Title of Project: A One-Stop Web-tool for Genomic Annotation and Analysis

By:

Poh Jun Kai, Nigel

Dunman High School

Dr Benedict Yan

National University Hospital

A One-Stop Web-tool for Genomic Annotation and Analysis

Poh Jun Kai, Nigel

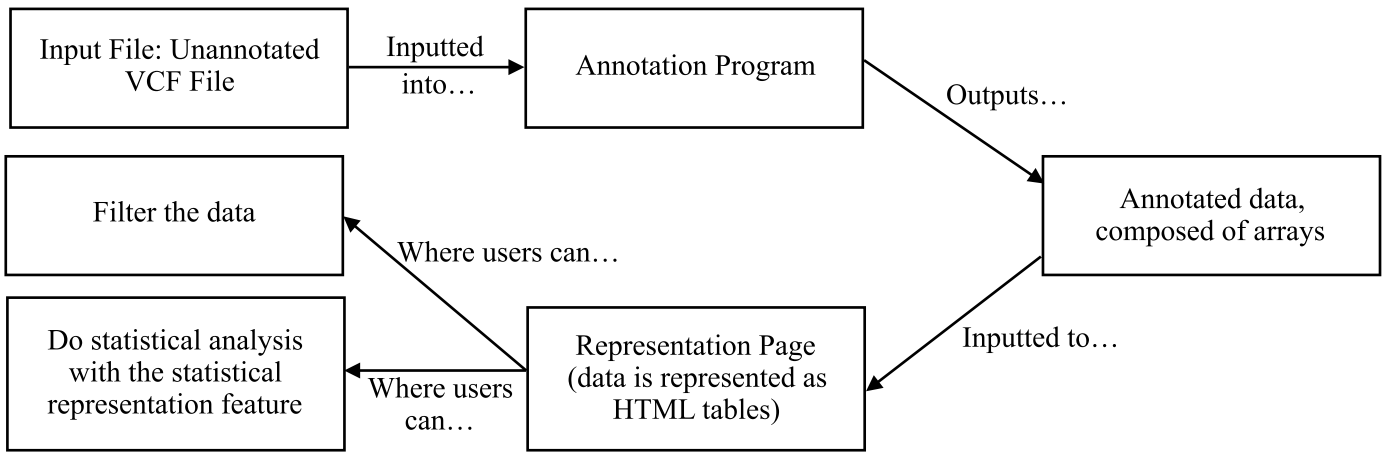
INTRODUCTION

Genomic information comes, usually, in the form of Variant Call Format (VCF) files that are generated by genomic sequencing machines. However, the generated VCF files contain crucial information that allows researcher to better understand the nature of certain diseases, such as cancer. Such information is usually presented in a way that is not friendly to human readers, let alone human analysis. Furthermore, these information often come in large databases, making it tedious for any meaningful processing. As such, there arises a need to annotate these genomic information to make it more human readable and create a tool to filter and process this information to output data that best suits the scientist’s area of research. Hence, the objective of this project is to create a one-stop web-tool to annotate, filter, process and perform statistical analysis (through functions such as counting) these data.

METHODOLOGY

Functions and Mechanisms

The software has 3 features – Annotation, Filtering and Statistical Representation. The overall workflow of this software is as follows: The software will take in the information, annotate it appropriately using the Annotation feature. The user can then filter out unwanted data using the Filtering feature and do frequency count analysis using the Statistical Representation feature. Once done, the users can then download these information for offline/other uses. The follow chart below demonstrates the overall flow of the software:


Input File: Unannotated VCF File
Inputted into…
Annotation Program
Outputs…
Annotated data, composed of arrays
Inputted to…
Representation Page (data is represented as HTML tables)
Where users can…
Filter the data
Where users can…
Do statistical analysis with the statistical representation feature

Flow chart 1: Overall flow of software

Software

This software was created using Hypertext Markup Language (HTML), Cascading Style Sheets (CSS) and Hypertext Preprocessor (PHP). These were used in conjunction with a Secure Shell Host (SSH) Server in creation of this locally hosted website. HTML was used to create the basic framework of the frontend webpage, CSS to add in some graphics designs in order to allow the entire webpage to become user friendly, and PHP was used to allow the frontend website communicate with the backend SSH Server which hosts all the data uploaded by users and required for the website to run.

1. Annotation

This feature forms the main crux of this project. It takes in large amounts of genomic data in the form of VCF files and output it in human readable HTML tables and tab separated values (tsv) files.

