Clustering Genes according to their Plant and Small-Molecule Drug Associations through the use of Self-Organizing Maps

Abigael Cia, Ma. Angelika Ferrer, Wesley Kayanan, Graciel Ogena, Regina Santos
Department of Physical Sciences and Mathematics
College of Arts and Sciences
University of the Philippines - Manila

***Abstract*—The extraction of small-molecule drugs for pharmaceutical development was the practice for different cultures for many centuries. And recently, researchers are regaining an interest in the search for therapeutic agents from natural sources such as plants to be used as an alternative for artificial compounds. The information gathered from the analysis of the relationship between genes, plants and small-molecule drugs (SMD) can be used for drug development using natural resources. This activity is a recreation of the paper of Recario et al. [1] on the gene association and conditions of plants and their phytochemicals with respect to their SMD. It is important to take note that in this paper, the primary focus is on the gene associations of plants with respect to their SMD. The graphs that will be generated in this project would be the (1) gene-smd and the (2) gene-plants graphs. Gene, plant, and SMD data were web scraped from *DrugBank Online* and *Dr. Duke's Phytochemical and Ethnobotanical Databases*. After cleaning and preprocessing, the data is then used to generate a graph based on their corresponding adjacency matrices; representing the relationship between the genes and SMD, as well as the genes and the plants. Degree centrality and weighted degree centrality was also computed to find the gene with the most association. For the final part of this research, *self-organizing maps (SOM)* is a type of artificial neural network that uses unsupervised learning to cluster graphs without predetermined data. This was then used for the gene-smd and gene-plants graphs to obtain the clusters within among the two.**

***Keywords*—genes, small-molecule drug (SMD), plants, web scraping, self-organizing maps**

I. Introduction

Over the past few years, the pharmaceutical industry gives its main focus solely on artificial compounds through manufacturing drugs as it utilizes easy production. However, a decreasing trend in the market has occurred in the latest drugs being released. Because of this, researchers gained a huge interest in discovering drugs from natural resources such as plants [2].

Herbal medicines are considered as a preferred alternative in treating diseases in primary healthcare [3, 4]. They usually consist of phytochemicals, components of plants that may provide good health benefits [5, 6]. However, it is important to take note that not all phytochemicals can have pharmacological activities on the human body. To demarcate between as to which phytochemicals that are bioactive compounds, the term small-molecule drug (SMD) has been coined. SMDs are chemical compounds that are developed to easily access their targets in the human body such as genes and they have a molecular weight that is less than 1000 daltons [7, 8, 9]. There are still adverse health effects present when using herbal medicine in spite of its health benefits. Therefore, it is necessary to discover the relationship between the components of herbal medicine and the target of the herbal medicine in the human body.

To be able to study the association between genes, SMDs, and plants, graph theory can be used to visualize the relationships between relevant data since it allows modeling and analyzing the structure of a network [10]. There are instances that a plant node can have a heavier weighted edge in association to a particular gene if it has a higher occurrence with respect to the SMD. This example represents the concept of degree centrality and weighted degree centrality. An example in our gene-smd graph, a gene that has a higher weighted degree centrality implies that the gene has a high association among the SMD that is mapped to it. On the other hand, degree centrality pertains to a gene's association to unique SMD nodes that are found in the graph. The higher the degree centrality means that it has a diverse set of SMDs that are mapped to it [11, 12].

For graph clustering, a group of nodes that are more likely to be connected to each other tend to form its own cluster. In this research, *self-organizing maps (SOM)* will be used as a technique to cluster gene-smd nodes and gene-plant nodes that are highly related to each other [13]. Recario et al. [1] created a web-based information management system that also uses graphs in providing visual representations of the relationships among plants, small-molecule drugs, and human genes.

This paper aims to determine the genes that are central, as well as the genes that tend to cluster with respect to SMDs and plants by constructing graphic visualizations of the gene-SMD and gene-plant associations through SOM.

II. Methodology

The data used in this paper were collected from two publicly available databases. The data for the small-molecule drugs (SMD) were taken from *DrugBank* and the plant data were collected from *Dr. Duke's Phytochemical and Ethnobotanical Databases* both use a web scraping library called Beautiful Soup. $BeautifulSoup$ is a Python library for pulling data out of HTML and XML files. After collecting the data, it is then filtered to get the information needed and discard the unnecessary variables from the data tables.

After filtering, two tables would then be generated, one table with gene as row and SMD as the column and the another table with gene as the row and plant as the column. The tables would then be used to generate a graph using *NetworkX*. *NetworkX* is a Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks. Using the generated graphs the degree centrality and weighted degree centrality for each node on the graph were calculated.

Data collection, data processing, and graph generation was done in *Google Colab*. *Google Colab* provides a Python notebook environment which allows multiple users to code and edit the program together real-time. It also offers cloud storage by connecting Google Drive and the project, saving memory space from the individual members' devices. To graph the associations of interest, we applied the concept of bipartite graphs. Bipartite graphs are graphs whose nodes can be divided into two disjoint sets and edges connect a node from one disjoint set to a node from the other disjoint set. For this study, we have to extract the set of genes and the set of SMDs for gene-SMD association as well as the set of genes and the set of plants for gene-plant association.

A. Data Collection

Data were collected and web scraped from two publicly available databases: *Drugbank Online* and *Dr. Duke's Phytochemical and Ethnobotanical Databases*, respectively; using *Python's Beautiful Soup library*.

DrugBank provides an online database of detailed descriptions about drugs and its targets. From there, we were able to extract generic names, synonyms, and genes of SMDs. A total of 2,695 rows of data were scraped and saved in a comma separated (CSV) file. On the other hand, Dr. Duke's Phytochemical and Ethnobotanical Databases offers chemical and ethnobotanical data of various plants. Scientific names, common names, and phytochemicals of plants were extracted from this database and 2,376 rows of data were gathered and also saved in a separate CSV file.

Since we are only interested in gene-SMD and gene-plant association, the group determined which data would make the mapping possible. For gene-SMD, the relationship can be accessed easily through the generated CSV files. As long as a target gene for an SMD exists, then there is an association between the said entities. On the other hand, there is no straightforward way to demonstrate the relationship between the genes and the plants, but we have the knowledge that certain phytochemicals that are present in plants are also SMDs, and these SMDs may or may not have target genes. Using this information, the data of DrugBank and Dr. Dukes data may be filtered to map the relationship of plants to genes. Filtering conditions to obtain data for gene-plant association are as follows: (i) plants with phytochemicals should only contain SMDs (ii) phytochemicals should have at least one SMD and (iii) target gene of the identified SMD in plant exists. Out of 2,695 rows of SMD-gene data, we were able to extract 120 genes that can be mapped to 1072 plants from previously 1228 rows of data. Both associations would be based on this filtered data.

