2018

# Final Report

GRAPH PROBLEM SOLVER

JASON CHUANG

# I. Abstract

Real-world problems can be commonly modeled as graphs. Different traveling methods can be modeled as a graph problem. Finding different travel methods and finding the most optimal travel method such as in the Traveling Salesman problem can be modeled with graphs. Amazon, Uber, Lyft all use graph problems, to optimize travel path and most efficient. Analyzing these graphs can allow us to find solutions and fascinating traits. Through these problems we can solve a graph and extract as much information as necessary to solve a problem. A variety of problems can be derived from a graph. A common one is the shortest path algorithm. The shortest path algorithm involves finding the shortest path between two nodes on a graph. This problem can be solved in NP-Complete, meaning that it is solved in polynomial time. Many of these problems such as the ones I used to help solve the graph are similarly NP-Complete. In this paper we will explore the problems solved on my webpage using common algorithms and methods using NetworkX, a library used to facilitate graph problem solving. In this paper, we will explore the problems I solved and implemented onto my webpage. These problems include Dominance, Dominating Sets, Eulerian, and Efficiency. These common problems have many subproblems that are not easily solved.

# II. Introduction

Graph problems are extremely common in many practical applications. As mentioned above many big companies utilize graphs to model problems to solve. Along with companies utilizing graph problems to find and solve issues, graphs are widely used in creation of neural networks. Using knowledge graphs, AI can be created. An example can be seen in the use of AI in Google, which utilizes Knowledge Graphs.

Dominance, Dominating Set, Eulerian, and Efficiency are common problems found in graph theory. By solving these problems and applying them to graphs we are able to use these to better utilize graphs in modeling problems. Dominance, Dominating Set, Eulerian, and Efficiency problems can be divided into several subproblems. These subproblems help make up the overall common issue. By solving these subproblems we are then able to solve the bigger problem.

Dominance can be split into several subproblems. Immediate Dominators and Dominance Frontiers are the subproblems of dominance. These subproblems are some examples of dominance. Immediate Dominators and the dominance frontier are special kind of dominance expressed by nodes of a graph.

Dominating Sets also contain several subproblems. Finding a dominating set is a NP-Complete problem and therefore impossible to complete in non-polynomial time. Finding the dominating set proves to be very difficult and costly. In this problem, subproblems include finding a dominating set (not necessarily the smallest one) and checking if a group of nodes is a dominating set of the graph.

Efficiency has subproblems of its own that can be split to be several different kinds. Efficiency can be expressed as a graph's average local efficiency. Besides finding a graphs local efficiency, it is also possible to find a graphs average global efficiency. Both these subproblems help express the efficiency of a graph in different ways. Through these subproblems we can more qualitatively describe a graph's efficiency with a more holistic scope as compared to just the efficiency.

The next problem is Eulerian. This can also be divided into several subproblems. In Eulerian problem, we have subproblems that check whether a given graph is Eulerian, which is defined as a graph that has Eulerian circuit. Another subproblem is to give an iterator to iterate over the

edges of the Eulerian circuit in the graph. If the graph does not already contain a Eulerian circuit we are able to turn it into a Eulerian graph through transformation of the edges.

In this paper I will discuss the implementation of these functions in my webpage using NetworkX, my webpage creation, user interface, methods, and the uses of these functions in real life problems to find solutions.

# III. Background

Graph problems are very common in real world applications. Corporations and programmers use graphs to model problems and as result are able to solve them more easily. Graph problems are one of the fundamental building blocks of AI and are used widely in companies such as Google Amazon Uber and Lyft. Using knowledge graphs for AI is common and very appropriate to using nodes to represent knowledge. Graph problems such as the shortest problem algorithm are used widely to find the most efficient route from one node to another node. A common graph problem use can be seen in Google maps where the map attempts to find the shortest path to a destination given a variety of parameters.