Uploading the file [UploadFile.php]

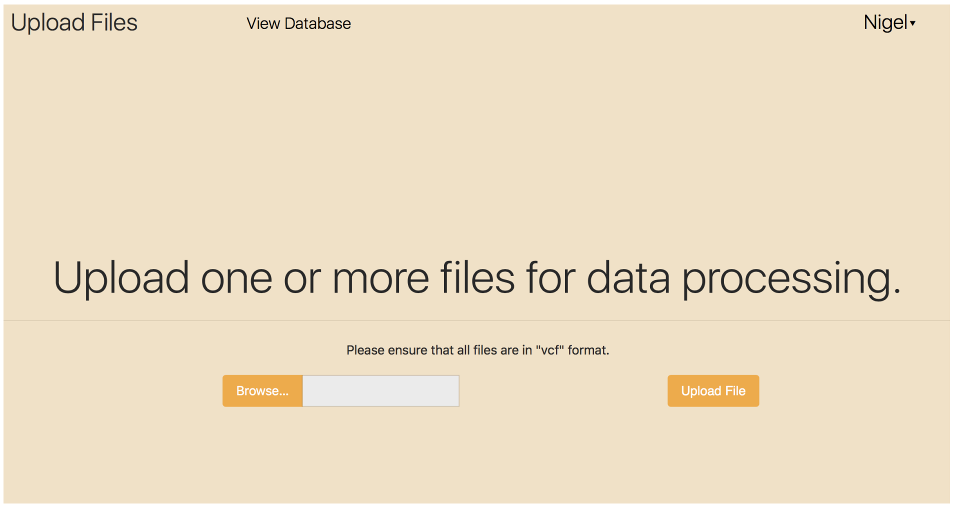

Upload Files
View Database
Nigel
Upload one or more files for data processing.
Please ensure that all files are in "vcf" format.
Browse...
Upload File

Figure 1: File Uploading Interface (Actual Website Interface: "UploadFile.php")

The user will be asked to upload one or more files for data processing, as seen in Figure 1. The user will have to click on the "Browse…" button, select their desired VCF files and click "Upload File" to submit the files for annotation. VCF files must adhere the following technical requirements: 1) VCF files must be in format 4.0 or newer, 2) The file settings and headers have to adhere to the standards specified in the International Genome Sample Resource webpage [1]. Once uploaded, a 3 step process of annotation occurs:

Important Note: Steps 1.1 and 1.2 are automated processes that the user will not be able to have access to or view. Step 1.3 will be the only step with a graphic interface, in the form of the webpage "ViewDatabase.php".

1.1. Addition

The addition mechanism refers to the collection of information from various sources on the internet/servers and the addition of all these information to a simple list-like array for processing in the later process of extraction. Mainly, we extract information from 2 different locations: Annovar and SnpEff. As this program requires us to not only use the information already in the VCF file, but also in other genomic databases such as the COSMIC Database and the ClinVar Database so as to meet the project's mission of providing all the necessary meaningful information for scientists' research, we hence have to tap on other pre-existing software such as the 2 mentioned above to gain a more complete set of information. A typical return array after the addition stage would usually contain information from the original VCF file (i.e. all the columns), ClinVar, COSMIC and SnpEff.

1.2. Extraction

In the process of extraction, the software extracts all possible meaningful data from the VCF file and stores them in the form of a multidimensional array. In order to understand the

Information in the INFO Column		**Corresponding key [2nd dimension] the elements is saved under**
123	→	Read Depth
CSF3R	→	Gene
NM_014350.2	→	Transcript
upstream_gene_variant	→	Consequence

extraction process, consider this line from an INFO column in a VCF file:

DP=123;EVS=10;CSQT=CSF3R|NM_014350.2|upstream_gene_variant — (I)

The following information contains various critical information that the program will have to extract and place in a multidimensional array. Using string manipulation and pattern recognition, the program will map the following information to preset keys within the multidimensional array:

A typical information array will have around 50 preset keys that reside in the second dimension of the array, with the first dimension being the file name uploaded (each file processed will occupy one element in the first dimension). Each key corresponds to a specific value that is nested within a string pattern. For example, in order to extract data that corresponds to the key "Read Depth" in the multidimensional array, a specific data extraction regular expression (regex) pattern that we can use to capture the wanted string is

DP=(\d+)[[:punct:]]. This would capture all digits between "DP=" and any punctuation, which in this case would refer to ";". Referring to the string (I) above, this regex pattern would allow us to capture "123", which corresponds to the Read Depth that we want. Using the string manipulation method described above, we are able to extract all the information that we want and enter all of them into the multidimensional array. This extraction will be done file by file and then more specifically gene by gene (e.g. CSF3R, DNMT3A, SF3B1). At the end of the extraction process, a multidimensional array would be returned with this structure:

```
Array (
        File_1 => Array(
                ['Gene'] => "ABCDE",
                ['Variant'] => "AB>CDE",
                ['Chr'] => "ChrM",
                        ⋮
                ['Ensemble ID'] => "ENST00000000"
        )
        File_2 => Array(…)
)
```

This array will be used in the next step to easily consolidate this data and present it in a humanly readable format.