B. Procedure

As mentioned, data was gathered through web-scraping. For the SMD collection, we created the $GetDrugBankData()$ and $GetDrugDetails(url)$ functions. The $GetDrugDetails(url)$ is responsible for getting the common names, synonyms, and genes from the url argument passed to it, while the $GetDrugBankData()$ iterates over all the SMDs listed in the DrugBank and collects each url which is then passed to the $GetDrugDetails(url)$ to gather the necessary information for the study. If a certain SMD does not have a target gene nor synonym, then we set the value to be "Not Available". These were stored to the $drug - bank - data.csv$. Simultaneously, we were able to fetch the plant data from Dr. Duke's using the $GetPlantDetails(plantNo)$ and the $GetDrDukesData()$ function. The $GetDrDukesData()$ visits each page of the plants in Dr. Duke's Phytochemical and Ethnobotanical Databases based on their assigned plant number. It calls the $GetPlantDetails(plantNo)$ to scrape common and scientific names of the plants from their individual pages and to dive deeper into the subpages where the phytochemicals are stated. Likewise, if a plant does not have a scientific name nor any phytochemical, then the field will be labelled "Not Available". Data is then stored to $dr-dukes-data.csv$.

From the previous step, we have collected the common names, synonyms, and target genes of SMDs as well as the scientific names, common names, and phytochemicals of plants which were stored in the $genesSMDData$ and the $plantsSMDData$ variable, respectively; then we proceed with the filtering. First, (1) we iterate over the phytochemicals in the $plantsSMDData$ to check whether they are SMDs by comparing them to the generic names and synonyms in the $genesSMDData$. Using the $RemoveNonSMDPhytochemicals(dbData, ddData)$, we were able to identify which phytochemicals are SMDs and which are not. If the phytochemical matches an SMD

the $genesSMDData$, then we keep the phytochemical in the $plantsSMDData$ under the column "SMD". Otherwise, we label the SMD as "Not Available". From the modified data, (2) we removed plants which have non-SMD phytochemicals together with the rows whose phytochemicals were originally "Not Available" using $RemoveRowWithoutValue(data, columnName, value)$. (3) The group also used the said function to eliminate rows of SMDs which do not contain genes because it is impractical to keep data that do not contain the information that we need; in this case, the genes. Next, (4) we need to verify if the remaining SMDs in the $genesSMDData$ are present in our $plantsSMDData$. Not all SMDs with genes are phytochemicals. Thus, there is a need to remove them as they are not useful in finding associations between genes and plants. We are able to eradicate these non-phytochemical SMDs using the *RemoveNonPlantSMD(genesSMDData, plantsSMDData)* function. Conversely, (5) we also need to validate if SMDs in the $plantsSMDData$ are present in the $genesSMDData$. Some phytochemicals may have been considered an SMD in the first step (1) of filtering, but we must take note that we removed SMDs due to the absence of genes (3). With that being said, those SMDs were labelled as "Not Available" in the $plantsSMDData$ by the *FilterPlantsOnUpdatedSMD(genesSMDData, plantsSMDData* function. Finally, (6) we remove non-phytochemical SMDs using the function *RemoveRowWithoutValue(plantsSMDData, "SMDs", "Not Available")*. The last three functions synchronize the $genesSMDData$ and the $plantsSMDData$ after the first three pre-processing.

Using the filtered data, we created two dataframes: one for gene-SMD and another for genes-plants. With the help of the $CreateGeneSMDTable(genesSMDData)$ function, we were able to construct the $geneSMDTable$ by extracting the union of all genes and generic names of all SMDs and setting them as the table rows and columns, respectively. If a specific gene is a target gene of the SMD, then we set their intersection to 1; else, 0. On the other hand, we constructed the $genePlantTable$ by getting the intersection of the SMD in the $genesSMDData$ and $plantsSMDData$ using the *CreateGenePlantTable(genesSMDData, plantsSMDData)* function. For every gene that has the same or synonymous SMD with a plant species, we set their intersection in the table into 1, else 0. These data frames were also subjected to the $GetDegreeCentrality(G, nodes)$ and $GetWeightedDegreeCentrality(G, nodes)$ functions for the computation of degree centrality and weighted degree centrality of each gene node.

For graphing, we used the $CreateGraphFromMatrix(matrix)$ function to enumerate the edges and nodes for the gene-SMD network and gene-plant network, then plotted the graph using the $DrawGraph(graph)$ function.

For the self-organizing maps, $Minisom$ was used. $MiniSom$ is a minimalistic and $Numpy$ based SOM. The $genePlantTable$ and $genesSMDTable$ are saved as a $.csv$ file for ease, and converted into a $NumPy$ array. The features are then isolated, and scaled using $MiniSom$. The model created will be trained and a cluster plot of the SOM is generated.

III. Results

As the group has produced the two graphs with respect to the two adjacency matrixes, the gene-smd and gene-plant graphs are shown in Figures 1 and 2 respectively.

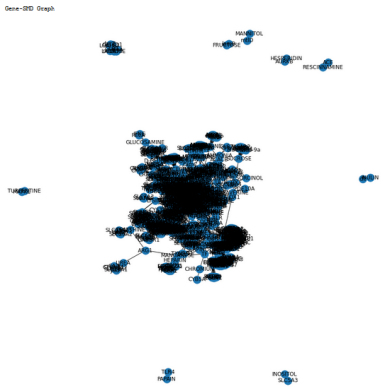

Figure 1: Gene-SMD Graph

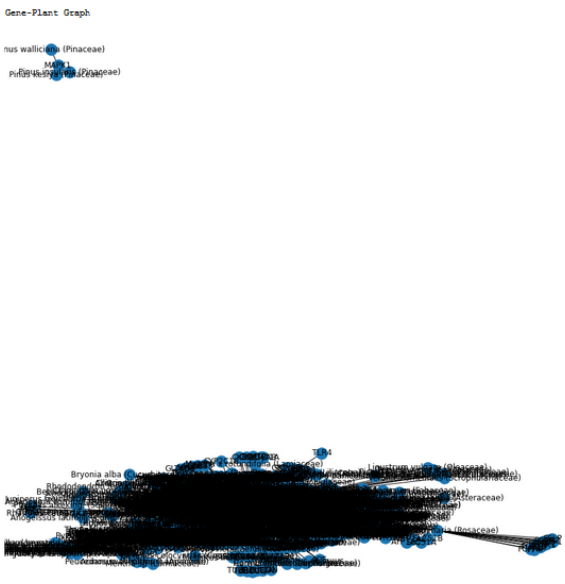

Figure 2: Gene-SMD Graph

Gene-SMD graph is composed of 846 genes and 120 SMDs. Table I shows the ten genes with the highest degree centrality; that is, genes with the most distinct SMD relations. CYP3A4, the gene with 31 SMD association, is an enzyme found in the liver and the intestine, which is responsible for getting

rid of toxins and drugs in the body. Gene-plant graph, on the other hand, is composed of 846 genes and 1064 plants. Table II shows the ten genes with the highest degree centrality, or genes with the most distinct plant relations. ALB or Albumin, the gene with 685 plant association, is a protein-coding gene for endogenous molecules like hormones, fatty acids, and metabolites.

