Ramapo College

Virus Protein Interactions in Plants Database

Final Report

Ritwik Katiyar

Advanced Bioinformatics

Dr. Stuart

10th May 2019

Abstract

Viral proteins and viral Si-RNA's (Small interfering RNA's) are important to understand in order to come up with ways to counteract viruses. Hence, a database which could record viral proteins along with the type of host the virus infects could prove beneficial in understanding what the viruses are capable of and thereby giving a step ahead against counteracting viruses and protecting agricultural crops for viral diseases.

Introduction & Background

In 1892 a Russian botanist, Dimitrii Ivanovsky conducts and experiment showing that a pathogen other than bacteria was infecting the tobacco plants and that the organism was able to pass through porcelain filters that would normally retain bacteria. Then in 1898 Marcus Beijerinck made similar observations and concludes that the pathogen must be a distinct organism and officially coins the term “Virus” (Lecoq). Since then, the study of viruses has extended into its own subfield of virology. Many different viruses have been indentified, named and sequenced. Various viral databases have also been created that record the potential viral hosts, record viral sequence, or simply record the existence of a specific virus. Since the discovery of viruses we have been able to create numerous vaccinations and nearly eliminate major viral diseases in humans. We have also created numerous vaccinations for domestic animals as well as for wild animals and even for plants; although, plant vaccinations are simply referred to as pesticides. With the use of pesticides we have insured the safety of our crops. However, there is still a lot that we do not understand about viruses. (Smith)

Viruses are simple parasitic orgasms that require a host in order to survive. Since viruses do not possess a proper cellular structure, they require a host to replicate. Viruses come in various shapes and sizes and comprise mostly of either DNA or RNA. The nucleic acid sequence can also be single or double stranded DNA or RNA. Viruses also possess the capability to encode various types of proteins ranging from as little as 3-4 proteins to 100-200 proteins (Lodish). Viruses depend on a relatively simple mechanism to infect their hosts. First a virus breaks and enters through the membrane (or membranes) of the host cell. Since, viruses are usually extremely small and have a simple structure, a single virus only contains limited amount of RNA or DNA, in other words a very limited number of genes. Viruses use these limited genes

to not only code for proteins that can be used to simply allow the virus to replicate within a host cell, but also influence the proteins produced by the host cell (Smith) (Lodish). This allows the virus to hijack the host cell and replicate. Eventually replicating enough to burst out of the cell and infect other neighboring cells.

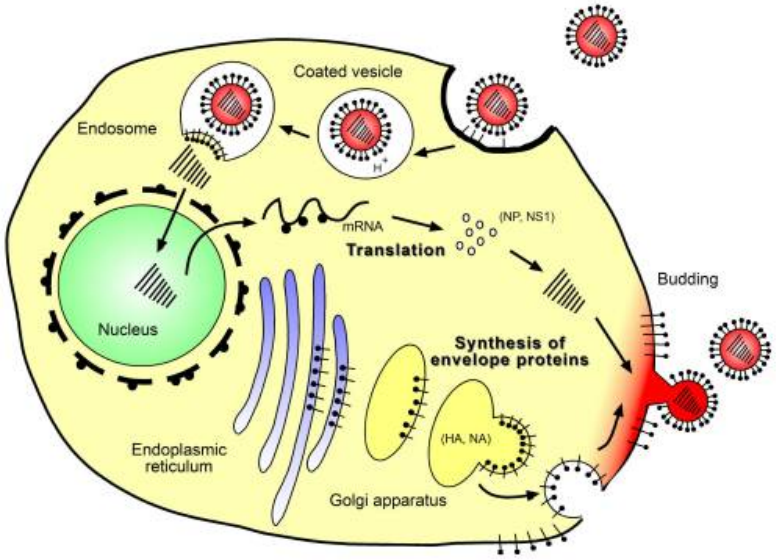


Figure 1: Replication cycle of influenza-A virus. Binding and entry of the virus, fusion with endosomal membrane and release of viral RNA, replication within the nucleus, synthesis of structural and envelope proteins, budding and release of virions capable of infecting neighboring epithelial cells. (Kamps)

Obviously many plants, animals and even some single celled organisms possess a natural anti viral defense systems. However, many viruses also come equipped with a counter host anti viral defense mechanisms.

One of the most important strategies of plants against viral infections is known as si-RNA- mediated (small interfering RNA mediated) gene silencing. Si-RNA-mediated gene silencing allows for defensive signal to spread to other neighboring cells otherwise the mechanism itself is similar to the mi-RNA (micro-RNA) mediated gene silencing. Plant mi-RNAs play two main functions in the antiviral defense; mi-RNAs target the viral RNA/DNA and

prevent the virus from reproducing, and mi-RNAs trigger the synthesis/biogenesis of si-RNA (which is responsible for the plant cells primary antiviral response). However, viruses have a counter defense system against the plant cells known as viral suppressors of RNA silencing (VSRs), which interfere with host RNA silencing in multiple ways. (However, VSRs additional functionality includes assistance in viral replication, encapsidation and movement.) VSRs primarily use two methods in dealing with plants defense system; first being the suppression and the assembly of AGOs (Argonautes; protein family that plays vital role in RNA silencing processes) into RISCs (RNA-induced silencing complex; meant to silence the viral DNA/RNA) and the second being interacting with AGOs (Argonautes) to degrade the protein. The figure below shows the two viral counters to the plant anti viral mechanisms in action. (Rui) (Kamthan)

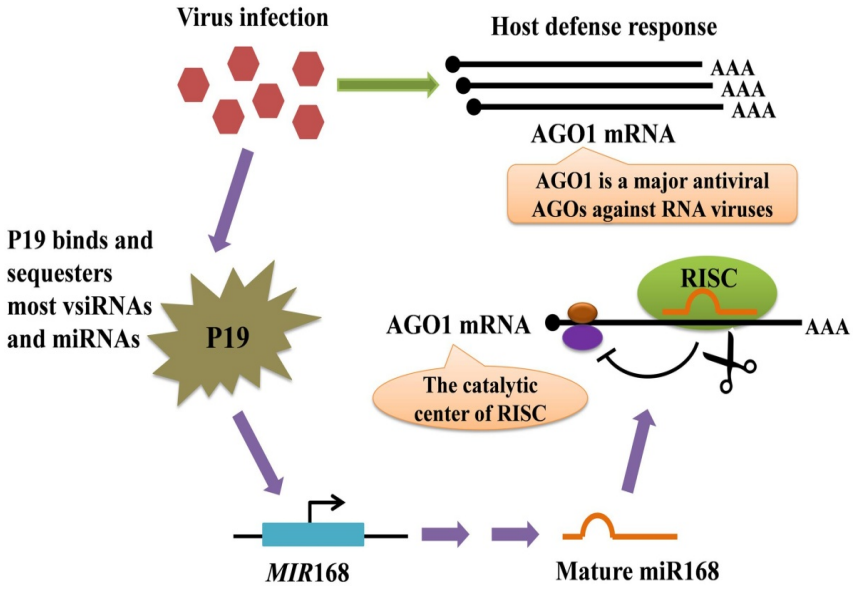


Figure 2: represents a model for the regulation of AGO mRNA level mediated by protein P19-induced miR168. The AGO mRNAs are made when a virus infection is detected within the cell. While the plant cell is creating mRNAs for AGO the virus produces P19 VSR which binds to the virus- encoded siRNAs and with host cell miRNAs which prevents the miRNA loading into AGO. The P19 protein however, does not bind to MIR168 micro RNA (MIR168, represses the AGO1 mRNA); resulting in low production of AGO1 mRNA. (Rui) (Pratt)

Based on the information, it is clear that viral proteins and viral si-RNA's are important to understand in order to come up with ways to counteract viruses, and there by protect

agricultural crops form viral diseases. Hence, a database which could record viral proteins along with the type of host the virus infects could prove beneficial in understanding what the viruses are capable of and thereby giving a step ahead against countering viruses and protecting agricultural crops; and thereby preventing massive crop loss due to viral outbreaks.

Requirements & Materials Used

Requirements & Materials	Summary
Visual Studio Code	Used to write all the code
Python and Bio-python	Python was used as a base to bio-python and postgresql. Bio-python was used to grab viral sequence information and viral IDs
Flask, HTML and CSS	To visualize all the data within the data and create an app to showcase all the data
Postgresql (Elephant SQL)	Database where the data was stored
R, R-studio and Taxize	Used to grab scientific names and common names of plants
Beautiful Soup	Used to grab information from Plant viruses online website
NCBI and Conserved Domains Database	NCBI was used from bio-python and CDD was used to search for various motifs and domains within a sequence of translated CDS's
Microsoft Excel	To better organize the raw data gathered from various sources
Matplotlib and Pandas	To create the graph on the home page of the website.

Table 1: Shows a comprehensive list of requirements for the following methods of setting up the database

Method

1. The first step to creating any such database is to develop a schema for the database.

```
Database : Vpipdb
Plants(Scientific Name, Common Name, PRIMARY KEY:(Scientific Name))
Infect(Infect_Unique_ID,Scientific Name, Virus_name, PRIMARY KEY:(Infect_Unique_ID))
Viruses(Virus_Name, Virus_ID, Sequence, PRIMARY KEY:(Virus_ID))
CDS(Virus_ID, CDS_UNIQUE_ID ,CDS_location, CDS_translatedseq, PRIMARY KEY:(CDS_UNIQUE_ID))
CDS_Analyze(UNIQUE_ID ,CDS_UNIQUE_ID, Start, End, E-Value, Bit-score, Accession#, Motifs_And_Domain_Name, PRIMARY KEY:(UNIQUE_ID))
```

Figure 3: displays the schema of the proposed database.