1.3. Consolidation and Representation

In this step, the arrays will be converted into HTML Tables for the researchers' easy viewing. After addition and extraction (this happens during the wait time between "UploadFile.php" and "ViewDatabase.php", users will be directed from "UploadFile.php" to "ViewDatabase.php", where they can view the tables of data by clicking on the buttons with the corresponding filenames.

The interface is as shown in Figure 2 below.

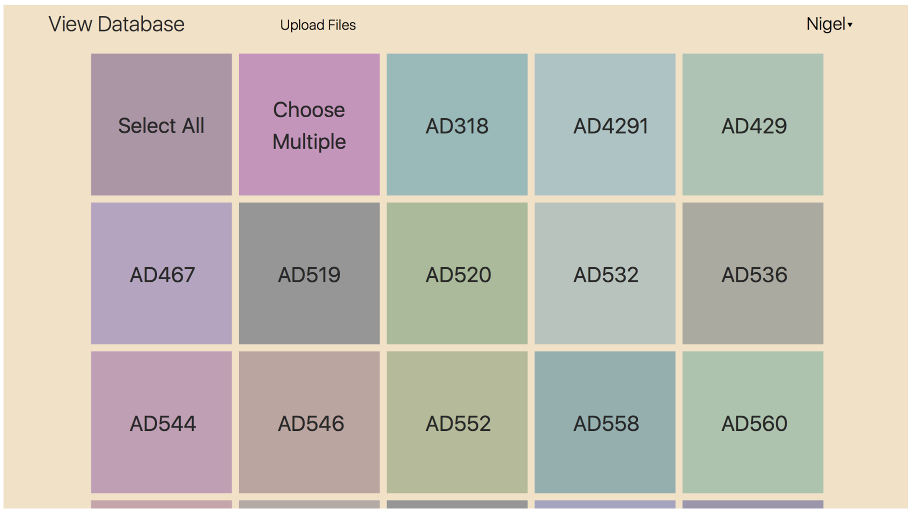


Figure 2: View Database Interface (Actual Website Interface: "ViewDatabase.php")

2. Filtering

This feature is seen in Figure 3, shown below. It is also constituted inside the "ViewDatabase.php" module.