Gene	Degree Centrality
CYP3A4	31
ALB	26
ABCB1	21
CYP2C9	17
CYP2E1	16
SLC22A2	15
CYP2D6	15
CYP1A1	15
CYP1A2	14
SLC22A1	14

Table I: Ten highest degree centrality in the Gene-SMD association network

Gene	Degree Centrality
ALB	685
CYP3A4	618
TF	610
CP	605
HBA1	604
NEIL1	603
NEIL2	603
HDAC8	600
TFRC	594
FXN	594

Table II: Ten highest degree centrality in the Gene-Plant association network

For Tables III and IV, we present ten genes with the highest weighted degree centrality from the gene-SMD graph and the gene-plant graph, respectively. CYP3A4 has the most edges to SMDs with a total of 37 edges while ALB has the most edges to plants with a total of 689 edges.

Gene	Weighted Degree Centrality
CYP3A4	37
ALB	26
ABCB1	21
CYP2C9	17
CYP2E1	16
SLC22A2	15
CYP2D6	15
CYP1A1	15
CYP1A2	14
SLC22A1	14

Table III: Ten highest weighted degree centrality in the Gene-SMD association network

Since the paper that the group has used as reference made a functionality to show the SMD that are associated with one gene, we also created the visualization wherein the genes that are associated to the SMDs in Figure 3 and the genes for plants in Figure 4 both resemble star graphs.

Gene	Weighted Degree Centrality
ALB	689
CYP3A4	622
TF	614
CP	609
HBA1	608
NEIL1	607
NEIL2	607
HDAC8	604
TFRC	598
FXN	598

Table IV: Ten highest weighted degree centrality in the Gene-Plant association network

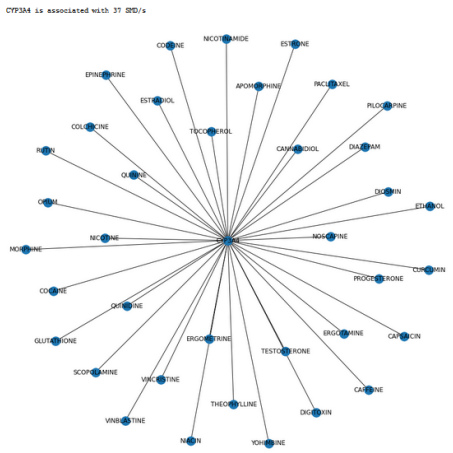

Figure 3: SMDs associated with CYP3A4

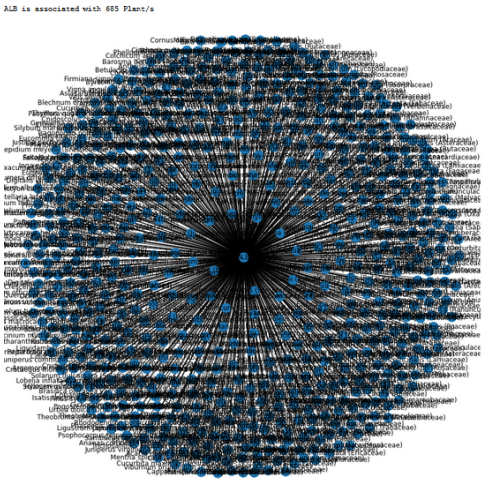

Figure 4: Plants associated with ALB

For the SOM, training took 3 minutes and 46 seconds to finish and resulted in a quantization error of 0.41 for the SMD-Gene SOM. For the SMD-Plant SOM, training took 19 minutes and 38 seconds and resulted in a quantization error of 0.77. The resulting clusters are shown in Figures 5 and 6.

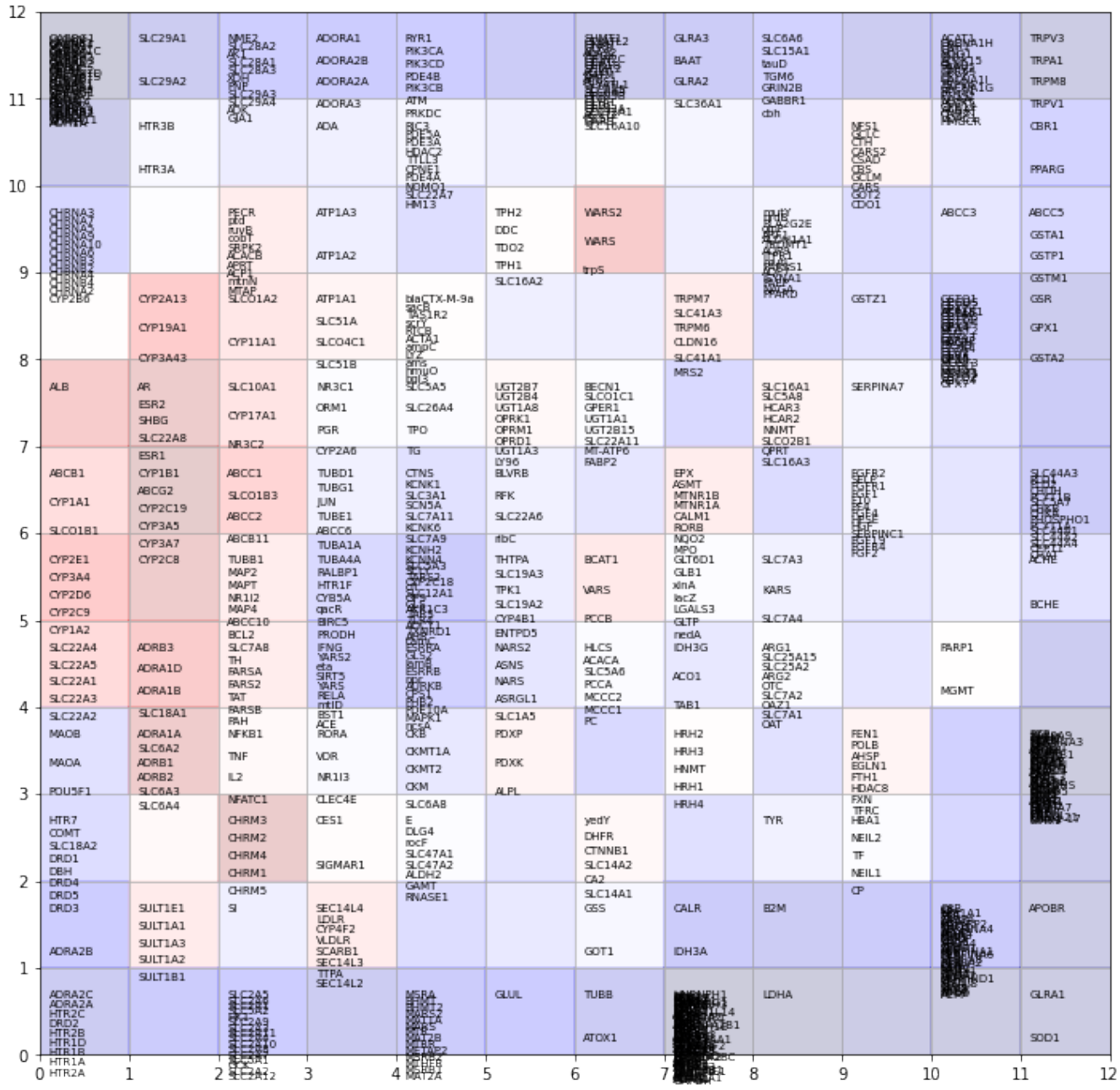

Figure 5: Gene - SMD SOM

IV. Discussion

Graphs are constructed to visualize the associations of gene-SMD and gene-plant. It is also used to determine the degree centrality and weighted degree centrality of both associations. Meanwhile, SOMs are used to determine which genes tend to group together to create a cluster.

