5 GATK

REQUIRED
Tools from tools directory: GenomeAnalysisTK.jar
Reference genomes from genome directory: hg38 mitochondrial reference genome (rCRS-MT.fa)
Input: bam file
Output: vcf file

The gatk script were adapted from the suggested pipeline by GATK. In particular, the following steps are run in order:

Picard's AddOrReplaceReadGroups, Picard's MarkDuplicates, GATK's RealignerTargetCreator, GATK's IndelRealigner, GATK's FixMateInformation, GATK's BaseRecalibrator, GATK's PrintReads, GATK's HaplotypeCaller, GATK's VariantFiltration.

An example of how gatk is called:

```
java -Xmx10g -jar $TOOLS/gatk/GenomeAnalysisTK.jar \
-T HaplotypeCaller \
-R $REFS/rCRS-MT.fa \
-I $TMPDIR/$1.tcga.marked.realigned.fixed.read.bam \
--maxReadsInRegionPerSample 200 \
--sample_ploidy 100 \
-stand_call_conf 50 \
-stand_emit_conf 10 \
--pcr_indel_model HOSTILE \
-minPruning 10 \
-A StrandAlleleCountsBySample \
--dbsnp $5/dbsnp/mtONLY.vcf \
-o $TMPDIR/$1.tcga.snps.vcf
```

Something important to note is that the gatk.jar executable must be placed within a folder called gatk within the tool's directory. Output is vcf file.

5.2.6 SNPEFF

REQUIRED
Tools from tools directory: snpEff.jar
Input: vcf file
Output: vcf file

This use's snpeff's most basic command and using the most recent mitochondrial reference genome GRCh38.86

```
java -Xmx4g -jar $TOOLS/snpEff/snpEff.jar GRCh38.86 $VCFS/$1$filetype.vcf > $SNPEFF/
↪$1_snpEff.vcf
```

This is the standard usage of snpEff. You can read more about it on their website. Also note that the snpEff executable must be placed within a snpEff folder within the tool's directory just like gatk.

5.2.7 ANNOVAR

REQUIRED
Tools from tools directory: Annovar's convert2annovar.pl, Annovar's table_annovar.pl
Input: vcf file
Output: vcf file

Annovar can only be downloaded after registering on their website.

```
#convert vcf file to avinput file
perl $TOOLS/convert2annovar.pl -format vcf4 $VCFS/$1$filetype.vcf  > $ANNOVAR/$1.
↪avinput

perl $TOOLS/table_annovar.pl $ANNOVAR/$1.avinput $TOOLS/humandb/ -remove -protocol␣
↪dbnsfp33a -operation f -build hg38 -nastring . > $3/$1.avoutput
```

One thing to note is that you have to first download the -buildver and databases for hg38 through a different script called annotate_variation.pl. You can read more about these files on their guide.

5.3 Running the Pipeline

This section explores details about running the pipeline such as the tools it uses, its requirements, and some suggestions on how to run it. Although some of these options are described in Command Line options, this will delve into more details.

5.3.1 Required Arguments

Start Directory:
The start directory is necessary in order to specify where all the files that need to be run are. One thing to note is that the pipeline use's the '.' from the file's extension to parse the file name. Thus, all files within the start directory can only have one period within its name that specifies the extension, i.e. FILENAME.bam. In addition, .bai index files are allowed to be in the same directory as the .bam files.

Tools Directory:
In order to run 3rd party softwares like seqtk, gatk, snpeff, and annovar, mitopipeline requires you to specify where these tools are kept. Each software package should have a folder within the tools directory that contains its executables. For example, gatk's executable should be path/to/tools/gatk/GenomeAnalysisTK.jar, snpeff's should be path/to/tools/snpeff/snpEff.jar and annovar's should be path/to/tools/annovar/annovar-executables. One choice is to first use mitopipeline's download function if you don't want to download these yourself.

Reference Genomes:
With steps such as GATK and RemoveNuMTs, it's necessary to have human reference genomes to align to. The required genomes for this pipeline are hg38-nochr.fa (the GRCh38/hg38 version human genome without the mitochondrial genome), hg38.fa (the GRCh38/hg38 version of the human mitochondrial genome), and rCRS-MT.fa (the GRCh38/hg38 version of the human mitochondrial genome). The file names must be changed to match the ones listed here so that the steps are able to find the files. You can read more about the human references genomes from the UCSC genome browser. Since the files are too large to be downloaded along with the mitopipeline package, the user must specify the path to the reference genomes after downloading.