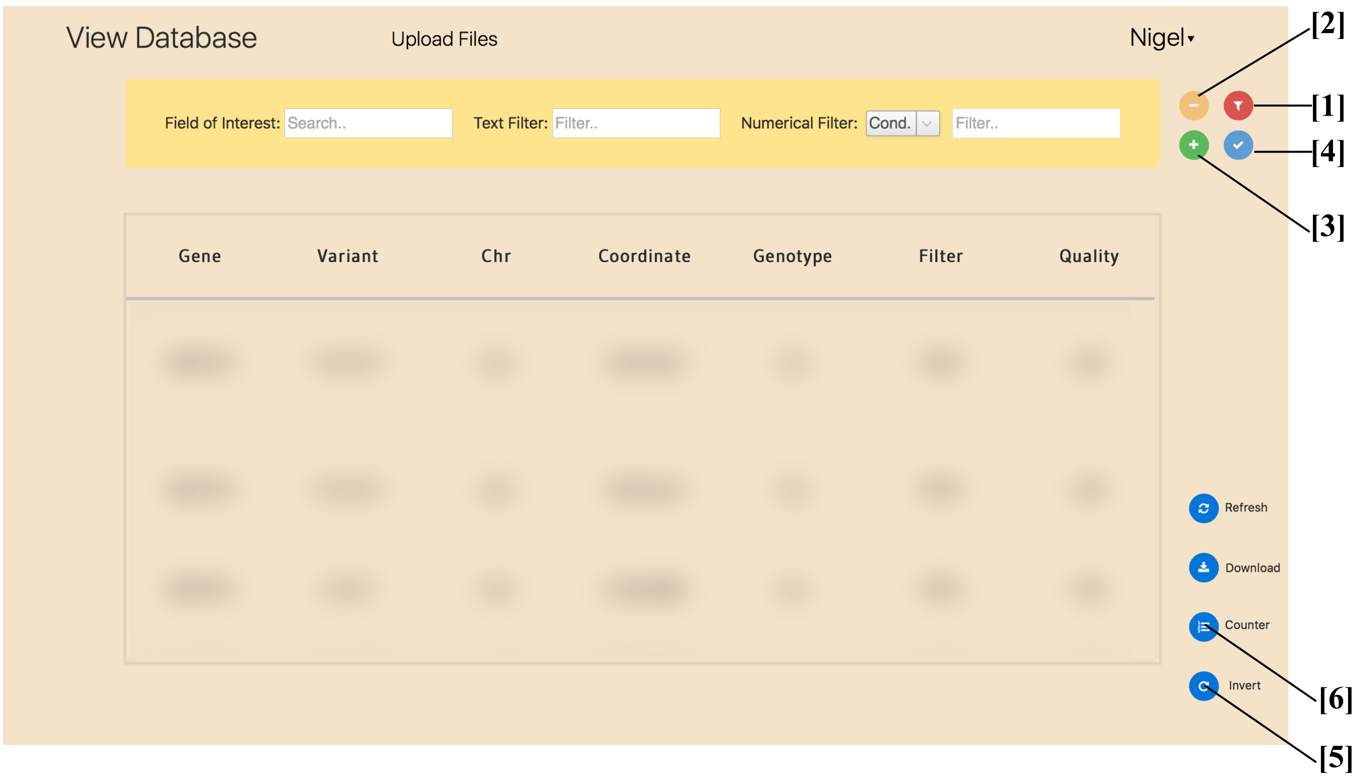


Figure 3: Filter Database Interface (Actual Website Interface: "ViewDatabase.php")

This feature consists of 2 sub features – generic filtering and default filtering. Default filtering (button [1]) follows a specific method of filtering that has been predefined for the convenience of the researcher. As of now, it has been predefined to the needs of the collaborating organisation, the National University Hospital. Generic filtering, on the other hand, can be further broken down into 2 parts – String Filtering and Numeric Filtering. The user will first have to key in their field of interest in the corresponding text box. Then, only the applicable filters will be activated for the user to access. For example, a search interest of "Gene" will only yield string filtering since the column "Gene" is comprised solely of strings. The mechanisms of each filtering technique is as such:

1. String Filtering

String filtering acts like a typical search box – it goes through the entire table and only returns those entries (under the field of interest selected) containing the string match. A point to note is that string filtering also encompasses matching of numbers, not just matching of words.

2. Numeric Filtering

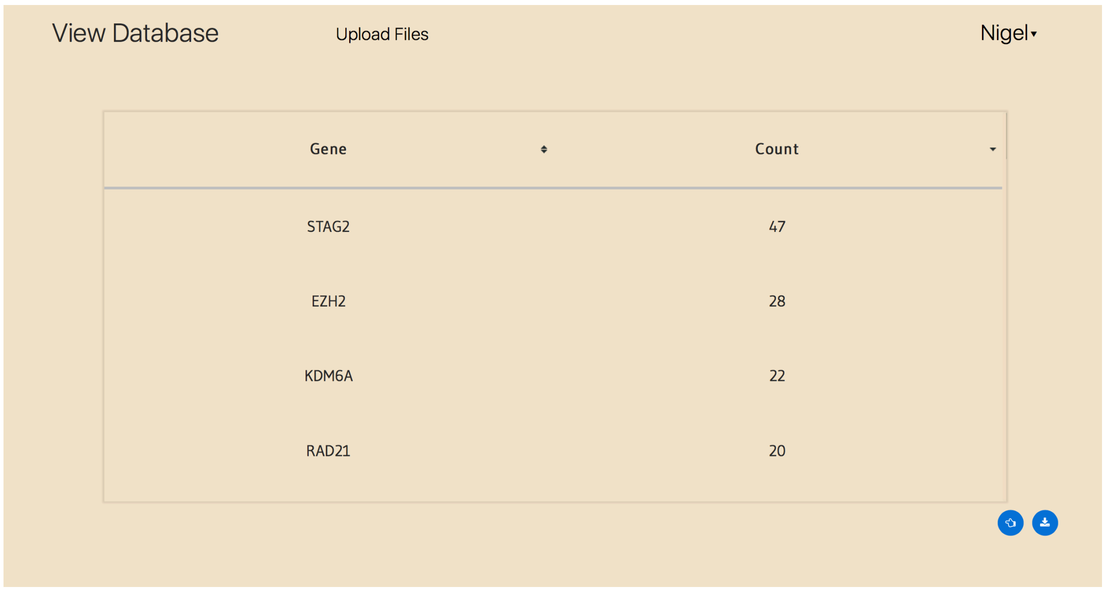
Numeric filtering involves conditional filtering. Conditional filtering is only allowed for fields which consist solely of numbers. For example, taking the "Filter" column as an example (as it consists solely of numbers), the user can select their preferred conditional from the "Cond." dropdown box (< , > , = , ≠) and input a numerical value to enable filtering. For example, if the conditional ">" is chosen and the number "99" is inputed, only all entries that are greater than 99 in the "Filter" column will shown.

