Analysis of Mitochondrial DNA Base Substitution Diseases: rRNA/tRNA mutations

Stephanie Gardner

University of California, Santa Cruz: BME160

Abstract

The mitochondrial genome is of public knowledge, and mutations that are found in this genome are also available to the public online. This data outlines where the mutation occurs in the genome, what phenotype is expressed by the mutation, and the pathogenicity of the disease caused by that mutation. This information is set up as a table on the internet, in which anyone can look up in order to use in scientific study. However, the table is difficult to use for comparative analysis and lacks a graphical representation of the mutations, which would allow for deeper investigation of the mutations. This paper outlines a new method in which genome and mutation information can be combined with the python computer language in order to create a more informative infographic and mutation calculations.

1 Introduction

The mitochondrial genome is circular and codes for 13 proteins, 22 tRNA's and 2 rRNA's. Mitochondria are responsible for producing protein subunits of enzyme complexes that participate in the oxidative phosphorylation system which supply the cell with ATP. Each cell consists of multiple mitochondria that contribute to this production of energy, making mitochondria the powerhouse of the cell. This leads mutations found in mtDNA (mitochondrial DNA) to develop a more pronounced mutated phenotype in tissues that require a high energy output, as this is where mitochondria are more concentrated. Tissues that require a higher energy output are ares line the brain, retinal and cardiac muscular tissue. [1]

mtDNA has a higher mutation rate than nuclear DNA. This is due to segregation during mitosis in mitochondria; the daughter cells of mitochondria differ from the parent cells, unlike nuclear chromosome segregation where the daughter cells are exact genetic copies of the parent cell.

Information about mtDNA and its mutations can be found on MITOMAP.org, along with a plethora of other information all centered around the mitochondrial genome. MITOMAP exhibits its data in various tables and charts, which can be daunting if one wanted to analyze patterns in variation of the genome. A more simple and informative way to analyze mtDNA is to implement this data into a program that can output a graphical representation of the data, along with various calculations in order to

better organize the information derived from the website. I have done this by entering the rRNA/tRNA mutation information from the website into an excel spreadsheet. It was then organized into various data frames that subsequently underwent calculations for various patterns. This includes average mutation rate and pathogenicity in a certain locus, and mutations within a specific pathogenicity interval, all the while printing a corresponding figure of the mtDNA map and/or a specific locus. [3]

2 Methods

2.1 Program Overview

This program uses the readFastA class from sequenceAnalysis.py in order to get the total base length of the mtDNA. Then, pandas is used to parse through two spreadsheets, one containing mutation information and one containing information about the various loci. The excel spreadsheets are then organized into data frames, to be later used by matplotlib functions to produce a graphical output. The data frames are also used in several calculations to generate a written output for a more precise analysis of the components of mtDNA. This program takes between 3.5 and 7.5 minutes to run, depending on how many figures are produced.

2.2 Program Architecture

2.2.1 Overview

The program consists of two classes; one for the figure saving and mutation calculations, and one for the command line arguments.

2.2.2 sequenceAnalysis.py / FastAReader

Fasta reader assembles the mtDNA genome fasta file into a separate head and sequence attribute. The length of the sequence attribute is equivalent to the length of the genome, therefore it is equal to the circumference of the mtDNA circle on my matplotlib plot.

2.2.3 GraphicRep

This class contains the FastAReader function, along with the pandas functions that collect the excel spreadsheet information to be used later in the class. It takes the information and organizes it into dictionaries, in which each key is the locus and the value is a list of various information. Subsequently, I assigned the values of each dictionary to their own list for easier and more concise accessibility in the program. There is no __init__ method in this program. Picture of the dictionary code and how to access specific elements of the code.

2.2.4 GraphicRep.optionA()

Option A method is for the user to receive a file with overall mutation and pathogenicity information. The file consists of the percent of total mutations found at a specified locus and the average pathogenicity at that same locus. In order to achieve this, a try catch is implemented in order to ignore the loci that have no mutation information, thus the program would only save the appropriate line of information for the user. This method also has to ignore all mutations that have an ‘N/A’ pathogenicity level, so a conditional of seeing whether or not the pathogenicity level is ‘nan’ (not a number) is necessary. It allows the phrase ‘nan’ to show up in the final print out and ignored this phrase in the calculation of the average pathogenicity.

2.2.5 GraphicRep.optionB()

Option B prints out all mutations within the specified pathogenicity interval. The user inputs the low and high values in the command line which is then passed to this function. This method tests all pathogenicity levels from the mutation dictionary to see if they are within the low and high values. It then saves this to a file

for the user to analyze. The default command line option for this method lists the pathogenicity, which tRNA/rRNA the mutation is expressed in, and where the base mutation occurs in the mtDNA. The extra option is for the print out to also list the linked disease with each mutation, but I found this print out to be too crowded so I kept the default to print the more concise information.