5.3.2 Python Modules on Case's HPC

On an online server, users typically do not have permission to download softwares and modules directly due to lack of administrative privileges. In this case, users will need to set up python modules that contains the location of mitopipeline. Here's an example of how to do this on Case Western's HPC server that uses the slurm workload mananger and lmod. For more details, you can refer to the Case Western HPC website for more detailed instructions on how to use pip to install and use python modules:

First, to install the module, we need to first set up the enviromental module $PYTHONUSERBASE.

```
export PYTHONUSERBASE=$HOME/.usr/local/python/3.5.1
pip install --user mitopipeline
```

This will install mitopipline into the $PYTHONUSERBASE directory. To use the installed module (since this tool contains binaries), we need to include it in the path. For our local directory of packages, we will create a module file that will set those variables for us. We will call the module "python-modules" and we will set the version of the module to the version of Python we are using. First, we will create the directory (just once).

```
PYTHONMODULES=$HOME/.usr/local/share/modulefiles/python-modules
mkdir -p $PYTHONMODULES
cd $PYTHONMODULES
```

Then, we will create a file called 3.5.1-gcc.lua in this directory that contains the following contet:

```
-- This is Lua module file for our local Python
-- modules.
-- To use it just run
--    module load python-modules/3.5.1-gcc
--
--

load("intel/17", "openmpi/2.0.1","python/3.5.1")

pushenv("PYTHONUSERBASE",pathJoin(os.getenv("HOME"),".usr/local/python/3.5.1"))
prepend_path("PATH",pathJoin(os.getenv("HOME"),".usr/local/python/3.5.1/bin))
```

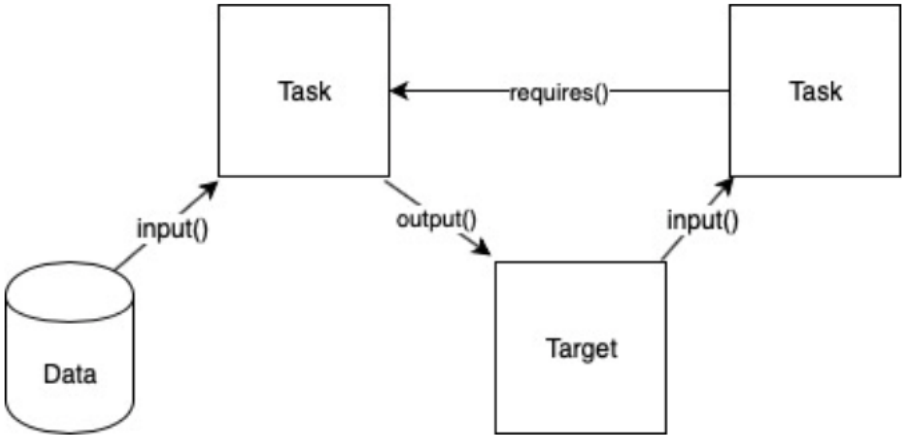
To use this module, just run

From here, if you used mitopipeline's "-d" command line option to download tools for you, the tools directory can now be referenced through: " /home/<case_ID>/.usr/local/python/3.5.1/lib/python3.5/site-packages/mitopipeline/tools/". Note: If you are using ANNOVAR, you must move the entire folder over to "~/.local/lib/python3.5/site-packages/mitopipeline" after downloading yourself, because ANNOVAR requires user registration before downloading (so it's unavailable through "-d"). Of course, you can also specify your own tools directory.

5.3.3 Luigi

As mentioned before, the dependency management of the pipeline is handled through a python package called luigi. However, currently the only options available when running luigi are adjusting the number of workers. You can read more about workers here

Luigi handles dependency management through class objects called Targets and Tasks. Targets are a Task's output. A Task is run after a required Task is complete and also outputs a Target for a next Task to be run. For example, the workflow for two tasks running on a database can be shown like this:



In this diagram, the first task takes in the data from the database as input and outputs a target. The target is then input to the next task to be run. In order for the second task to be run, it "requires" the first task to be finished first. This is tracked through the existence of the first task's output (the target). Once it sees the target in the output, the 2nd task will begin running. The advantage of such a design is its asynchronous processes. Since the time for each individual file may be different for the same task, having a worker that looks solely for the output target allows for the multiple tasks to be run at the same time.