In the gene-SMD graph, the CYP3A4 gene has the highest degree centrality with 31 edges. It also has the highest weighted degree centrality with 37 edges. One possible reason for this outcome is the fact that CYP3A4 is mainly used for the metabolism of various drugs in the market [14, 15]. In addition, there are a few smaller clusters that are disconnected from the bigger cluster. This shows that some genes can only be affected by specific SMDs only. The SMDs in those clusters do not interact with the genes within the bigger cluster and vice versa.

For the gene-plant graph, the ALB gene has the highest degree centrality and weighted degree centrality with a value of 685 and 689, respectively. This result may be supported by the fact that ALB or albumin has been of interest in the

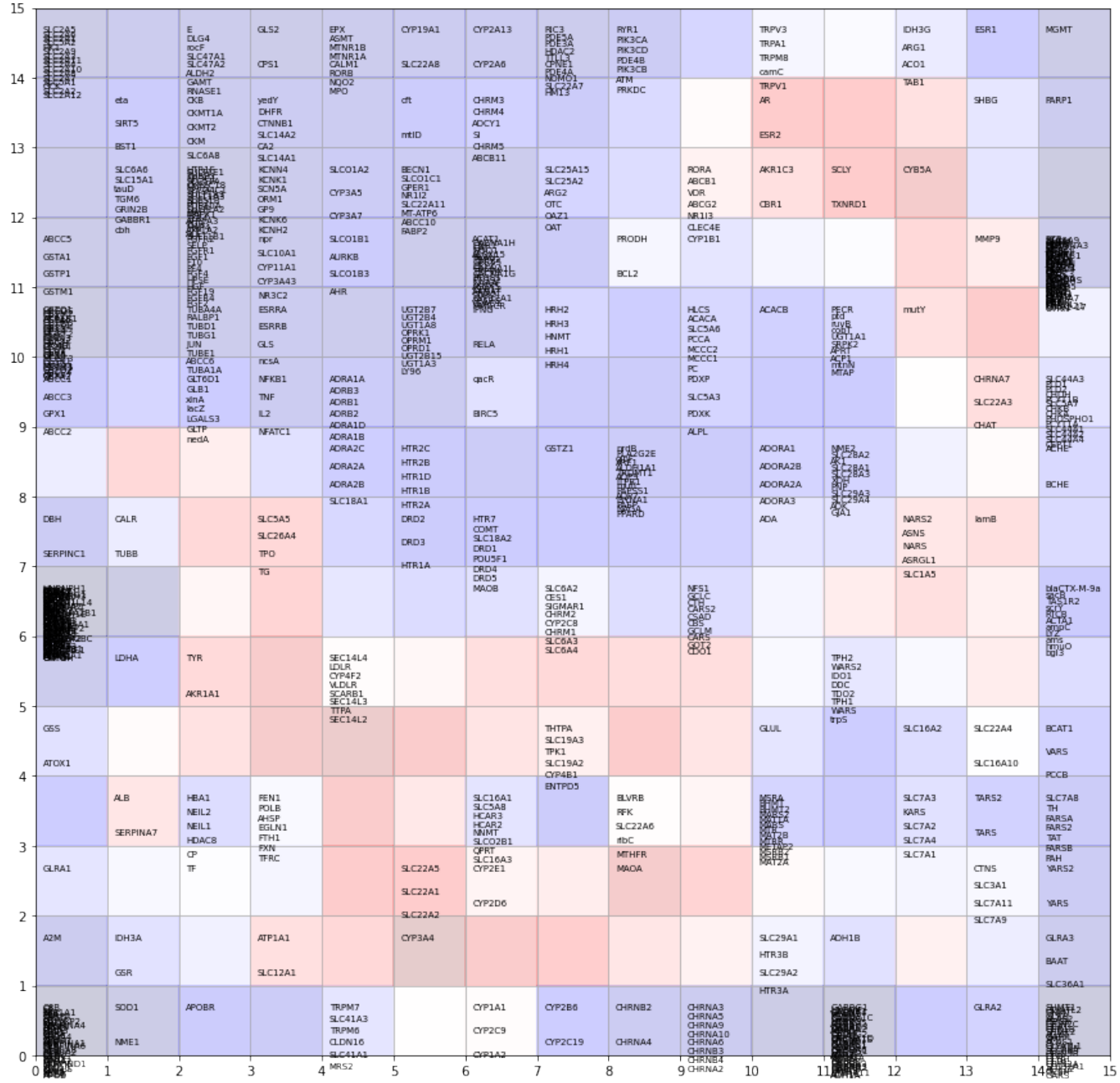

Figure 6: Gene-Plant SOM

pharmaceutical industry because of its ability to aid in drug transportation and improve pharmacokinetics, targetability, solubility, and instability; thus making it an initial target of drugs to improve clinical conditions like shock, burns, trauma, and blood loss [16, 17]. In this graph, there is a single small cluster that is disconnected from the bigger cluster. It has a similar concept to the gene-SMD graph, wherein the plants in the small cluster do not interact with the genes in the bigger cluster since it indicates that some genes can be affected by particular plants only.

For the SOM, genes are clustered based on unsupervised learning. Based on the results, some clusters are crowded while there are also clusters that have few genes. Cells of the same color belong to the same cluster. In the SOM graph, we have the blue and pink clusters. Areas that are white in color are considered outliers or points that do not belong to any clusters. Also, more genes tend to cluster within the blue regions than the pink ones.

Additionally, clustering helps in narrowing down choices of ingredients to come up with a drug that has a certain

target; like in this study, clustering shown on the graphs may give an idea on what SMDs or plants should be analyzed if we were to synthetically or herbally produce a drug whose target gene is either CYP3A4 or ALB.

V. Conclusion

There are plenty of benefits in determining the essential phytochemicals within plants. This is because some of these phytochemicals can induce pharmacological activities within the human body. These compounds that are known to be bioactive compounds are referred to as small-molecule drugs (SMD) that have a molecular size that is less than 1000 daltons. By demarcating the non-SMDs from the SMDs, this would be the bridge for us to know the drugs that may have an impact on the human body. This would help us determine the SMDs that would have an association with the human genes. In this paper, since we were able to cluster the genes according to their SMD and plant associations, it would help us determine the group of genes that can be affected by one particular drug or plant.