2. Next to select the plants for the database. Since there are numerous agricultural crops grown around the world and a lack of data linking a specific virus to a particular crop; it is important to focus on the top grown crops around the world. To do that, the data about the amount of agricultural crops grown around the world was taken from Food and agricultural organization (FAO). This data was then sorted and only the top 10 plants were kept.
3. FAO presents the data with the use of common names hence to get the scientific names of the data a simple R script was used that would grab the genus of the plants based on the common name.

```
#Finds taxonomic information on the following common names within the NCBI database
library("taxize")
set_entrez_key("8f31f1938108f0e82af1c7406f19d44fac08")
Sys.getenv("ENTREZ_KEY")

name <- c('Sugarcane', 'Corn', 'Rice', 'Wheat', 'Cucumbers', 'Tomatoes', 'Carrots', 'Spinach', 'Papay
uids <- get_uid(commonname)
uids.found <- as.uid(uids[!is.na(uids)])
commonname.found <- commonname[!is.na(uids)]
sciname <- sci2comm(name, db = 'ncbi')
names(sciname) <- commonname.found
sciname
```

Figure 4: Shows the R-script that finds taxonomic information from common names within NCBI

```
Saccharum       Sugarcane
Zea     Corn
Oryza   Rice
Glycine Soybean
Triticum        Wheat
Cucumis Cucumbers,Melons
Solanum Tomatoes,Potatos,Eggplants
Brassica        Cabbages,Cauliflowers
Daucus  Carrots
Spinacia        Spinach
```

Figure 5: The data was then organized using excel

4. With this information in hand a script was written using beautiful soup to gather information from the Plant and viruses online website. The data that was extracted from Plant and viruses online website was the data on plant susceptibility to a specific virus , all the plant species that the data was available for and all the viruses that were recorded by the website.

```
import lxml
from urllib.request import urlopen
from bs4 import BeautifulSoup

url = "http://sdb.im.ac.cn/vide/hostlist.htm"
html = urlopen(url)
soup = BeautifulSoup(html, 'lxml')
type(soup)

all_links = soup.find_all("a")
for link in all_links:
   if link.get("href") != None:
       temp = link.get("href")
       str(temp)
       f = open("Host_names.txt", "a")
       f.write(temp + "\n")
else:
    f = open("Host_names.txt", "a")
    f.write("None" + "\n")
```

Figure 6: The first script used to get the list of hosts used by the website

```
famly035.htm#Abelia grandiflora
famly082.htm#Abelmoschus esculentus
famly082.htm#Abelmoschus manihot
famly078.htm#Abrus precatorius
famly082.htm#Abutilon
famly082.htm#Abutilon hirtum
famly082.htm#Abutilon indicum
famly082.htm#Abutilon theophrasti
famly044.htm#Acanthospermum hispidum
famly002.htm#Acer palmatum
famly044.htm#Achillea filipendulina
famly044.htm#Achillea ptarmica
famly025.htm#Adansonia digitata
```

Figure 7: Shows the output from the script

```
Saccharum officinarum
Zea diploperennis
Zea mays
Zea mays ssp. mays
Zea mays ssp. mexicana
Zea perennis
Oryza australiensis
Oryza barthii
Oryza cubensis
Oryza glaberrima
```

Figure 8: Using Excel the output data was organized

```
import lxml
from urllib.request import urlopen
from bs4 import BeautifulSoup

url = "http://sdb.im.ac.cn/vide/sppindex.htm"
html = urlopen(url)
soup = BeautifulSoup(html, 'lxml')
type(soup)

all_links = soup.find_all("a")
for link in all_links:
        f = open("Virus_names.txt", "a")
        f.write(str(link) + "\n")
```

Figure 9: the second script used to extract all viruses that infect the said host from the website

Once the list of all plant and the viruses used by the Plant Viruses Online website was gathered, the list was then organized and arranged using excel to contain appropriate data. The viruses were also then linked with their host plants; figure 10 shows the final version of what the data looked like.

```
Saccharum officinarum   Maize dwarf mosaic potyvirus
Saccharum officinarum   Sugarcane bacilliform badnavirus
Saccharum officinarum   Sugarcane Fiji disease fijivirus
Saccharum officinarum   Sugarcane mosaic potyvirus
Saccharum officinarum   Sugarcane streak monogeminivirus
Zea mays        Barley stripe mosaic hordeivirus
Zea mays        Barley yellow dwarf luteovirus
Zea mays        Barley yellow striate mosaic cytorhabdovirus
Zea mays        Bermuda grass etched-line marafivirus
Zea mays        Brome mosaic bromovirus
```

Figure 10: shows how the data looks like once it's been organized in the tab delimited format. The first column shows the species of plant. The second column shows the virus that infects the plant.

5. With the use of a similar script to that shown in figure 4, the common names were determined for each and every plant.

Saccharum officinarum	Sugar Cane
Zea mays	Corn
Zea mays ssp. Mays	Common Corn-Variations
Zea mays ssp. mexicana	Corn- Subspecies
Oryza australiensis	Australian Rice
Oryza barthii	Wild Rice
Oryza rufipogon	Wild Red Rice
Oryza glaberrima	African Rice
Oryza latifolia	Broadleaf Rice
Oryza longistaminata	Longstamin wild rice
Oryza nivara	Wild Asian Rice
Oryza perennis	Brown Bread Rice
Oryza punctata	Red Rice

Figure 11: The data for the plants table mentioned in the schema in figure 3. Scientific Names on the left and the common names on the right

6. Once the viruses were connected to their respective host plants the NCBI viruses ID's was determined using a script. Then another script was used to grab the sequences for the viruses based on the determined ID.

```
import csv
import Bio
from Bio import Entrez

Entrez.email = "rkatiyar@ramapo.edu"
textfile = open("10.txt", "r")
reader = csv.reader(textfile)
allRows = [row for row in reader]

for i in range(len(allRows)):
    handle = Entrez.esearch(db = "nucleotide", term = allRows[i], retmax = 1)
    record = Entrez.read(handle)
    handle.close()
    idList = record["IdList"]
    f = open("idrecords10.txt", "a")
    f.write(str(idList)+ "\n")
```

Figure 12: Shows the script that extracts a single NCBI ID from the search of the virus

.

```
#Gets Sequences from the NCBI- ID
import csv
import Bio
from Bio import SeqIO
from Bio import Entrez

Entrez.email = "rkatiyar@ramapo.edu"
Entrez.api_key = "8f31f1938108f0e82af1c7406f19d44fac08"

textfile = open("ID_only_1.txt", "r")
reader = csv.reader(textfile)
all_ID = [row for row in reader]

for i in range(len(all_ID)):
        handle = Entrez.efetch(db = "nucleotide", id = all_ID[i], rettype = "gb", retmode = "text")
        record = SeqIO.read(handle, "genbank")
        handle.close()
        Sequence = record.seq
        f = open("seq_list.txt", "a")
        f.write(str(Sequence) + "\n")
```

Figure 13: shows the script used to extract sequence information based on the ID obtained from previous script

7. The resulting data was then arranged using excel

Maize dwarf mosaic potyvirus	['18490052']	AAAAACAACAAGACTCAACACAACACAACCAAACACGA
Maize rayado fino marafivirus	['14141972']	GTCGACGTCGCATTCTGCACCAGCTTTCGCTCGTCCAGAA
Rice tungro bacilliform badnavirus	['18026839']	TGGTATCAGAGCGATGTTCGAACTTTAAGGGAAAATAGA
Rice dwarf phytoreovirus	['20428565']	GGCAAAACCTCGCCATGGCTTATCCTAACGACGTCAGAA
Rice ragged stunt oryzavirus	['20428612']	GATAAATCTCCATGACTCTATTAGTGATCACCGAGCAGAC
Rice grassy stunt tenuivirus	['9635243']	ACACAAAGTCCTGGGCAATTACAAACAAGAAAAACTTAA
Barley stripe mosaic hordeivirus	['19744922']	GTAAAAGAAAAGGAACAACCCTGTTGTTGTTCGACGCTA
Glycine mosaic comovirus	['944542958']	TATTAAAATCTTTATAAGATTTTGATAACCGCAATCATAA
Glycine mottle carmovirus	['216905810']	GGGTAACCCAGCCAGTTATCCACCATTTAATCTTTCAGGA
Soybean mosaic potyvirus	['1591440922']	AAATTAAAACTAGTTATAAAGACAACAAACAAATTAAAC
Barley yellow striate mosaic cytorhabdovirus	['1586082777']	CACGACCAGTGATCGTATAATTTGATTATTGGTGATCCTC
Abutilon mosaic bigeminivirus	['1464307913']	ACCGGATGGCCGCGAAATTTTTGGTGTCCAGAACTTTAAT
Melon Ourmia ourmiavirus	['194351521']	CCCAGATTACGGTATCTTTCGACACCGCAAGAGCGAACTT
Heracleum latent trichovirus	['1464311295']	TCCCTCCGATTATGAGGCGTTCGATCGTAGGTCAAGATGA
Tomato spotted wilt tospovirus	['1389531354']	AGAGCAATTGTGTCAATTTTATTCAAACCTTAACACTCAG
Peach rosette mosaic nepovirus	['1159189057']	GGAAAAAACCAAAGTTGTTTTCCTTTGACTGCCTTTTGTTT
Alfalfa mosaic alfamovirus	['1561349898']	ACTATGCTGCCTTGCGCAAAGCTCAACTGCCGAAACCTCC
Datura Colombian potyvirus	['1572772023']	CAAACTTGGGTTGTGGAATGGCTCACTTAAAGCTGAATTG
Beet pseudo-yellows closterovirus	['268529022']	ATAAATTTATCCTTAGGGTTAAAGAAAGTTTTCCTCCCCCC

Figure 14: shows how the data looked in excel. The first column shows the name of the virus. The second column shows the ID found. The third column shows the complete sequence of the virus.