5.3.4 Softwares

As described in the pipeine steps section, all of the steps have some software requirements in order to be run. There are two options for getting the softwares necessary.

The first choice is to use the command line option `-d`. For example, the command

```
$ mitopipeline -d -r annovar snpeff
```

will download all the necessary software into mitopipeline's tool's directory for all steps except for annovar and snpeff. You can then use the mitopipeline normally without specifying the tools directory.

The second choice is to specify a directory that has all the necessary softwares downloaded. This is only necessary only for the step softwares, including seqtk, GATK, SNPEFF, and ANNOVAR. Keep in mind that mitopipeline will check for the naming convention of the software's folder that contains its executable as the same name as the step i.e. 'gatk' step will look for a folder called 'gatk' within the specified directory for its executable.

A number of softwares are necessary to be run on the command line as they are called directly through the bash scripts. In particular, 'samtools' and 'bwa' need to be able to be executed through the comand line. On MacOSX/Linux, this can be achieved by either copying the executable to your `/usr/local/bin` folder or adding the folder of your executable to your $PATH variable. You can read more about each step's required softwares on the Pipeline Steps page.

5.3.5 Using Slurm Jobs

Some servers have the slurm workload manager set up on their system. If you are using such a server, an available option is to use the option `-l`. This will submit slurm jobs for each step of the pipeline for each file and save the files in a folder within the specified -out directory.

5.3.6 Tmux

Currently, luigi's scheduler is not implemented within this tool and only uses its local scheduler (read in luigi's docs). Thus, it requires that whatever process that is running mitopipeline to be continually running. One way to do this is to run it on a server using a tmux session. You can read more about tmux here.

Once tmux is downloaded, you can start a new tmux session by typing `tmux` into your command line. Then, after beginning the pipeline through the `mitopipeline` command, you can exit the session by pressing `ctrl+b` and then `d`. This will detach the current tmux session from your terminal.

In order to reenter your tmux session, you can type in `tmux ls` in order to list all of your sessions and then `tmux a -t <your-session-id>` to re-enter that tmux session where your mitopipeline is running.

5.4 Command Line Options

This section explores the different options that are possible when running the tool. Before running the tool, you can first run the command `mitopipeline -d` in order to download all of the dependencies necessary for the steps in your pipeline if you do not specify a tools directory.

5.4.1 Downloading Software

Option	Description
`-d` or `--download`	**REQUIRED**. Option to download softwares for steps not listed in -r.
`-r` or `--remove`	Specifies steps that won't be run in the pipeline. **Input**: <name of steps> **Default**: None

5.4.2 Running Pipeline

5.5 Troubleshooting

5.5.1 Outdated Modules on Case Western's HPC

As mentioned in the Running The Pipeline section, using mitopipeline on a separate server without admin privileges may require you to update various python modules on your own local ennvironment. For Case Western's HPC server, the details can be found here. In particular, python-dateutil will most likely need to be updated and installing mitopipeline itself will require you to use this method.

5.5.2 Reference Genomes

Since the reference genomes are used by their exact file names, it's important to name all of your reference genomes the same way. Specifically, the reference genomes must be named hg38-nochr.fa, hg38.fa, and rCRS-MT.fa.

In addition, you must index the reference genomes through samtools/bwa. For each reference genome, the following complimentary file must exist (obtained through indexing):

<REF_name>.fa, <REF_name>.fa.amb, <REF_name>.fa.ann, <REF_name>.fa.bwt, <REF_name>.fa.fai, <REF_name>.fa.pac, <REF_name>.fa.sa, <REF_name>.dict

Lastly, the mitochondrial genome MUST be named "MT" (not chrM, chrM_rCRS, M) in all of the reference genomes. This is because in hg38, the mito genome is named MT, and matching to the reference contig is done through the literal string of the name.

5.5.3 SnpEff

SnpEff requires its own database and reference genome in order to be run. Thus, if it's your first time running the pipeline and you have multiple workers, it may be necessary to run 1 file first. Since SnpEff will be downloading its database the first time it runs, having multiple workers overlapping and trying to download the same database can cause some issues within the pipeline.

5.5.4 ANNOVAR

Similar to SnpEff, ANNOVAR also requires its individual database and files in order to run. After registering and downloading, it's necessary to run table_anovar.pl before runnig mitopipeline. In particular, the quick start page for ANNOVAR explains in detail how to do this. Note: the scripts in mitopipeline uses hg38, NOT hg19 as suggested on the website. Other than that, you can run each command as listed there.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search