Graphs are constructed to visualize the associations of gene-SMD and gene-plant. It is also used to determine the degree centrality and weighted degree centrality of both associations. It was found that the CYP3A4 gene has the most association with SMDs. On the other hand, the ALB gene has the most association with plants. These genes are often targeted by drugs because they assist in more effective drug transportation and absorption. Meanwhile, SOMs are used to determine which genes tend to group together to create a cluster. Both gene-SMD and gene-plant created two clusters with a few independent clusters in between.

VI. Recommendation

Certain parts of this study may be further improved by future researchers. (1) Data filtering may or may not have caused any discrepancies to the data. Functions were made based on our understanding on how to extract data that is needed to observe the associations between the genes and SMDs, and the genes and plants. There may have been errors, like tagging an entity as "Not Available" even if it really exists and typographical errors due to string manipulation or web-scraping; that have been overlooked due to time constraints. (2) Web-scraping, especially in the Dr. Duke's Phytochemical and Ethnobotanical Databases, takes approximately 43 minutes to finish. There may be more efficient ways to go about this process that future researchers may apply. Scope of the study may also be extended to a broader group. For instance, (3) future researchers may opt to include phytochemicals that are not SMDs. Since we are interested in the gene-plant association, we removed non-phytochemical SMDs because they are not bioactive; in other words, they do not interact with genes and thus would not result in any relationships. In relation to this, others may investigate if SMDs can come from other sources, like say, animals, and observe whether they also have any relationship with genes and what are possible biomedical applications that can be drawn from it. Other recommendations may be towards the pharmaceutical community. The Cytochrome P450 3A4 (CYP3A4) and Albumin (ALB), the two genes that have the highest degree centrality and weighted degree centrality in their respective networks, are both associated with the liver organ of the human body. (4) Pharmaceutical companies may focus on developing and producing herbal drugs based on these genes for chronic liver disease since it is considered as one of the complications that causes high mortality rates worldwide.