8. After the sequences were obtained the next step was to obtain information on potential CDS regions for each virus. Not all viruses contain a CDS region and for viruses without a CDS region no further data was necessary to obtain.

```
import csv
import Bio
from Bio import Entrez
from Bio import SeqIO

Entrez.email = "rkatiyar@ramapo.edu"
Entrez.api_key = "8f31f1938108f0e82af1c7406f19d44fac08"

textfile = open("Virus_ID.txt", "r")
reader = csv.reader(textfile)
all_ID = [row for row in reader]

for i in range(len(all_ID)):
    with Entrez.efetch(db="nucleotide", rettype="gb", retmode="text",id= all_ID[i]) as handle:
        for seq_record in SeqIO.parse(handle, "gb"):
            f = open("CDS_loc.txt", "a")
            f.write(str(all_ID[i]) + "\n")
            for feature in seq_record.features:
                if feature.type == 'CDS':
                    f.write(str(feature.location) + "\n")
```

Figure 15: Shows the script for getting the CDS region for each virus

18490052	CDS_01	139	9265	MAGTWTHVTHKWQPNLDNPRDVRRIMELFAAKGQVYDEKRALEHNSKLLRRAQVVDVEPMITVQPKKCAQIV
18490052	CDS_02	2682	2922	NLCRSVESSVDRIIIVWKILRNMACVQGQEILQAIFNPAKKRRFRRCLQYISYASNIKFSAEKSQSSQLYFNQTPPRI
14141972	CDS_03	96	6180	MSSFLRGGHLLSGVESLTPTTHRDTITAPIVESLATPLRRSLERYPWSIPKEFHSFLHTCGVDISGFGHAAHPHPVHK
14141972	CDS_04	301	1561	MPLTPTPSIRPSRPTSFSMSGPTTLGVRQTSCSSSLRSSPSSSPDSPTSPTSSTTGSCPKTPPGTPPLPRTSRTARPSS
18026839	CDS_05	67	667	VLKRNLTSQNIESRYEKLEFLDLAVWGKEKKQKYLLSTDNISFYCYFDTSKTSESERKHTFHSDNKQLNSIVDLIIKHSI
18026839	CDS_06	663	996	MSADYPTFKEALEKFKNLESDTAAKDKFNWVFTLENIKTTADVNLASKGLVQLYALQEIDKKINNLTAQVSKLPTT
18026839	CDS_07	992	6026	MSLRPFTGTSRTITQDSTSESNIKKGKNSTKRELIEEVDVNQDVENFDWKKLSGIKPNKLYEKNWQEKVKLKQQSI
18026839	CDS_08	6046	7216	MNIEYPYSIHIIDKNKVPIYDQGNLFHTEKSARLSHESRGLLDHLFTFSSDNTERVRKLHILADYLYLLESERESYKNEV
20428565	CDS_09	14	3365	MAYPNDVRNVWDVYNVFRDVPNREHLIRDIRNGLVTVRNLTNMLTNMERDDQLIIAQLSNMMKSLSIGIEKAQ
20428612	CDS_10	11	3779	MTLLVITEQTIHSLCLDHGETNQIIAEIKQLEKPELLFSYITDAEPLATGEVFVGPDICGNCITHTFRVPDYVAKPPPYD
20428612	CDS_11	490	1471	MPSVRASDPKQQLYPIAELEQLSQEDVRNIAKTHRSPTLLRTQTKFCVGRRGELSYDTTLAPLQYSEPGDSILRGSDI
9635243	CDS_12	74	614	MSKSHSDVVGTVSGLNYRLFYDMIPDRISQKLRLREITDPKTCNASKIPLVLKAAEEVSRMDIDHDKDGYTKVQVK
9635243	CDS_13	1527	2505	MALLQKLGSSKVSSKRMSPAMIPLDSINQDLVDPQQEKDAKNKKEGKKKDLDVSMDPLTGKLPLGKKKQVDTGC
19744922	CDS_14	89	686	MPNVSLTAKGGGHYIEDQWDTQVVEAGVFDDWWVHVEAWNKFLDNLRGINFSVASSRSQVAEYLAALDRDLI
19744922	CDS_15	803	2390	MDMTKTVEEKKTNGTDSVKGVFENSTIPKVPTGQEMGGDGSSTSKLKETLKVADQTPLSVDNGAKSKLDSSDRQ
19744922	CDS_16	2358	2754	MKTTVGSRPNKYWPIVAGIGVVGLFAYLIFSNQKHSTESGDNIHKFANGGSYRDGSKSISYNRNHPFAYGNASSP
19744922	CDS_17	2563	3031	MAMPHPLECCCPQCLPSSESFPIYGEQEIPCSETQAETTPVEKTVRANVLTDILDDHYYAILASLFIIALWLLYIYLSSII
944542958	CDS_18	205	3502	MSIPEAKYRTYWFSDNYVKYVPTAWQTDTGHTWARICELRVERFRNAFDSRFDFGQTEWRTRRDISDTIFLHTTC
216905810	CDS_19	39	2280	MMIQFGTMPPVRIDGPPRAFSMGMALRAVGKFLVNCCSAFGDNWADSITRNRTQHVCALQYFGDLPIECIVKS
216905810	CDS_20	2279	2486	MDKSPQRGRSRSRSRQTQGPKGPKPENKQIQVAHHAVDKARGKPPGGDHGGDFVIVAHTVTVNINFNI

Figure 16: Shows how the data looks in excel. The first column shows the Virus_ID. The second column shows the Unique_ID given to every CDS region found. The third column shows the length/ region of CDS within the viral genome. Finally, the last column is the translated protein sequence.

9. Using **NCBI: CDD** data on possible Motifs and Domains within the Translated Protein Sequence was obtained. NCBI:CDD allows for batch sequence searches which looked like:

Figure 17: Shows the CDD batch search.

Vpipmotdom_01	VpipCDS_01	260	693	9.84E-104	340.818	cl20022	Peptidase C6 superfamily
Vpipmotdom_02	VpipCDS_01	2254	2715	2.34E-98	326.635	cl02808	RT like superfamily
Vpipmotdom_03	VpipCDS_01	2806	3038	1.58E-91	297.593	cl02961	Poty coat superfamily
Vpipmotdom_04	VpipCDS_01	1495	1768	5.16E-68	231.607	cl07169	Poty PP superfamily
Vpipmotdom_05	VpipCDS_01	1986	2219	3.71E-57	199.161	cl24133	Peptidase C4 superfamily
Vpipmotdom_06	VpipCDS_01	2	232	2.52E-54	191.006	pfam01577	Peptidase S30
Vpipmotdom_07	VpipCDS_01	708	1160	1.11E-53	196.399	cl16319	Potyvirid-P3 superfamily
Vpipmotdom_08	VpipCDS_01	1198	1329	4.91E-12	66.0336	cl28899	DEAD-like helicase N superfamily
Vpipmotdom_09	VpipCDS_01	1186	1334	2.03E-24	103.341	smart00487	DEXDc
Vpipmotdom_10	VpipCDS_01	1372	1468	3.04E-09	55.6809	smart00490	HELICc
Vpipmotdom_11	VpipCDS_03	45	325	2.26E-78	261.839	pfam01660	Vmethyltransf
Vpipmotdom_12	VpipCDS_03	722	817	2.55E-16	76.3005	cl05113	Peptidase C21 superfamily
Vpipmotdom_13	VpipCDS_03	1450	1639	3.45E-10	64.2033	cl03049	RdRP 2 superfamily
Vpipmotdom_14	VpipCDS_03	1840	1995	4.22E-10	60.6161	cl03052	Tymo coat superfamily
Vpipmotdom_15	VpipCDS_03	1090	1140	0.00610147	37.156	cl38915	DEAD-like helicase C superfamily
Vpipmotdom_16	VpipCDS_03	908	1138	6.33E-42	154.07	pfam01443	Viral helicase1
Vpipmotdom_17	VpipCDS_04	2	417	0.00019676	43.7737	PHA03247	PHA03247
Vpipmotdom_18	VpipCDS_05	85	194	0.00705485	36.7863	cl26680	SidE superfamily
Vpipmotdom_19	VpipCDS_06	1	110	2.97E-71	207.393	cl05711	RTBV P12 superfamily
Vpipmotdom_20	VpipCDS_07	1240	1389	4.84E-44	157.757	cd01647	RT LTR
Vpipmotdom_21	VpipCDS_07	1488	1614	2.58E-16	76.3773	cl14782	RNase H like superfamily
Vpipmotdom_22	VpipCDS_07	976	1073	2.38E-05	44.2496	cd00303	retropepsin like
Vpipmotdom_23	VpipCDS_07	1248	1392	1.97E-35	133.572	pfam00078	RVT 1
Vpipmotdom_24	VpipCDS_08	1	389	0	823.578	cl05630	RTBV P46 superfamily
Vpipmotdom_25	VpipCDS_12	9	168	2.78E-21	85.1108	cl20278	Tenui NCP superfamily
Vpipmotdom_26	VpipCDS_13	67	271	3.12E-24	99.4412	cl03993	Tenui NS4 superfamily
Vpipmotdom_27	VpipCDS_14	18	183	1.52E-64	196.31	pfam00721	TMV coat
Vpipmotdom_28	VpipCDS_15	266	359	7.12E-06	46.0084	cl28899	DEAD-like helicase N superfamily

Figure 18: shows the final results that were obtained from the NCBI: CDD database. These results are edited and arranged based on the schema. Look at image 1 for reference.