These 2 filtering methods can be used together. Users can also add filtering criteria to other columns by clicking on the plus button (Button [3]) and remove filtering criteria by clicking the minus button (Button [2]). Once selection is complete, click on the blue tick button (Button [4]) to submit and a filtered table will be returned. The table can also be inverted (i.e. entries that are visible will become invisible and vice versa) should researchers deem it necessary (click on Button [5], the invert button to do so).

3. **Statistical Representation**

This feature aims to assist scientists in their research work by providing information on the occurrence frequency of mutations. In order to access this feature, users have to click button [6] (refer to Figure 3). The counter will prompt you to input up to 3 fields to submit for counting. The input limit has been set at 3 as having more fields will not add any meaningful value to the user experience since the similarities between entries across more than 3 fields would almost often return single counts (i.e. there are many counts that would end up with count "1").

The resultant user interface in the representation is as shown in Figure 4 below. Users can also download this set of data for offline safekeeping.



Gene	Count
STAG2	47
EZH2	28
KDM6A	22
RAD21	20

Figure 4: Count Database Interface (Actual Website Interface: "CounterDatabase.php")

Conclusion

This project aims to provide researchers and scientists a one-stop tool to annotate and analyse their genomic data. With the annotation tool, scientists will have an avenue to access all the information that matters to them and view them in a convenient format. They can also download the information should they require it for offline safekeeping. Furthermore, the filtering and statistical representation tools allows researchers to analyse the information to observe trends and draw inferences from the statistical observations.

Further development of this project can include graphical representations of genomic data, such as in the form of bar charts or line graphs since these forms of representation would make it even easier for researchers to conduct their analysis. Furthermore, pictorial representation of genes can also improve the overall user experience, since it can not only show frequency, but also the location of the mutation. This would make this tool more comprehensive as a whole.