VII. Source Code

```
!pip3 install matplotlib

import re
import requests
import pandas as pd
import ast

import matplotlib.pyplot as plt
import networkx as nx

from google.colab import drive
from bs4 import BeautifulSoup

# Authentication to Read and Upload Files
drive.mount("/content/drive")

# HI 193.1 (Group 6) - Final Paper [Drive]
    Folder
GDRIVE = '/content/drive/My Drive/HI 193.1 (
    Group 6) - Final Paper [Drive]/'

# Columns of the CSV files
DRUG_DETAILS = ["Generic Name", "Synonyms", "
    Genes"]
PLANT_DETAILS = ["Scientific Name", "Common
    Name", "Phytochemicals"]

# Link to CSV files
DRUGBANK_CSV = GDRIVE + 'drug-bank-data.csv'
DRDUKES_CSV = GDRIVE + 'dr-dukes-data.csv'

GENES_SMD_DATA_CSV = GDRIVE + 'genes-smd-data.
    csv'
PLANTS_SMD_DATA_CSV = GDRIVE + 'plants-smd-
    data.csv'
GENE_SMD_TABLE_CSV = GDRIVE + 'gene-smd-table.
    csv'
GENE_PLANT_TABLE_CSV = GDRIVE + 'gene-plant-
    table.csv'

# DrugBank Links for processing
DRUGBANK_URL = 'https://go.drugbank.com/'
DRUGBANK_DRUGS_URL = 'https://go.drugbank.com/
    drugs'

# Dr. Dukes Links for processing
DRDUKES_URL = 'https://phytochem.nal.usda.gov
    /'
DRDUKES_PLANTS_URL = 'https://phytochem.nal.
    usda.gov/phytochem/plants/show/'
```

```
DRDUKES_PHYTOCHEMICALS_URL = 'https://
    phytochem.nal.usda.gov/phytochem/plants/
    plantsFarmacyList/'

# Get Drug Bank Data
def GetDrugBankData():
  data = pd.DataFrame(columns=DRUG_DETAILS)
  url = DRUGBANK_DRUGS_URL

  while url:
    # Request HTML of Page
    r = requests.get(url)

    # Parse Request of Page
    soup = BeautifulSoup(r.text ,"lxml")

    # Obtain Anchor Element of Links
    links = soup.select('strong a')

    # Get Drug Details Per Drug
    for link in links:
      data = data.append(GetDrugDetails(
          DRUGBANK_URL + link['href']),
          ignore_index=True)

    # Check if next page exists
    url = soup.findAll('a', {'class': 'page-
        link', 'rel': 'next'})
    if url:
      url = DRUGBANK_URL + url[0].get('href')
    else:
      break

  return data

# Get Drug Details Based on URL
def GetDrugDetails(url):
  # Request HTML of Drug Details Page
  r = requests.get(url)

  # Parse Request of Drug Details Page
  soup = BeautifulSoup(r.text ,"lxml")

  # Get Drug Detail Labels
  dts = soup.findAll('dt')

  # Get Drug Detail Values
  dds = soup.findAll('dd')

  # Create a dictionary w/ labels mapping to
      its values
  curr = {}

  # Get Generic Name
  for dt, dd in zip(dts, dds):
    if dt.text == "Generic Name":
      curr[dt.text] = dd.text

  # Get Synonyms
  synonyms = []
  syn_dd = soup.find("dt", {"id": "synonyms"}).
      findNext('dd')
  if syn_dd.ul:
    for li in syn_dd.findAll('li'):
      synonyms.append(li.text)
  else:
    synonyms = "Not Available"
  curr["Synonyms"] = synonyms

  # Get Genes
  genes = []
  gene_dts = soup.findAll("dt", {"id": "gene-
      name"})
  if gene_dts:
    for gene_dt in gene_dts:
      gene_dd = gene_dt.findNext('dd')
      if gene_dd.text != "Not Available":
        genes.append(gene_dd.text)
    if not genes:
      genes = "Not Available"
  else:
    genes = "Not Available"

  curr["Genes"] = genes

  return curr

# Get Dr Dukes Data
def GetDrDukesData():
  data = pd.DataFrame(columns = PLANT_DETAILS)

  plantNo = 1
  url = DRDUKES_PLANTS_URL + str(plantNo)

  while url:
    # Request HTML of Page
    r = requests.get(url)

    # Parse Request of Page
    soup = BeautifulSoup(r.text, "lxml")

    # Page
    page = soup.find("h1", {"class": "
        entityHeader"})

    if page:
      sn = soup.find("h1", {"class": "
          entityHeader"})
      sn.a.decompose()
      sn.span.decompose()
      sn = FixWhiteSpace(sn.text)

      # Common Name
      cn = "Not Available"
      cnHeader = soup.find("h2", text="Common
          name(s)")
      if cnHeader:
        cnHeader = cnHeader.find_parent("li")
        cnHeader.h2.decompose()
        cn = FixWhiteSpace(cnHeader.text)

      # Get Phythochemicals of Current Plant
      phytochemicals = GetPlantDetails(plantNo)

      data = data.append({"Scientific Name": sn,
          "Common Name": cn, "Phytochemicals":
          phytochemicals}, ignore_index=True)

      plantNo += 1
      url = DRDUKES_PLANTS_URL + str(plantNo)
    else:
      break
```

```
  return data

# Get Plant Details based on current page
def GetPlantDetails(plantNo):

  phytochemicals = []
  offset = 0
  url = DRDUKES_PHYTOCHEMICALS_URL + str(
      plantNo) + "?offset=" + str(offset) + "&
      max=20"

  while url:
    # Request HTML of Plant Details Page
    r = requests.get(url)

    # Parse Request of Plant Details Page
    soup = BeautifulSoup(r.text, "lxml")

    # Check if Plant Details Subpage contains
        Phythochemicals
    pcs = soup.find_all("a", {"title": "Click
        to view details for this Chemical"})

    if pcs:
      # print('\n [PLANT] {0} [PC PAGE] {1} \n'.
          format(plantNo, int(offset/20) + 1))
      for pc in pcs:
        phytochemicals.append(FixWhiteSpace(pc.
            text))
      offset += 20
      url = DRDUKES_PHYTOCHEMICALS_URL + str(
          plantNo) + "?offset=" + str(offset) +
          "&max=20"
    else:
      break

  if not phytochemicals:
    phytochemicals = "Not Available"

  return phytochemicals

 def FixWhiteSpace(input):
  return ' '.join(input.strip().split())

def RemoveDuplicates(input):
  return list(dict.fromkeys(input))

# Remove columns and rows with zeros
def SimplifyTable(data):
  table = data.copy()
  table = table.loc[(table!=0).any(1)]
  table = table.loc[:,(table!= 0).any(axis=0)]
  return table

# Remove Unwanted Rows
def RemoveRowWithoutValue(data, columnName,
    value):
  table = data.copy()
  table.drop(table.loc[table[columnName] ==
      value].index, inplace=True)
  return table

# Convert Columns Containing String of Lists
    into Lists of Strings
def FixStringListColumn(data, columnName):
  table = data.copy()
  for index, rows in table.iterrows():
    rows[columnName] = ast.literal_eval(rows[
        columnName])
  return table

# Bold printing
def PrintBold(input):
  print("\033[1m " + input + " \033[0m")

def GetPlantsWithSMD(csv, smdsynonyms):
  data = pd.DataFrame(columns = PLANT_DETAILS)

  # Iterate over Plants
  for index, p in csv.iterrows():

    # Obtain plant and remove non-smd
        phytochemicals
    pcList = ast.literal_eval(p["
        Phytochemicals"])
    pcList = RemoveDuplicates(pcList)
    pcList = set(smdsynonyms).intersection(set
        (pcList))

    if pcList:
      data = data.append( {"Scientific Name":
          p["Scientific Name"], "Common Name":
           p["Common Name"], "Phytochemicals":
           pcList}, ignore_index = True)
  return data

# Change Dr Dukes Phytochemical Column to SMD
    Column
def RemoveNonSMDPhytochemicals(dbData, ddData)
    :
  table = ddData.copy()
  smdsynList = []

  # Get All SMD Generic Names
  smdsynList = list(set(smdsynList) | set(
      dbData["Generic Name"]))

  # Get All SMD Common Names
  for row in dbData["Synonyms"]:
    if row != "Not Available":
      syn = ast.literal_eval(row)
      smdsynList = list(set(smdsynList) | set(
          syn))

  # Capitalize SMD-Synonym List for comparison
  smdsynList = [smdsyn.upper() for smdsyn in
      smdsynList]

  # Change Phytochemical Column to SMD column (
      Remove unwanted phytochemicals)
  for index, row in table.iterrows():
    smd = []
    if row["Phytochemicals"] != "Not Available
        ":
      pcs = ast.literal_eval(row["Phytochemicals
          "])
      smd = list(set(smdsynList).intersection(
          set(pcs)))
      if smd:
        row["Phytochemicals"] = smd
      else:
        row["Phytochemicals"] = "Not Available"
```

```
  table.rename(columns = {"Phytochemicals": "
      SMDs"}, inplace = True)

  return table

# Remove Non-plant SMDs
def RemoveNonPlantSMD(genesSMDData,
    plantsSMDData):
  table = genesSMDData.copy()

  plantSMDs = []

  # Obtain All SMDs found in plants
  for index, row in plantsSMDData.iterrows():
    plantSMDs = list(set(row["SMDs"]) | set(
        plantSMDs))

  # Delete SMDs that are not needed by plants
  for index, row in table.iterrows():
    if row["Generic Name"].upper() not in
        plantSMDs:
      row["Generic Name"] = "Not Needed"
    else:
      row["Generic Name"] = row["Generic Name"].
          upper()
      row["Genes"] = ast.literal_eval(row["Genes
          "])

  table = RemoveRowWithoutValue(table, "Generic
      Name", "Not Needed")

  return table

def FilterPlantsOnUpdatedSMD(genesSMDData,
    plantsSMDData):
  table = plantsSMDData.copy()

  smdsynList = []

  # Add SMD Names
  smdsynList = list( set(genesSMDData["Generic
      Name"]) | set(smdsynList) )

  # Add Common Names
  for row in genesSMDData["Synonyms"]:
    if row != "Not Available":
      syn = ast.literal_eval(row)
      smdsynList = list(set(smdsynList) | set(
          syn))

  # Capitalize SMD-Synonym List for comparison
  smdsynList = [smdsyn.upper() for smdsyn in
      smdsynList]

  # Remove SMDs that have no genes
  for index, row in table.iterrows():
    smd = []

    smd = list(set(smdsynList).intersection(set
        (row["SMDs"])))
    if smd:
      row["SMDs"] = smd
    else:
      row["SMDs"] = "Not Available"

  return table

# Create Gene-SMD Table
def CreateGeneSMDTable(genesSMDData):
  genes = []
  smd = []

  # Initialize Table
  for index, row in genesSMDData.iterrows():
    genes = list(set(genes) | set(row["Genes"])
        )

  smd = list(set(genesSMDData["Generic Name"])
      | set(smd))

  table = pd.DataFrame(0, index = genes,
      columns = smd)

  # Fill-up Table
  for index, row in genesSMDData.iterrows():
    for gene in row["Genes"]:
      table[row["Generic Name"]][gene] += 1

  return table

def CreateGenePlantTable(genesSMDData,