10. Now that all the data was gathered, the next step was to place the data in the database so that further queries can be written to allow for greater analysis of the data.

```python
#Create tables and populates them
import csv
import psycopg2
import psycopg2.extras
from urllib.parse import urlparse, uses_netloc
import configparser
#initalize connection to the database
def connect_to_db(conn_str):
    uses_netloc.append('postgres')
    url = urlparse(conn_str)

    conn = psycopg2.connect(database=url.path[1:],
                            user=url.username,
                            password=url.password,
                            host=url.hostname,
                            port=url.port)

    return conn
#Load connection infomration from Config.ini
config = configparser.ConfigParser()
config.read('config.ini')
connection_string = config['database']['postgres_connection']
conn = connect_to_db(connection_string)
cursor = conn.cursor()
#Create the tables and columns

cursor.execute("""
    CREATE TABLE IF NOT EXISTS Plants (ScientificName VARCHAR(255) NOT NULL, CommonName VARCHAR(255) NOT NULL,
    PRIMARY KEY(ScientificName))
""")
cursor.execute("""
    CREATE TABLE IF NOT EXISTS Infect (Infection_ID VARCHAR(20) NOT NULL, ScientificName VARCHAR(255) NOT NULL,
    VirusName VARCHAR(255) NOT NULL , PRIMARY KEY(Infection_ID),
    FOREIGN KEY (ScientificName) REFERENCES Plants(ScientificName)
    ON DELETE CASCADE ON UPDATE CASCADE
    DEFERRABLE INITIALLY DEFERRED)
""")
cursor.execute("""
    CREATE TABLE IF NOT EXISTS Viruses (Name VARCHAR(255) NOT NULL, ID INTEGER NOT NULL,
    Sequence TEXT, PRIMARY KEY (ID))
""")
cursor.execute("""
    CREATE TABLE IF NOT EXISTS CDS (VirusID INTEGER NOT NULL, CDSID VARCHAR(20) NOT NULL, CDSStart INTEGER,
    CDSStop INTEGER, CDSTranslatedSeq TEXT, PRIMARY KEY (CDSID),
    FOREIGN KEY (VirusID) REFERENCES Viruses(ID)
    ON DELETE CASCADE ON UPDATE CASCADE
    DEFERRABLE INITIALLY DEFERRED)
""")
cursor.execute("""
    CREATE TABLE IF NOT EXISTS CDSResult (CDSResultID VARCHAR(20) NOT NULL, CDSUniqueID VARCHAR(20) NOT NULL,
    MotifStart INTEGER, MotifStop INTEGER, EValue DOUBLE PRECISION, BitScore DOUBLE PRECISION,
    AccessionNo VARCHAR(50), MotifName VARCHAR(225), PRIMARY KEY (CDSResultID),
```

```
    MotifStart INTEGER, MotifStop INTEGER, EValue DOUBLE PRECISION, BitScore DOUBLE PRECISION,
    AccessionNo VARCHAR(50), MotifName VARCHAR(225), PRIMARY KEY (CDSResultID),
    FOREIGN KEY (CDSUniqueID) REFERENCES CDS(CDSID)
    ON DELETE CASCADE ON UPDATE CASCADE
    DEFERRABLE INITIALLY DEFERRED)
""")
#Insert the data into the tables
with open('Plants.csv', 'r') as Plants:
    reader = csv.reader(Plants)
    for row in reader:
        cursor.execute('INSERT INTO Plants VALUES (%s,%s)',row)

with open('Infect.csv', 'r') as Infects:
    reader = csv.reader(Infects)
    for row in reader:
        cursor.execute('INSERT INTO Infect VALUES (%s,%s,%s)',row)

with open('Viruses.csv', 'r') as Viruses:
    reader = csv.reader(Viruses)
    for row in reader:
        cursor.execute('INSERT INTO Viruses VALUES (%s,%s,%s)',row)

with open('CDS.csv', 'r') as CDS:
    reader = csv.reader(CDS)
    for row in reader:
        cursor.execute('INSERT INTO CDS VALUES (%s,%s,%s,%s,%s)',row)

with open('CDS_Motifs.csv', 'r') as CDS_Motifs:
    reader = csv.reader(CDS_Motifs)
    for row in reader:
        cursor.execute('INSERT INTO CDSResult VALUES (%s,%s,%s,%s,%s,%s,%s,%s)',row)
# Commit and close the connection after the queries
conn.commit()
conn.close()
```

Figure 19 & 20: Shows the scrip used to seed the database.

11. Next to display the data in a table properly from the database a Flask application was set up and quires were written to allow user to further use the gathered data and to create a web interface.

```python
from flask import Flask, render_template, request, redirect
from .Queries import *
app = Flask(__name__)
#Calls function that connects to the database and loads all data onto the database
#At initialization of the app the user is first directed to the home page
################## Main Page ###################
@app.route('/')
def Home():
    return render_template('Home.html')
################## Home Pages ##################
@app.route('/Home/Common')
def Home_Common():
    return render_template('Search/Home-Common.html')

@app.route('/Home/Virus')
def Home_Virus():
    return render_template('Search/Home-Virus.html')

@app.route('/Home/Motif')
def Home_Motif():
    return render_template('Search/Home-Motif.html')
################ Search Results ################
@app.route('/Scientific Name/Result/', methods=['GET'])
def search_results_Scientific():
    searchs = request.args.get('search')
    Results = search_scientific(searchs)
    return render_template("Result/Scientific-Result.html",Results = Results)

@app.route('/Common Name/Result/', methods=['GET'])
def search_results_Common():
    searchs = request.args.get('search')
    Results = search_full(searchs)
    return render_template("Result/Common-Result.html",Results = Results)

@app.route('/Virus/Result/', methods=['GET'])
def search_results_Virus():
    searchs = request.args.get('search')
    Results = search_virus(searchs)
    return render_template("Result/Virus-Result.html",Results = Results)

@app.route('/Motif/Result/', methods=['GET'])
def search_results_Motif():
    searchs = request.args.get('search')
    Results = search_motif(searchs)
    return render_template("Result/Motif-Result.html",Results = Results)
################ About Page ###################
@app.route('/About/')
def About():
    return render_template('About/About.html')
################## View Plants #################
@app.route('/Plants/')
def Plants():
```

Figure 21: Shows the main flask app code. For full code simply open the “Main.py” file in the application folder.

```python
#Sends queries to the database to recieve infomration
import psycopg2
import pandas as pd
import psycopg2.extras
from urllib.parse import urlparse, uses_netloc
import configparser
#initalize connection to the database
def connect_to_db(conn_str):
    uses_netloc.append('postgres')
    url = urlparse(conn_str)
    conn = psycopg2.connect(database=url.path[1:],
                            user=url.username,
                            password=url.password,
                            host=url.hostname,
                            port=url.port)
    return conn
#Load connection infomration from Config.ini
config = configparser.ConfigParser()
config.read('config.ini')
connection_string = config['database']['postgres_connection']
conn = connect_to_db(connection_string)
#Get all data from plants
def get_plants():
    with conn.cursor(cursor_factory=psycopg2.extras.RealDictCursor) as cursor:
        cursor.execute('SELECT * FROM Plants')
        for d in cursor:
            yield d
#Get all data from viruses
def get_viruses():
    with conn.cursor(cursor_factory=psycopg2.extras.RealDictCursor) as cursor:
        cursor.execute('SELECT * FROM Viruses')
        for d in cursor:
            yield d
#Get Just the scientifc name and virus name from infects
def get_infects():
    with conn.cursor(cursor_factory=psycopg2.extras.RealDictCursor) as cursor:
        cursor.execute('SELECT scientificname,virusname FROM Infect')
        for d in cursor:
            yield d
#Get all data from the CDS as well as the virus table.
#To display the name of the virusID
def get_CDS():
    with conn.cursor(cursor_factory=psycopg2.extras.RealDictCursor) as cursor:
        cursor.execute('SELECT * FROM cds JOIN viruses ON  cds.virusid = viruses.id')
        for d in cursor:
            yield d
#Get all data from the motifs_domain
def get_motifs_domain():
    with conn.cursor(cursor_factory=psycopg2.extras.RealDictCursor) as cursor:
        cursor.execute('SELECT * FROM CDSresult\
                        JOIN cds ON cdsresult.cdsuniqueid  = cds.cdsid\
                        JOIN viruses ON  cds.virusid = viruses.id')
```

Figure 22: The figure shows the queries that were then called to the database and displayed on the flask application using HTML (to display on browser) and CSS (to style the display).

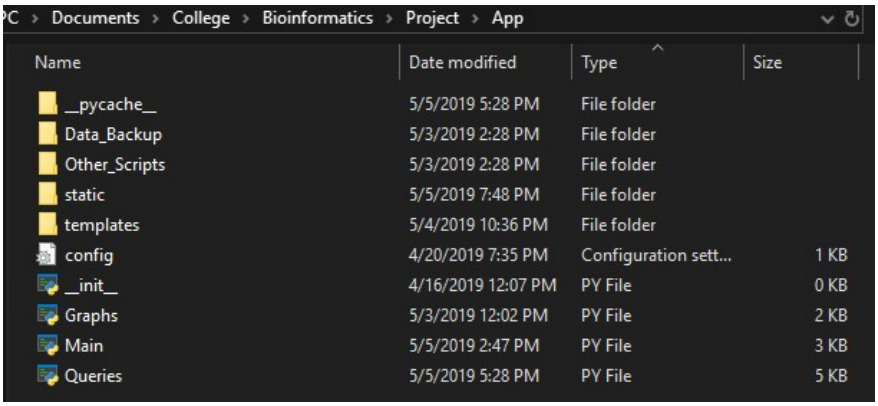
Results

Over All Statistics for the application

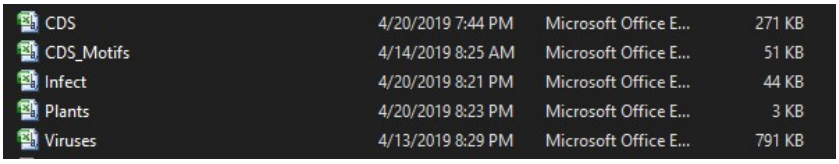
10 Plant genuses and 73 Plant species were recorded
193 Viruses were recorded that infect at-least one plant species
Over all 764 Recorded infections
486 Coding regions identified for various viruses
717 Various motifs identified from the coding regions

Folder structure of the application

a.



b.



c.