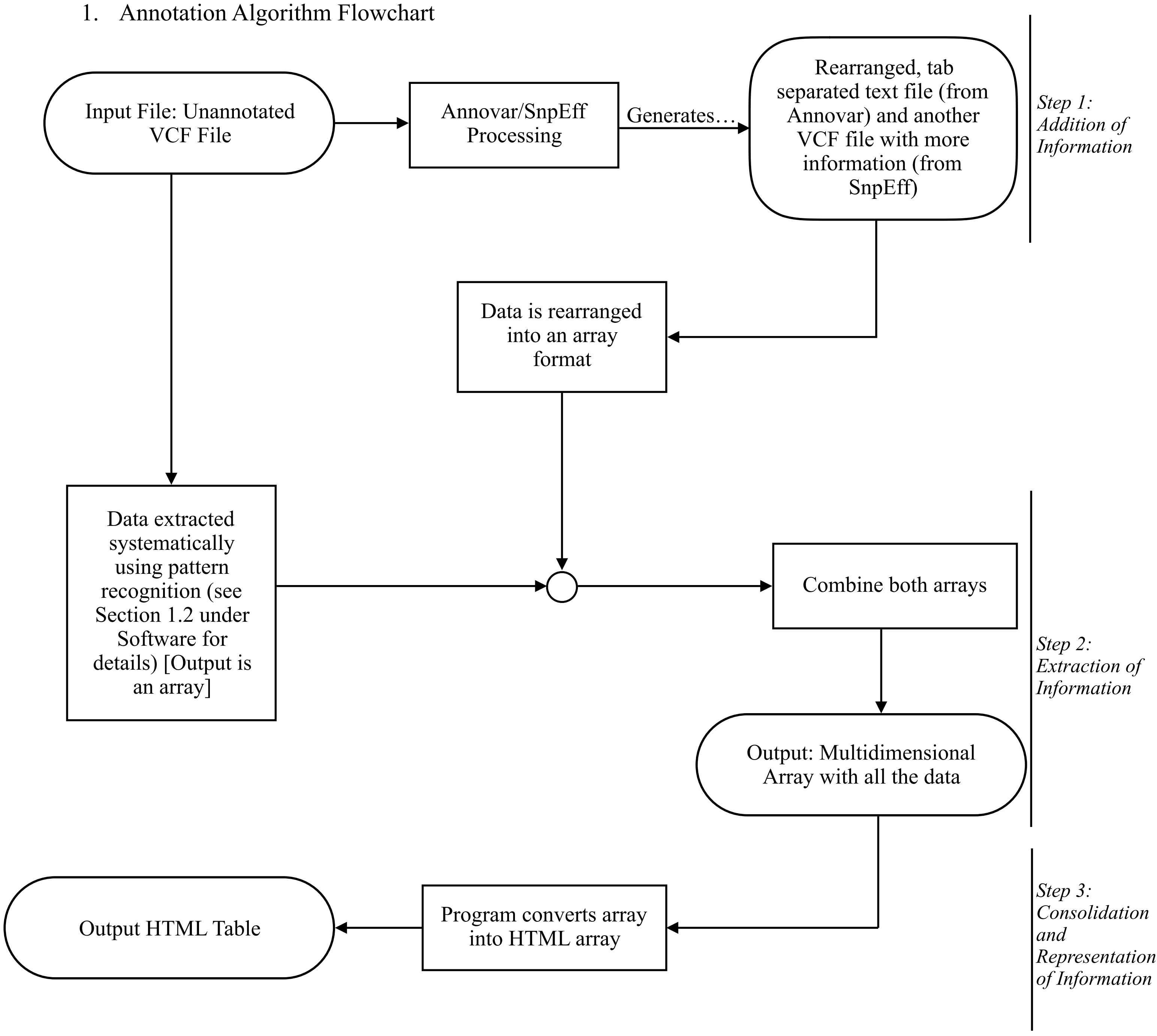
Some limitations of this project would include the processing speed of the server. Without a adequately efficient server, this project would not be able to run all its functions efficiently, resulting in lag time, which would result in long wait times for the user, especially in the annotation and filtering stages.