    plantsSMDData):

  genes = []
  plants = []

  # Initialize Table
  for index, row in genesSMDData.iterrows():
    genes = list(set(row["Genes"]) | set(genes)
        )

  plants = list(set(plantsSMDData["Scientific
      Name"]) | set(plants))

  table = pd.DataFrame(0, index = genes,
      columns = plants)

  smd = []
  smd = list(set(genesSMDData["Generic Name"])
      | set(smd))

  # Fill-up Table
  for indexP, rowP in plantsSMDData.iterrows():
    smdsPerPlant = list(rowP["SMDs"])

    genes = []
    for indexG, rowG in genesSMDData.iterrows()
        :

      smdsynList = []
      smdsynList.append(rowG["Generic Name"])
      if rowG["Synonyms"] != "Not Available":
        smdsynList = list(set(ast.literal_eval(
            rowG["Synonyms"])) | set(smdsynList)
            )

      # Capitalize SMD-Synonym List for
          comparison
      smdsynList = [smdsyn.upper() for smdsyn in
          smdsynList]

      if set(smdsynList).intersection(set(
          smdsPerPlant)):
```

```
      genes = list(set(rowG["Genes"]) | set(
          genes))

    for indexTC, columnTC in table.iteritems():
      if indexTC == rowP["Scientific Name"]:
        for gene in genes:
          table[indexTC][gene] += 1

  return table


# Create a Graph with Matrix Input
def CreateGraphFromMatrix(matrix):
  G = nx.Graph()

  for indexRow, row in matrix.iterrows():
    for indexCol, column in matrix.iteritems():
      if matrix[indexCol][indexRow] > 0:
        # G.add_edge(*(indexCol, indexRow))
        G.add_edge(indexCol, indexRow, weight =
            matrix[indexCol][indexRow])

  return G

# Draw Graph
def DrawGraph(graph):
  plt.figure(figsize = (12, 12))
  nx.draw(graph, with_labels = True)
  plt.show()

# Get Degree Centrality
def GetDegreeCentrality(G, nodes):
  degCen = {}
  for node in nodes:
    degCen[node] = len(G.edges(node))
  return sorted(degCen.items(), key = lambda
      item: item[1], reverse = True)

# Get Weighted Degree Centrality
def GetWeightedDegreeCentrality(G, nodes):
  weightedDegCen = {}
  for node in nodes:
    wdSum = 0;
    for edge in G.edges(node):
      wdSum += G.get_edge_data(edge[0], edge[1])
          ["weight"]
    weightedDegCen[node] = wdSum

  return sorted(weightedDegCen.items(), key =
      lambda item: item[1], reverse = True)

# View gene associations based from given
    graph
def DisplayAssociationsFromGene(graph,
    nameOfAssociateNodes, gene):
  endString = "" if len(graph.edges(gene)) <= 1
       else "/s"
  PrintBold(gene + " is associated with " + str
      (len(graph.edges(gene))) + " " +
      nameOfAssociateNodes + endString)
  G = nx.Graph()
  for edge in graph.edges(gene):
    G.add_edge(*edge)
  DrawGraph(G)

# Update Drug Bank Data (~ 15 mins)
drugBankData = GetDrugBankData()

# Save Drug Bank Data to CSV
drugBankData.to_csv(DRUGBANK_CSV, index=False)

# Update Dr Dukes Data (~ 43 mins)
drDukesData = GetDrDukesData()

drDukesData.to_csv(DRDUKES_CSV, index=False)

# Get SMD data from DrugBank CSV
drugBankData = pd.read_csv(DRUGBANK_CSV)
print(len(drugBankData))

# Get Plant Data from Dr. Dukes CSV
drDukesData = pd.read_csv(DRDUKES_CSV)
print(len(drDukesData))

# FILTER (genes - smd - PLANTS)

# [1] Include Phytochemicals that are SMD
plantsSMDData = RemoveNonSMDPhytochemicals(
    drugBankData, drDukesData)

# [2] If plant has no smd, genes to plants
    cannot be mapped
plantsSMDData = RemoveRowWithoutValue(
    plantsSMDData, "SMDs", "Not Available")
print(len(plantsSMDData))

# FILTER (GENES - smd - plants)
# [1] If no genes, genes to plants cannot be
    mapped
genesSMDData = RemoveRowWithoutValue(
    drugBankData, "Genes", "Not Available")
print(len(genesSMDData))

# FILTER (genes - SMD - plants)

# [1] Include SMD that are found in plants
genesSMDData = RemoveNonPlantSMD(genesSMDData,
     plantsSMDData)
print(len(genesSMDData))

# FILTER (genes - smd - PLANTS)
# [1] Have to update since some genes have
    been deleted, thus plant cannot map out
plantsSMDData = FilterPlantsOnUpdatedSMD(
    genesSMDData, plantsSMDData)

# [2] If plant has no smd, genes to plants
    cannot be mapped
plantsSMDData = RemoveRowWithoutValue(
    plantsSMDData, "SMDs", "Not Available")
print(len(plantsSMDData))

# Save to CSV (Uncomment if want to update and
     view data)
genesSMDData.to_csv(GENES_SMD_DATA_CSV,index =
     False)
plantsSMDData.to_csv(PLANTS_SMD_DATA_CSV ,
    index = False)

# Create Gene (Rows) - SMD (Columns) Table
geneSMDTable = CreateGeneSMDTable(genesSMDData
    )

# Create Gene (Rows) - Plants (Columns) Table
```

```
genePlantTable = CreateGenePlantTable(
    genesSMDData, plantsSMDData)

# Save to CSV (Uncomment if want to update and
     view data)
geneSMDTable.to_csv(GENE_SMD_TABLE_CSV,index =
     True)
genePlantTable.to_csv(GENE_PLANT_TABLE_CSV ,
    index = True)

# Create Gene - SMD Graph
geneSMDGraph = CreateGraphFromMatrix(
    geneSMDTable)

# Create Gene - Plants Graph
genePlantGraph = CreateGraphFromMatrix(
    genePlantTable)

# Get Degree Centrality For All Genes
dcGeneSMD = GetDegreeCentrality(geneSMDGraph,
    geneSMDTable.index.values)
dcGenePlant = GetDegreeCentrality(
    genePlantGraph, genePlantTable.index.
    values)

# Get WeightedDegree Centrality For All Genes
wdcGeneSMD = GetWeightedDegreeCentrality(
    geneSMDGraph, geneSMDTable.index.values)
wdcGenePlant = GetWeightedDegreeCentrality(
    genePlantGraph, genePlantTable.index.
    values)

# Display Webscrapped DrugBank Data Set
PrintBold("SMD Generic Name, Synonyms, and
    Genes CSV [DrugBank Data Set]")
print(genesSMDData)

print()

# Display Webscrapped Dr Dukes Data Set
PrintBold("Plant Scientific Name, Common Name,
     and SMDs CSV [DrugBank Data Set]")
print(plantsSMDData)

# Display Gene (Rows) - SMD (Columns) Table
PrintBold("Gene (Rows) - SMD (Columns) Table")
print(geneSMDTable)

print()

# Display Gene (Rows) - Plants(Columns) Table
PrintBold("Gene (Rows) - Plants (Columns)
    Table")
print(genePlantTable)

# View Graphs (Graphs are randomly drawn,
    sometimes not centered and is off from
    display)
PrintBold("Gene-SMD Graph")
DrawGraph(geneSMDGraph)
PrintBold("Gene-Plant Graph")
DrawGraph(genePlantGraph)

# Get Degree Centralities of Graphs
PrintBold("Gene-SMD Graph Degree Centrality
    for all Genes")
print(dcGeneSMD)

print()

PrintBold("Gene-Plant Graph Degree Centrality
    for all Genes")
print(dcGenePlant)

# Get Weighted Degree Centralities of Graphs
PrintBold("Gene-SMD Graph Weighted Degree
    Centrality for all Genes")
print(wdcGeneSMD)

print()

PrintBold("Gene-Plant Graph Weighted Degree
    Centrality for all Genes")
print(wdcGenePlant)

# Print graph of the associated SMD/s based on
     a particular gene
DisplayAssociationsFromGene(geneSMDGraph, 'SMD
    ', 'CYP3A4')
DisplayAssociationsFromGene(geneSMDGraph, 'SMD
    ', 'ALB')

# Print graph of the associated plants/s based
     on a particular gene
DisplayAssociationsFromGene(genePlantGraph, '
    Plant', 'CYP3A4')
DisplayAssociationsFromGene(genePlantGraph, '
    Plant', 'ALB')

!pip install sklearn-som
!pip install minisom

from sklearn_som.som import SOM
from minisom import MiniSom
from matplotlib.gridspec import GridSpec
from sklearn.preprocessing import minmax_scale
    , scale

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

#file reading for gene and smd
df = pd.read_csv('/content/drive/MyDrive/Copy
    of HI 193.1 (Group 6) - Final Paper [Drive
    ]/gene-smd-table.csv',sep=',',header=None)
df2 = df.iloc[1: , 1:]
df3 = df2.to_numpy() #minisom uses numpy arraw
df4 = df3.astype(np.float)

gene_name = df.iloc[1:, 0].values
smd_name = df.iloc[0, 1:].values

#Building the som

som = MiniSom(12, 12, len(df4[0]),
          neighborhood_function='gaussian',
              sigma=1.5,
          random_seed=1)
#initialise weights to the map
som.pca_weights_init(df4)

#training the model might take some time to
    finish
```

```
som.train_batch(df4, num_iteration=len(df4)
    *500, verbose=True)

#creates a plot of the SOM for smd and genes
def plotSOM(c):
   if len(c) > 120:
      return gene_name[c]
   else:
      return c

gene_map = som.labels_map(df4, df.iloc[1:, 0].
    values)

plt.figure(figsize=(12, 12))
for p, genes in gene_map.items():
   genes = list(genes)
   x = p[0] + .1
   y = p[1] - .3
   for i, c in enumerate(genes):
      off_set = (i+1)/len(genes) - 0.05
      plt.text(x, y+off_set, plotSOM(c),
          fontsize=7)
plt.pcolor(som.distance_map().T, cmap='seismic
    ', alpha=.2)
plt.xticks(np.arange(12+1))
plt.yticks(np.arange(12+1))
plt.grid()

plt.show()

#file reading for plant and smd
dfp = pd.read_csv('/content/drive/MyDrive/Copy
     of HI 193.1 (Group 6) - Final Paper [
    Drive]/gene-plant-table.csv',sep=',',
    header=None)

dfp2 = dfp.iloc[1: , 1:]
dfp3 = dfp2.to_numpy()
dfp4 = dfp3.astype(np.float)

gene_name = dfp.iloc[1:, 0].values
plant_name = dfp.iloc[0, 1:].values
print(plant_name)
print(gene_name)
dfp4

som = MiniSom(15, 15, len(dfp4[0]),
          neighborhood_function='gaussian',
              sigma=1.5,
          random_seed=1)
som.pca_weights_init(dfp4)
som.train_batch(dfp4, num_iteration=len(dfp4)
    *500, verbose=True) #this will take some
    time to finish

#creates a plot for plant and smd SOM
def plotSOM2(c):
   if len(c) > 1064:
      return gene_name[c]
   else:
      return c

gene_map = som.labels_map(dfp4, dfp.iloc[1:,
    0].values)

plt.figure(figsize=(15, 15))
for p, genes in gene_map.items():
   genes = list(genes)
   x = p[0] + .1
   y = p[1] - .3
   for i, c in enumerate(genes):
      off_set = (i+1)/len(genes) - 0.05
      plt.text(x, y+off_set, plotSOM2(c),
          fontsize=7)
plt.pcolor(som.distance_map().T, cmap='seismic
    ', alpha=.2)
plt.xticks(np.arange(15+1))
plt.yticks(np.arange(15+1))
plt.grid()

plt.show()
```