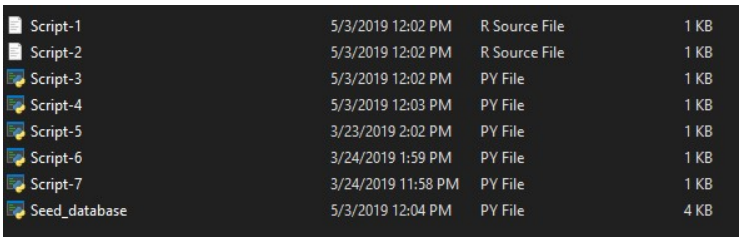


Figure 23 – a: Shows the folder structure of the application and how the scripts were arranged. The _pycache_ folder is created at use of the application and holds information on the usage of the app. The Data_Backup folder (figure 23 - b) contains all the data

that was gathered in csv format and tab-delimited format. The Other_Scripts folder (figure 23- c) contains all the scripts that were used for gathering the data including script used to seed the database. Static and Templates contain CSS and HTML files for the application. _init_.py file simply initializes the application and should be empty. The Graphs.py script creates the graph based on the data in the database. Main.py is the main application and is called to run the application. Queries.py connects the database and the application together and calls queries to the database.

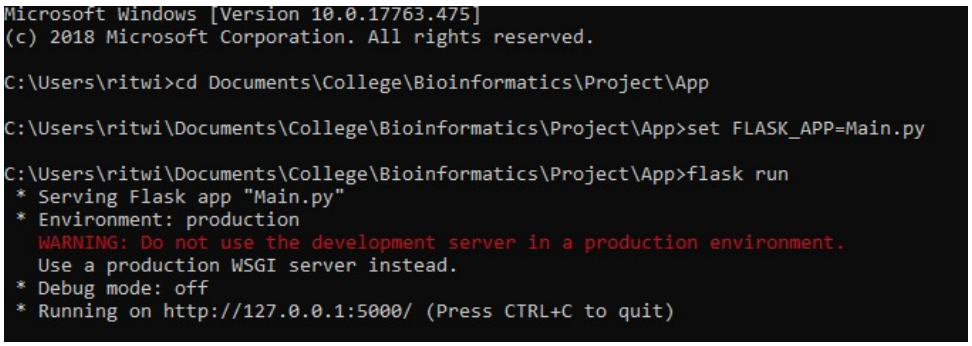


Figure 24: Shows how to run the application once everything was set up.

Application

Home and About pages

The application contains a home page, about page, a browse section (allows the user to browse the all the data that was gathered for the project) and a search bar (which can be set according to what the user wants to search for.

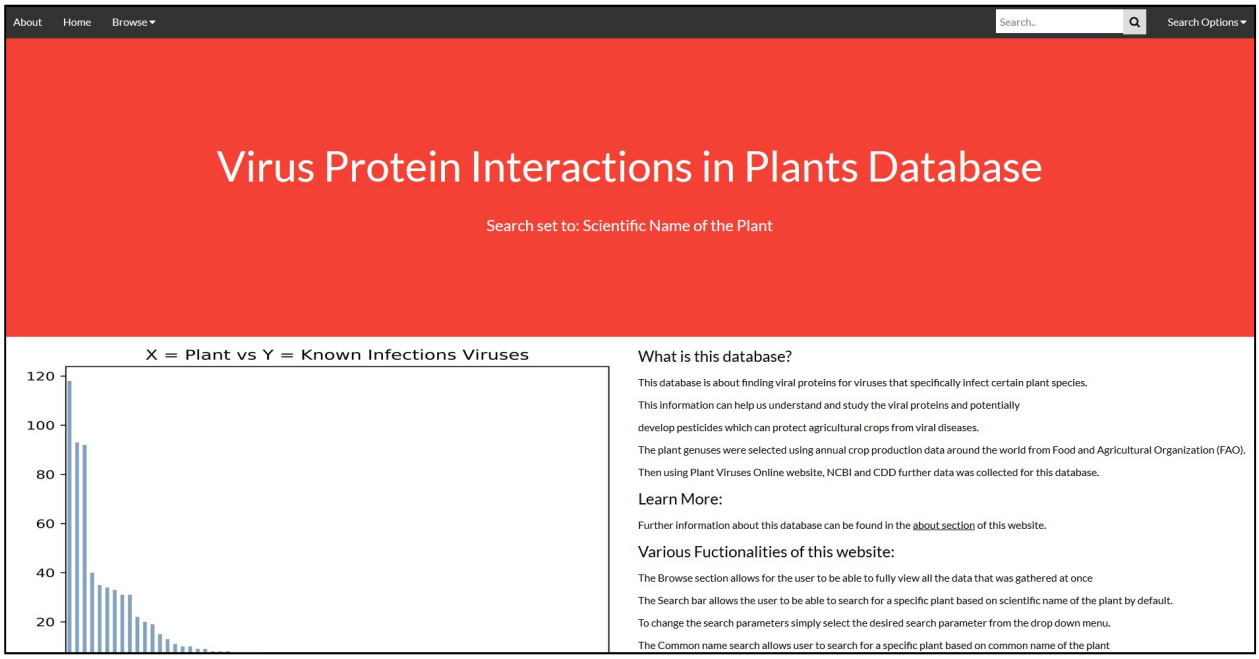


Figure 25: Shows the home page of the application. The graph showcases all the recorded infections for a particular plant species. The home page of the application goes in more detail about the database and about the graph itself.

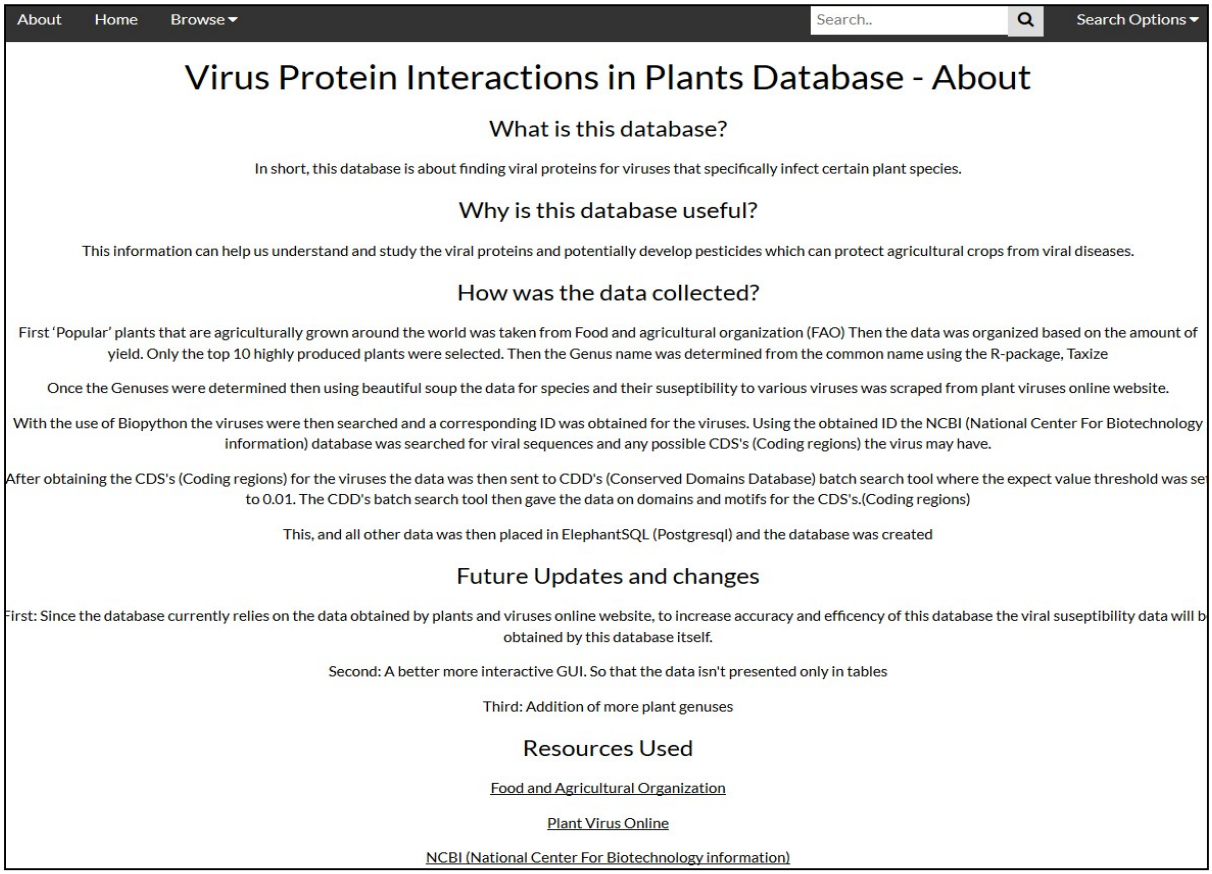
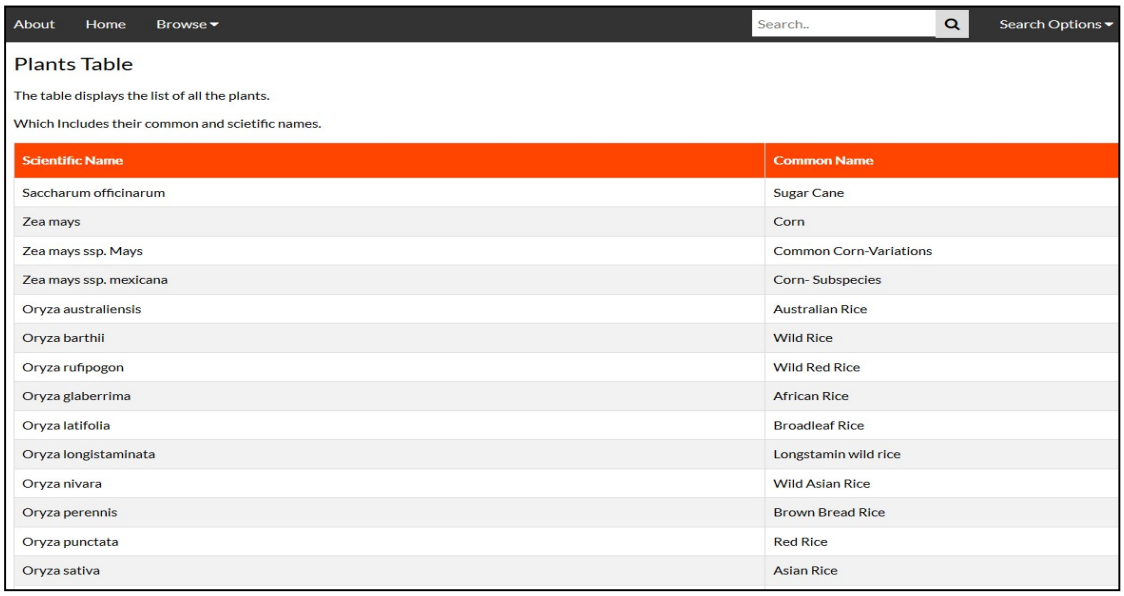


Figure 26: Shows the about page of the application. The about page summarizes how the data was obtained and what recourses were used for the creation of the application.