References

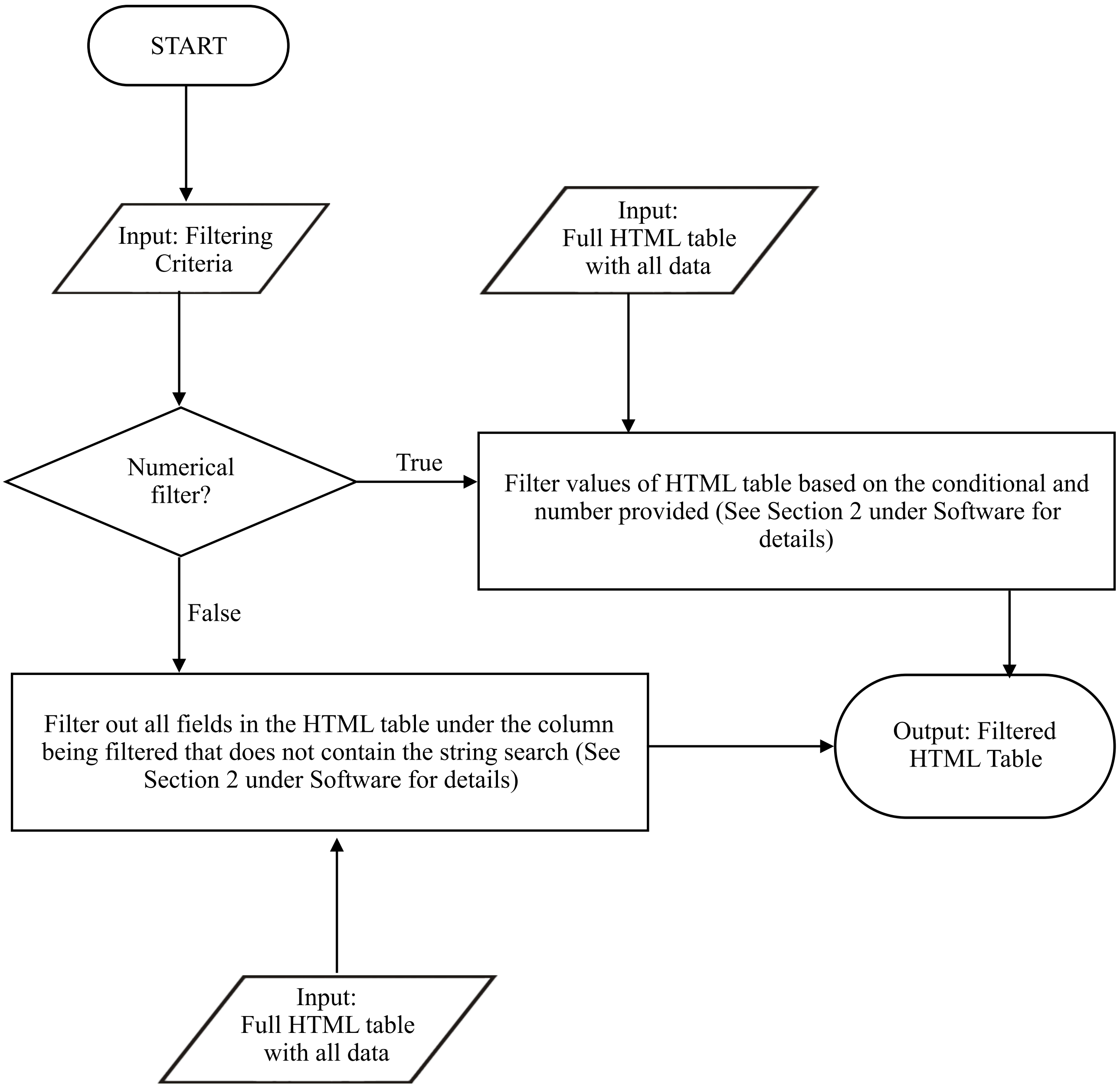
1. IGSR: The International Genome Sample Resource. (n.d.). Retrieved December 20, 2017, from http://www.internationalgenome.org/wiki/Analysis/Variant%20Call%20Format/VCF%20(Variant%20Call%20Format)%20version%204.0/encoding-structural-variants
2. Al, K. W. (n.d.). Retrieved December 21, 2017, from http://annovar.openbioinformatics.org/en/latest/
3. SnpEff. (n.d.). Retrieved December 21, 2017, from http://snpeff.sourceforge.net/

Appendices

1. Annotation Algorithm Flowchart


Input File: Unannotated VCF File
Annovar/SnpEff Processing
Generates…
Rearranged, tab separated text file (from Annovar) and another VCF file with more information (from SnpEff)
Step 1: Addition of Information
Data is rearranged into an array format
Data extracted systematically using pattern recognition (see Section 1.2 under Software for details) [Output is an array]
Combine both arrays
Step 2: Extraction of Information
Output: Multidimensional Array with all the data
Output HTML Table
Program converts array into HTML array
Step 3: Consolidation and Representation of Information

2. Filtering Algorithm Flowchart/Code


START
Input: Filtering Criteria
Input: Full HTML table with all data
Numerical filter?
True
Filter values of HTML table based on the conditional and number provided (See Section 2 under Software for details)
False
Filter out all fields in the HTML table under the column being filtered that does not contain the string search (See Section 2 under Software for details)
Output: Filtered HTML Table
Input: Full HTML table with all data

Code for numerical filter (Using "<" condition and Row 1 as example):

```
1 if($('.Conditionals').val() == '<'){
2         if(Number($(‘.FilterNumber').val()) > Number($(‘#Row1')).text())){
3                 if($(‘#Row1').is(':visible')){      //Ensure that the row is already visible
4                         $('#Row1').css('display','table-row');
5                 }
6         }else{
7                 $(‘#Row1').css('display','none');
8         }
9 }
```

Row 1: If the conditional selected is "<"

Row 2: If the field value is smaller than the filter (numerical) value

Row 3-8: Remove rows that does not fit criteria and ensure that only visible rows are filtered

Code for string filter (Using Row 1 as example):

```
1 if($(‘#Row1').text().indexOf($('.FilterText').val()) === -1){
2         $(‘#Row1').css('display','none');
3 }else{
4         if($('#Row1').is(':visible')){
5                 $(‘#Row1').css('display','table-row');
6         }
7 }
```

Row 1: If the row field value does not contain the search value $(‘.FilterText’).val()