2.2.6 GraphicRep.get_cmap()

This function was created by stackoverflow.com user 'Ali'. this method "returns a function that maps each index in 0, 1, ..., n-1 to a distinct RGB color; the keyword argument name must be a standard mpl colormap name." It is called in a for loop in later methods by: 'color=cmap(i)'. [2]

2.2.7 GraphicRep.createMap()

This method creates the overall mtDNA map. It starts out with basic assignment of the parameters of the mtDNA map, like the circumference, radius, and location of each base according to the other parameters. Then, the figure is declared, it is assigned to polar axis, and the grid is removed for a cleaner looking picture, and the origin is labeled at the top of the plot. Each mutation is iterated through and assigned a color based on which pathogenicity interval it is in. It is then plotted at its corresponding position on the plot, and rotated according to its own specific theta. The marker I chose was a vertical bar line ('|'), I did not want the base markers to be too bulky, and thin markers allow for a more accurate figure. Each marker is at radius 0; I chose this value to keep the marker placement simple and consistent. These markers were also placed above the locus markers in order for the user to see where they are placed in the loop ('zorder' parameter).

Example:

```
ax.plot(theta,0,
marker=
(2,0,(theta*(180/np.pi))),
color=colors,
markersize=20,
zorder = 6)
```

The next part of this method is placing markers for each base, each one is color coded based on their locus. This color was chosen by the color=cmap(i) assignment in the making of each marker. These markers were also thin bar markers in order to maintain consistency in the map. The locus markers spanned the entire mtDNA circle and were also rotated according to their location on the map. Each locus also consisted of another marker plot, the start of the locus was marked with an extra long bar marker to indicate the boundaries of each locus.

Overall, this figure takes 3.5 minutes to render, which I consider to be too long. Later versions of this program I hope to decrease the run time.

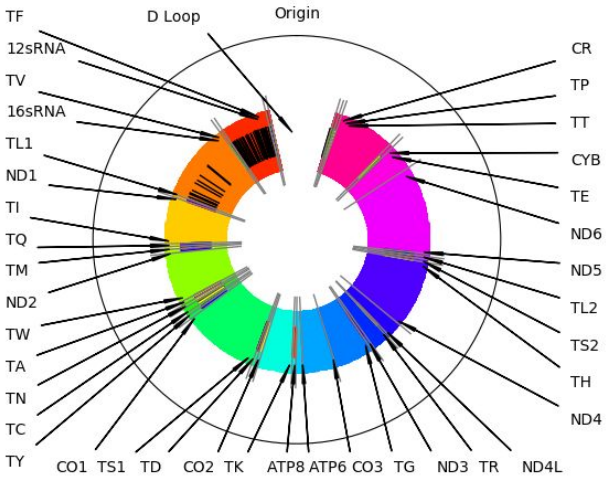

Figure 1: Graphical output from createMap() method.

2.2.8 GraphicRep.zoomPlot()

The first part of this function is the create Map function in its entirety. The difference between these two plots is their axis minimum and maximum values. The createMap has a minimum of 0 theta (0 degrees), and the maximum is 2π theta (360 degrees). In this plot, a locus is specified in the command line argument. After which the method finds the start and stop of that locus and sets the minimum and maximum theta accordingly. This figure is then saved and labeled in reference to its locus.

This figure also takes 3.5 minutes to render, this is due to the entire map being created and then then the theta minimum and maximum being set.

2.2.9 CommandLine

My command line class allows the users to specify distinct parameters to be implemented in the program. This includes: locus, pathogenicity interval, and the option to print the disease linked to each mutation in the optionB method. The command line arguments are also controlled in each method individually. For instance, in optionA and zoomPlot methods, if a locus is 'None', the entire method is ignored. And in optionB, if low or high argument is 'None', this method is ignored.

3 Discussion

While writing this program, I came up with more and more ideas to implement into it. With the data I have from MITOMAP, there are hundreds of options and routes I can take to better interpret the data. For instance, including a method where heteroplasmy or homoplasmy mutations are categorized. Or a method that only graphically plots mutations that in a specific pathogenicity interval, or that result in a specified disease. In addition to the ideas for the methods I came up with, I also thought about updates and improvements that could be added to this program. I would also like to implement an automated labeling system into the createMap method, in order for the program to handle all circular genomes, without crowding of the labeling.

This program is incredibly slow; too slow for my liking. I hope to soon improve this aspect of the program. I would also like to improve the zoomPlot method, as it is still too small to do any real analyzing of a certain locus without the written print out. A way to make a locus stretch out to about 90 degrees, while maintaining the base location and accuracy would be a more ideal and helpful method.

4 Conclusion

The figure this program prints out is more accurate than a simple figure found online of mtDNA. My figure is accurate down to the individual base placement. Being able to graphically represent a truly accurate circular genome is useful for better understanding and analyzing mutations that occur within the genome. I hope to soon update this program to be useful for analysis of all circular genomes and their corresponding mutations. Overall this program is a good start to creating a useful and accurate genome analyzing tool, but still needs to be improved to become an advantageous tool for biologists and other researchers.

4 References

[1] Chial, H., Ph.D., & Craig, J., Ph.D. (n.d.). MtDNA and Mitochondrial Diseases. Retrieved from https://www.nature.com/scitable/topicpage/mtdna-and-mitochondrial-diseases-903

[2] stackoverflow.com/users/341970/ali (n.d.). How to generate random colors in matplotlib? Retrieved from https://stackoverflow.com/questions/14720331/how-to-generate-random-colors-in-matplotlib

[3] MITOMAP: Reported Mitochondrial DNA Base Substitution Diseases: RRNA/tRNA mutations. (n.d.). Retrieved from https://www.mitomap.org/foswiki/bin/view/MITOMAP/MutationsRN