The Browse section

Contains 5 different tables that showcase all the data from the database.



Scientific Name	Common Name
Saccharum officinarum	Sugar Cane
Zea mays	Corn
Zea mays ssp. Mays	Common Corn-Variations
Zea mays ssp. mexicana	Corn- Subspecies
Oryza australiensis	Australian Rice
Oryza barthii	Wild Rice
Oryza rufipogon	Wild Red Rice
Oryza glaberrima	African Rice
Oryza latifolia	Broadleaf Rice
Oryza longistaminata	Longstamin wild rice
Oryza nivara	Wild Asian Rice
Oryza perennis	Brown Bread Rice
Oryza punctata	Red Rice
Oryza sativa	Asian Rice

Figure 27: Shows the first table within the browse section.

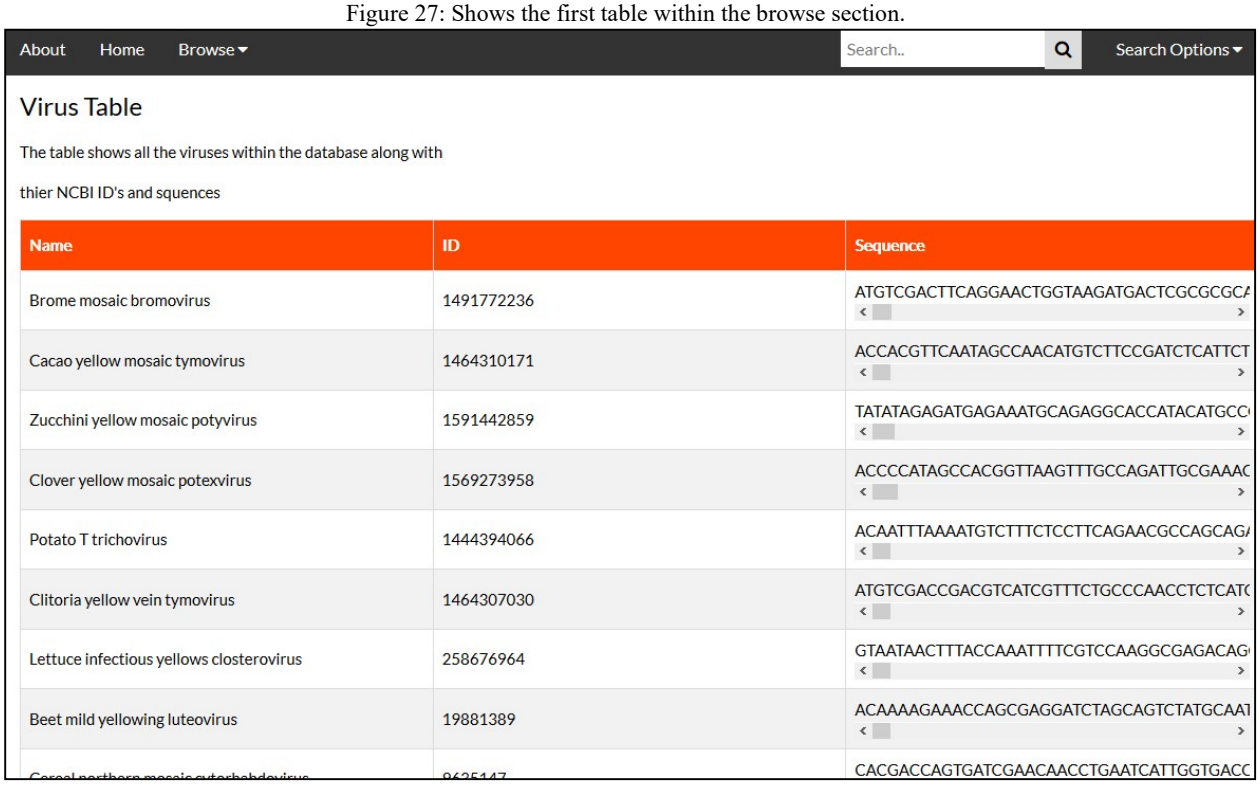


Figure 28: Shows the second table within the browse section. The scroll under the sequences allows the user to view the whole sequence without the need to scroll horizontally on the page itself.

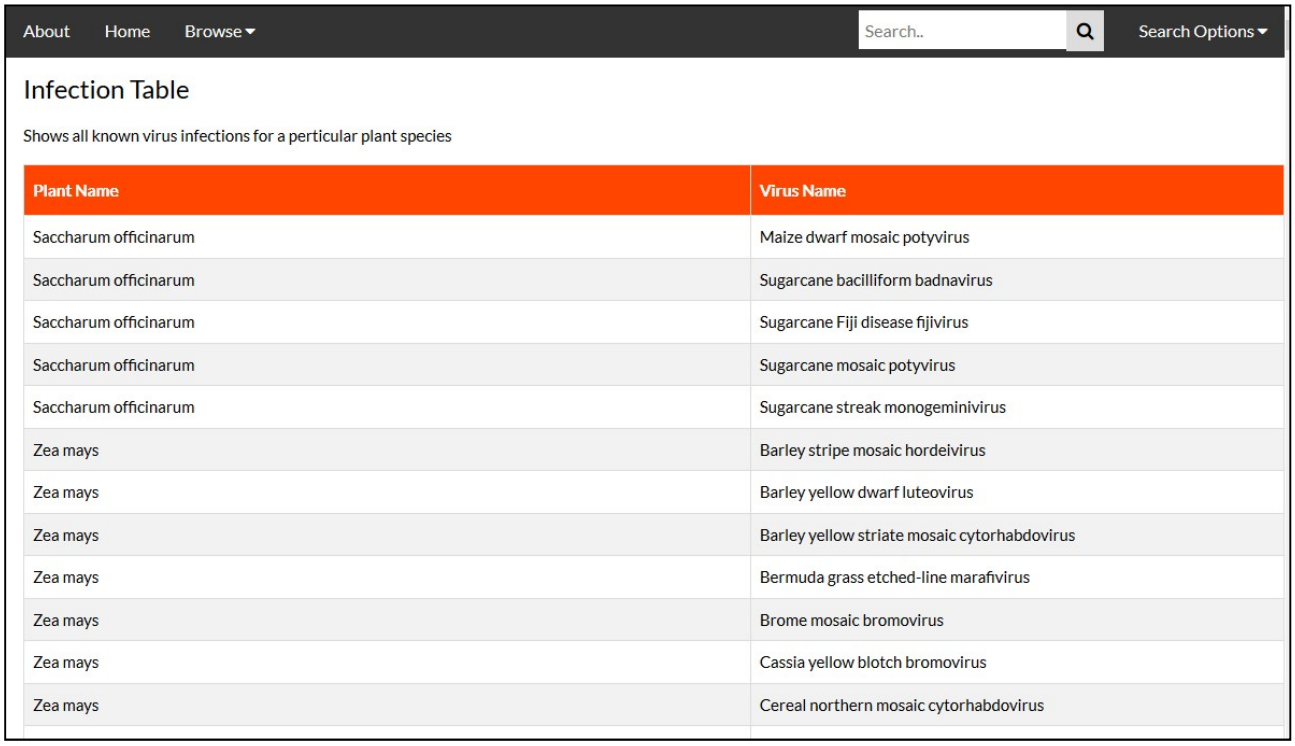


Figure 29: Shows the third table within the browse section.

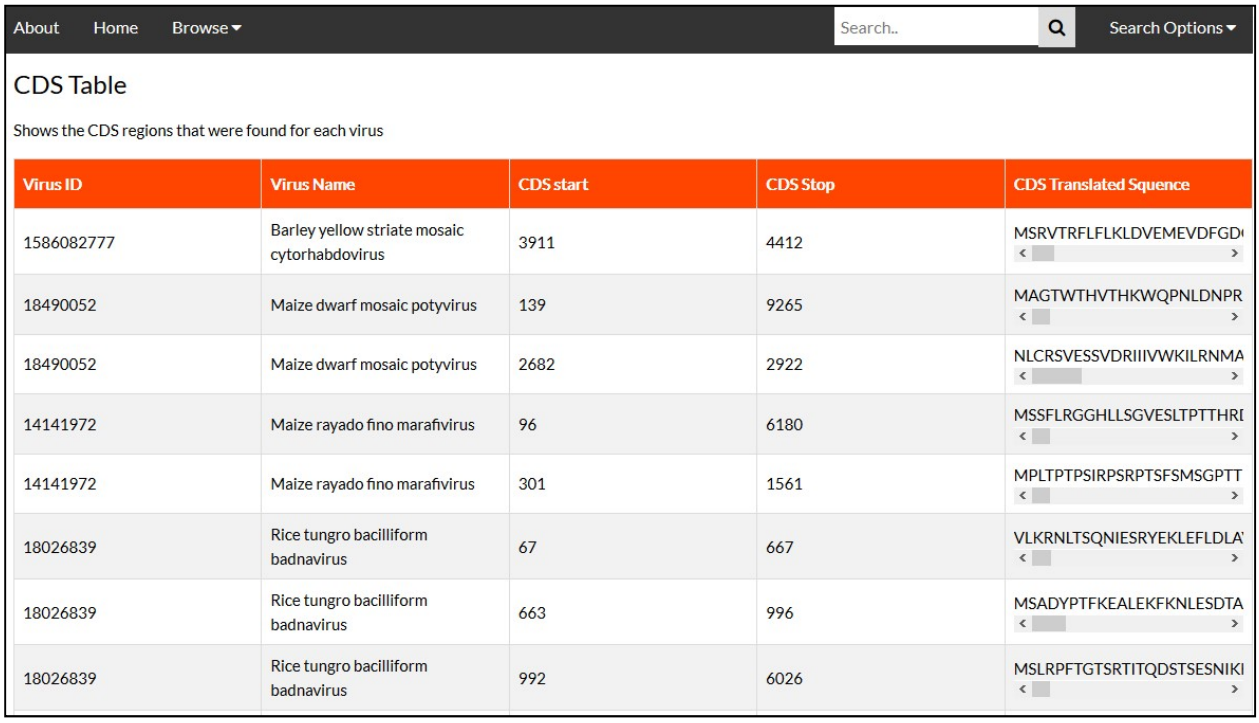


Figure 30: Shows the forth table within the browse section. The scroll under the sequences allows the user to view the whole sequence without the need to scroll horizontally on the page itself.