REFERENCES

[1] Jazmine Mae Recario, Geoffrey Solano, and Angelyn Lao. "PlaPhy-SMD: A Database System of Plants and their Phytochemicals Integrated with Small-Molecule Drugs and their Condition and Gene Associations with Clustering and Product Ingredient Matching". In: *2020 11th International Conference on Information, Intelligence, Systems and Applications (IISA*. IEEE. 2020, pp. 1–8.

[2] Atanas G Atanasov et al. "Discovery and resupply of pharmacologically active plant-derived natural products: A review". In: *Biotechnology advances* 33.8 (2015), pp. 1582–1614.

[3] N Chattopadhyay and R Maurya. "Herbal Medicine". In: (2015).

[4] Chit Shing Jackson Woo, Jonathan See Han Lau, and Hani El-Nezami. "Herbal medicine: toxicity and recent trends in assessing their potential toxic effects". In: *Advances in botanical research* 62 (2012), pp. 365–384.

[5] Kristina B Martinez, Jessica D Mackert, and Michael K McIntosh. "Polyphenols and intestinal health". In: *Nutrition and functional foods for healthy aging*. Elsevier, 2017, pp. 191–210.

[6] H Wiseman. "Phytochemicals: health effects". In: (2013).

[7] Munish Chhabra. "Biological therapeutic modalities". In: *Translational Biotechnology*. Elsevier, 2021, pp. 137–164.

[8] HM Hatcher. "Principles of systemic therapy". In: *Specialist Training in Oncology*. Elsevier, 2011, pp. 30–44.

[9] Lachy McLean. "Drug development". In: *Rheumatology*. Elsevier, 2015, pp. 395–400.

[10] Andre Dauphine. *Geographical Models with Mathematica*. Elsevier, 2017.

[11] Derek L Hansen et al. "Social network analysis: measuring, mapping, and modeling collections of connections". In: *Analyzing social media networks with NodeXL* (2011), pp. 31–51.

[12] Jennifer Golbeck. *Introduction to social media investigation: a hands-on approach*. Syngress, 2015.

[13] SG Roy and A Chakrabarti. “A novel graph clustering algorithm based on discrete-time quantum random walk”. In: *Quantum Inspired Computational Intelligence*. Elsevier, 2017, pp. 361–389.

[14] Ramón Cacabelos, Pablo Cacabelos, and Clara Torrellas. “Personalized medicine of Alzheimer’s disease”. In: *Handbook of pharmacogenomics and stratified medicine* (2014), p. 563.

[15] Rommel G Tirona and Richard B Kim. “Introduction to Clinical Pharmacology”. In: *Clinical and Translational Science*. Elsevier, 2017, pp. 365–388.

[16] Parastou Rahimizadeh, Sungtae Yang, and Sung In Lim. “Albumin: An Emerging Opportunity in Drug Delivery”. In: *Biotechnology and Bioprocess Engineering* (2020), pp. 1–11.

[17] Ulrich Kragh-Hansen. “Human serum albumin: a multifunctional protein”. In: *Albumin in Medicine*. Springer, 2016, pp. 1–24.