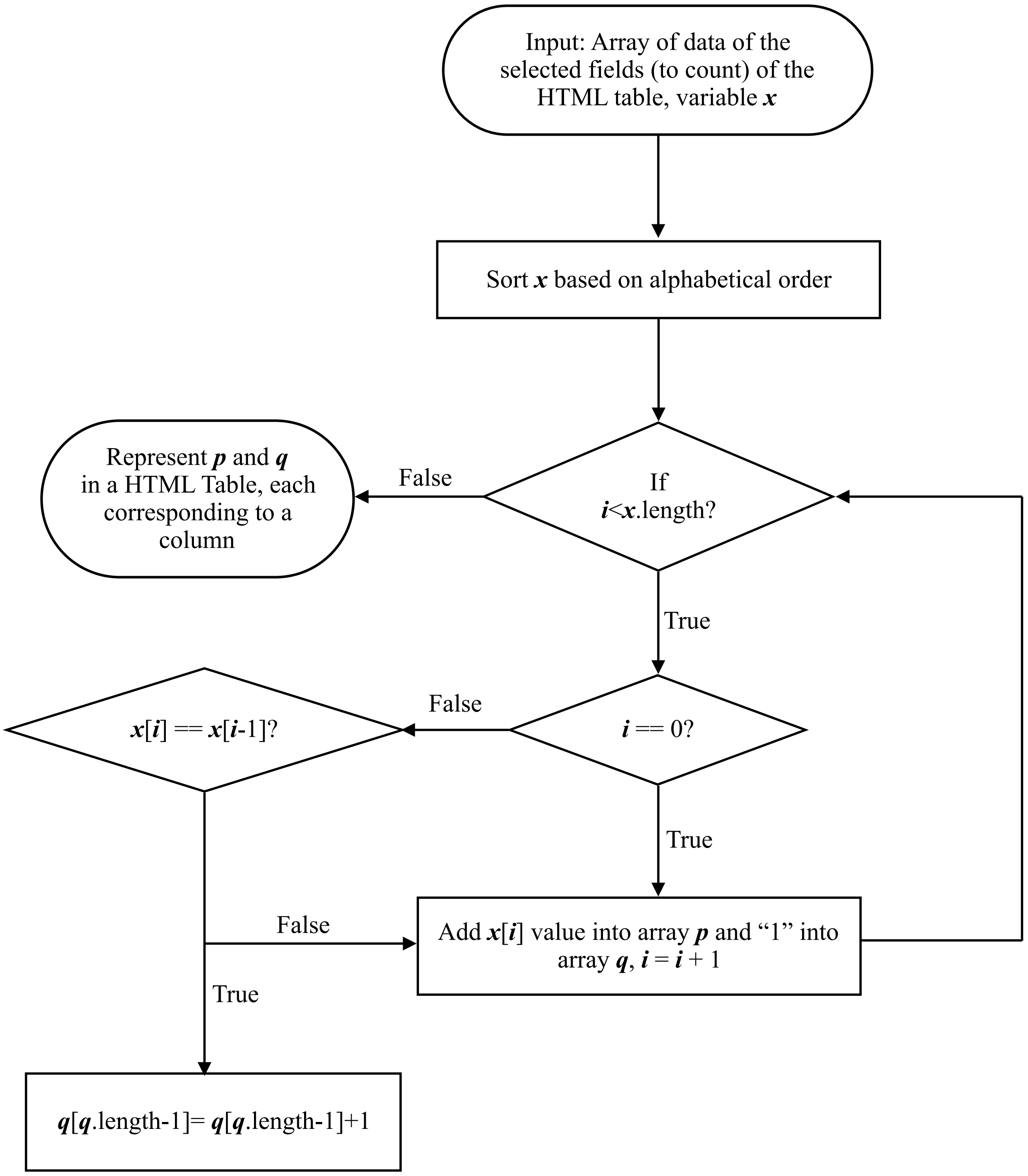
Row 2: Remove the row (visibility = 0)

Row 3: If the row field value contains the search value $(‘.FilterText’).val()

Row 4-6: Leave the row be (and ensure that only visible rows are filtered)

Note: Both codes are iterated through the entire table in order to output a filtered table.

3. Statistical Representation Algorithm Flowchart


Input: Array of data of the selected fields (to count) of the HTML table, variable ***x***
Sort ***x*** based on alphabetical order
If ***i***<***x***.length?
False
Represent ***p*** and ***q*** in a HTML Table, each corresponding to a column
True
i == 0?
False
x[***i***] == ***x***[***i***-1]?
True
Add ***x***[***i***] value into array ***p*** and "1" into array ***q***, ***i*** = ***i*** + 1
False
True
q[***q***.length-1]= ***q***[***q***.length-1]+1