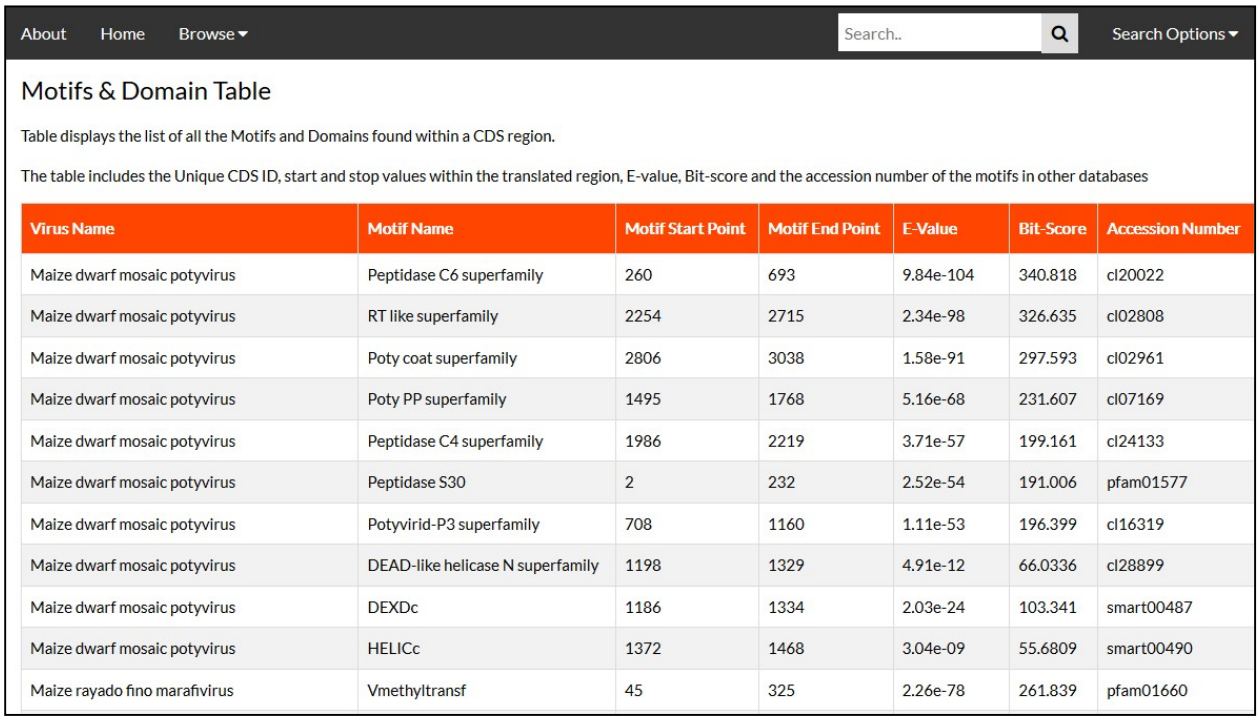
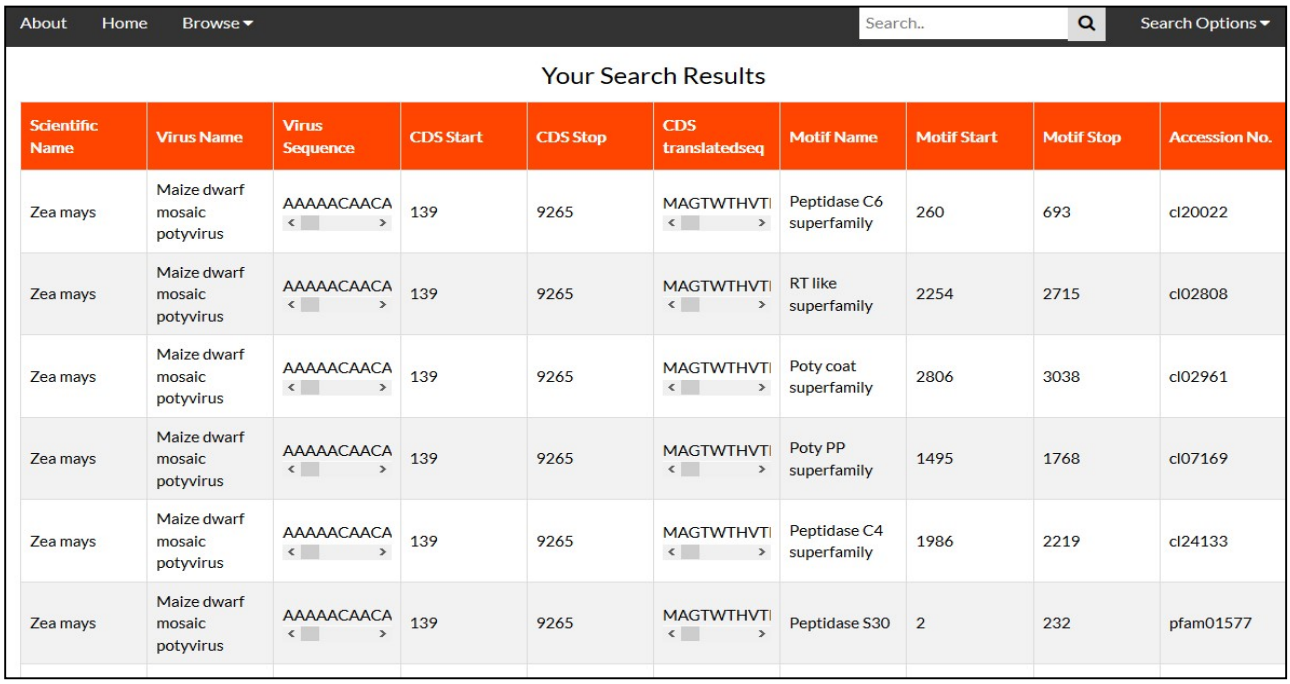


Figure 31: Shows the fifth table within the browse section. The accession numbers for the motifs can be used in the CDD website to search for more information on the motifs that were found.

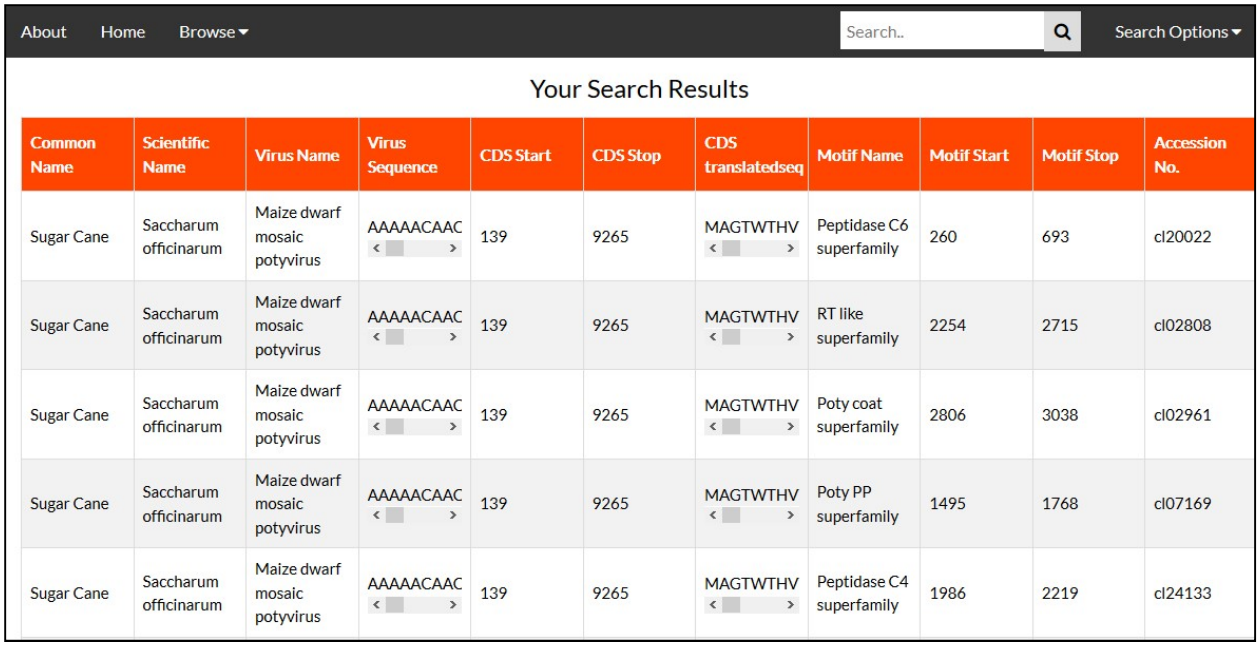
Search Section

The search section allows the user to search by the scientific name by default. However, it also allows the user to search by common name, virus and motif as well. The search option for scientific and common name of the plant is for the user to see the search results for a specified plant rather than needing to browse through the browse section of the application. The Virus search allows the user to search for a specific virus and view the virus sequence, cds and motifs found for the cds that a virus might posses. The motif search results allows user to search for various viruses that may possess a specific motif; this search also provides the accession number for the motifs.



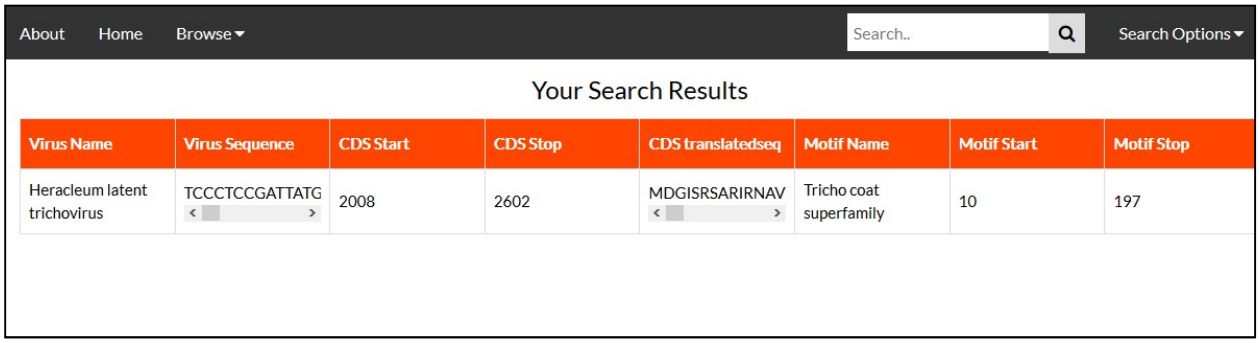
Scientific Name	Virus Name	Virus Sequence	CDS Start	CDS Stop	CDS translatedseq	Motif Name	Motif Start	Motif Stop	Accession No.
Zea mays	Maize dwarf mosaic potyvirus	AAAAACAACA	139	9265	MAGTWTHVTI	Peptidase C6 superfamily	260	693	cl20022
Zea mays	Maize dwarf mosaic potyvirus	AAAAACAACA	139	9265	MAGTWTHVTI	RT like superfamily	2254	2715	cl02808
Zea mays	Maize dwarf mosaic potyvirus	AAAAACAACA	139	9265	MAGTWTHVTI	Poty coat superfamily	2806	3038	cl02961
Zea mays	Maize dwarf mosaic potyvirus	AAAAACAACA	139	9265	MAGTWTHVTI	Poty PP superfamily	1495	1768	cl07169
Zea mays	Maize dwarf mosaic potyvirus	AAAAACAACA	139	9265	MAGTWTHVTI	Peptidase C4 superfamily	1986	2219	cl24133
Zea mays	Maize dwarf mosaic potyvirus	AAAAACAACA	139	9265	MAGTWTHVTI	Peptidase S30	2	232	pfam01577

Figure 32: Shows an example of the default search results where “Zea Mays” plant was searched.



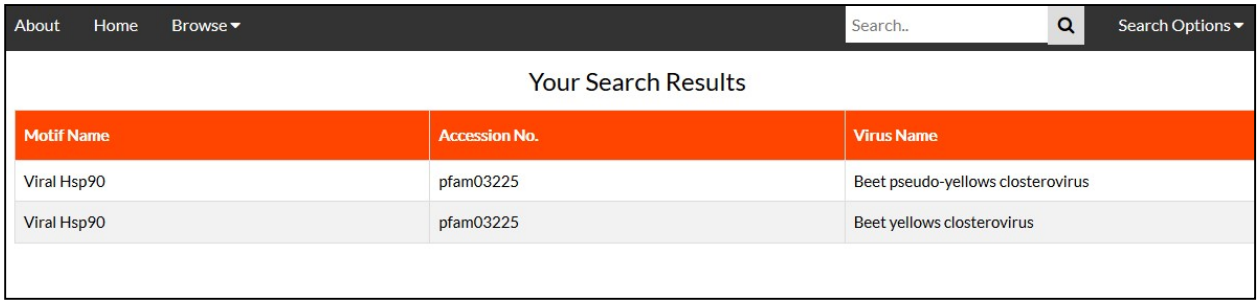
Common Name	Scientific Name	Virus Name	Virus Sequence	CDS Start	CDS Stop	CDS translatedseq	Motif Name	Motif Start	Motif Stop	Accession No.
Sugar Cane	Saccharum officinarum	Maize dwarf mosaic potyvirus	AAAAACAAC	139	9265	MAGTWTHV	Peptidase C6 superfamily	260	693	cl20022
Sugar Cane	Saccharum officinarum	Maize dwarf mosaic potyvirus	AAAAACAAC	139	9265	MAGTWTHV	RT like superfamily	2254	2715	cl02808
Sugar Cane	Saccharum officinarum	Maize dwarf mosaic potyvirus	AAAAACAAC	139	9265	MAGTWTHV	Poty coat superfamily	2806	3038	cl02961
Sugar Cane	Saccharum officinarum	Maize dwarf mosaic potyvirus	AAAAACAAC	139	9265	MAGTWTHV	Poty PP superfamily	1495	1768	cl07169
Sugar Cane	Saccharum officinarum	Maize dwarf mosaic potyvirus	AAAAACAAC	139	9265	MAGTWTHV	Peptidase C4 superfamily	1986	2219	cl24133

Figure 33: Shows an example where “Sugar Cane” was searched by setting the search settings to “common name” by using the dropdown menu next to the search bar.



Virus Name	Virus Sequence	CDS Start	CDS Stop	CDS translatedseq	Motif Name	Motif Start	Motif Stop
Heracleum latent trichovirus	TCCCTCCGATTATG	2008	2602	MDGISRSARIRNAV	Tricho coat superfamily	10	197

Figure 34: Shows an example where “Heracleum latent trichovirus” was searched by setting the search settings to “viruses” by using the dropdown menu next to the search bar.



Motif Name	Accession No.	Virus Name
Viral Hsp90	pfam03225	Beet pseudo-yellows closterovirus
Viral Hsp90	pfam03225	Beet yellows closterovirus

Figure 35: Shows an example where “Viral Hsp90” Motif was searched by setting the search settings to “Motif” by using the dropdown menu next to the search bar.

Discussion & Conclusion

Overall the application was connected and the data was retrieved from the database properly. The overall purpose of the application was fulfilled; however there is a lot of room for improvements. Future updates for the application would involve creating a proper GUI for viewing the sequences and results. The results would not contain redundancy. The reason for redundancy in the current version of the application is simply due to the join statements on the various tables, which are displayed without any edits made to the joined tables. The search bar needs to made more user friendly. Currently to change the search parameter one must use the drop down menu. The drop down menu simply directs the user to a version of the home page where the search bar itself has been changed based on what the user wishes to search for. Another update that needs to be made for the search bar is to make the searches not case sensitive. Currently the searches are very case sensitive and require the user to enter the exact search term. For example, in Figure 33: if the user had searched for “sugar cane” instead of “Sugar Cane” no results would be shown to the user. Additionally, more plant genuses need to be added to the database. The reason why only 10 genuses were used in the current version of the database is because Elephant SQL has a limit of 128 MB of free usage. Once the data exceeds 128 MB the user is required to purchase more space for the database. Finally the application currently requires data from Plant and viruses online website; the website itself isn’t update all too much. Hence, to remove reliance on Plant and viruses online website, an algorithm that generates viral infection data based on plant species is required, within the application. Creating this algorithm would remove reliance on Plant and viruses online website. Despite these future improvements the database is able to record viral proteins along with the type of host the virus usually infects. This could prove beneficial in understanding what the viruses are capable of and thereby giving us a step ahead against countering viruses and protecting agricultural crops.

References

1. (Lecoq) Lecoq, H. “Discovery of the First Virus, the Tobacco Mosaic Virus: 1892 or 1898?” *Comptes Rendus De L'Academie Des Sciences. Serie III, Sciences De La Vie*, U.S. National Library of Medicine, Oct. 2001, www.ncbi.nlm.nih.gov/pubmed/11570281.
2. (Smith) Smith ,H. “Mechanisms of Viral Pathogenicity” *Department of microbiology, University of Birmingham, Birmingham, England*, Vol.36,NO.3 Bacteriological Reviews. American Society of Microbiology, Sept. 1972, https://mmbr.asm.org/content/mmbr/36/3/291.full.pdf
3. (Kamps) Kamps, Bernd Sebastian, et al. *Influenza Book | Pathogenesis and Immunology*, www.influenzareport.com/ir/pathogen.htm.
4. (Lodish) Lodish H, Berk A, Zipursky SL, et al. Molecular Cell Biology. 4th edition. New York: W. H. Freeman; 2000. Section 6.3, Viruses: Structure, Function, and Uses. Available from:
5. (Rui) Rui, Liu Sheng, et al. “MicroRNA-Mediated Gene Silencing in Plant Defense and Viral Counter-Defense.” *Frontiers in Microbiology* , vol. 08, no. 1664-302x, 2017, p. 1801., doi:10.3389/fmicb.2017.01801.
6. (Pratt) Pratt, Ashley J and Ian J MacRae. “The RNA-induced silencing complex: a versatile gene-silencing machine” *Journal of biological chemistry* vol. 284,27 (2009): 17897-901.
7. (Kamthan) Kamthan, Ayushi et al. “Small RNAs in plants: recent development and application for crop improvement” *Frontiers in plant science* vol. 6 208. 2 Apr. 2015, doi:10.3389/fpls.2015.00208