One of the problems solved on my webpage is the Dominance problem. The immediate dominators are one of the dominance subproblems. To describe immediate dominators, it is necessary to understand what dominator is. A dominator is explained by a node d dominates node n if every entry path n must go through d. There can be many different types of dominators including immediate dominators strictly dominator and dominance frontier. Strictly dominate can be defined as a node n that is dominated by d but does not equal n. An immediate dominator of a node is a node that dominates n but doesn’t dominate other node that strictly dominates n. Dominators can be extremely useful in compilers and computing static single assignment form. Dominators can help in compiler optimizations especially dominating set. The second

subproblem solved by the dominators are finding the dominance frontiers. Dominance frontiers of a node d is the set of all nodes n that d dominates a predecessor of n but d does not strictly dominate n. It is basically the set of nodes where node d dominances stops. This is extremely important to speed up the compilation times of programs which can be very long depending on the size of code.

The second problem solved on my webpage are dominating sets. Dominating sets is a graph with node set V is a subset D of V such that every node not in D is adjacent to at least one member of D. These nodes can cover the entire graph. This subproblem is finding one of the subproblems, not necessarily the smallest one. The NP-complete dominating set problem involves finding the smallest dominating set. This NP-complete problem has no efficient algorithm to find the smallest dominating set, but finding any size of dominating set is rather simple. Dominating set are widely used in many applications. Dominating sets are widely used in wireless networking, dominating sets are used to find efficient routes within ad-hoc mobile networks. By ensuring that a graph of wireless network signals dominating set is active ensures that a wide area is covered wirelessly of the graph. Through this method, a wide area coverage can be made when an efficient route can be made for a mobile network. Dominating sets are also widely used in designing secure systems for electrical grids. Another subproblem of dominating sets is to be able to see if a set of nodes is a dominating set of a graph. This allows to check if something is a dominating set.

The third problem solves on my webpage involves the efficiency of a graph. A graph's efficiency is the multiplicative inverse of the shortest path between two nodes. By finding the inverse of the shortest path between two nodes we calculate how efficient it is. In our webpage we were able to calculate the efficiency of a graph as well as the average local efficiency and

average global efficiency. Global efficiency is the average efficiency of all pairs of nodes. Local efficiency is the average global efficiency of the subgraph by neighbors of the node. The efficiency of a graph is extremely important in quantifying how efficiently a network exchanges information. Efficiency talks about how good a network or graph where information is exchanged between each node. The efficiency of a network has many uses in networking. Efficiency can be found to find the most cost-effective networks. Using global efficiency, we are able to find the lowest most efficient network is. Efficiency is also good in finding good man-made networks such as transportation and communications. Efficiency is extremely important to determine how good a network is. Efficiency, or more specifically global efficiency, can be used to quantify how reliable and good a network is by minimizing distance between nodes. Being able to quantify the effectiveness and reliable a tolerant is extremely important and being able to compare different efficiency of networks we can choose a better model.

The last problem on my webpage is the Eulerian problem. In the eulerian problem the subproblems can check if a graph is eulerian. A eulerian graph is defined as a graph that has a eulerian circuit. A eulerian circuit is a closed loop that includes each of the graph once. Another subproblem is that we want to be able to iterate over all edges of a Eulerian graph. Another subproblem is being able to eulerize a graph. Eulerizing the graph will transform the graph into a eulerian graph by changing the edges to create a eulerian circuit. Eulerian graph has many applications in the real world. Eulerian trails are used to help in DNA construction. It is also used in creating CMOS circuit design and logic gate design. A practical application for Eulerian graph is the Konisberg Bridge Problem. The Konisberg Bridge problem involves a two islands in a river and seven bridges. Someone must plan a walk so that they cross the bridge once. This problem lead to the creation of a branch of mathematics called graph theory.

# IV. Methods

How I created my project was quite complicated. From networkx the functions associated with the dominance problem are immediate_dominators(G,start) and dominance_frontiers(G,start). The first one immediate_dominators(G,start) returns immediate dominators given a graph and a starting node. It will find the immediate dominator of the given node and return a list of the immediate dominators. The next function dominance_frontier(G,start) allows users to pass in a graph and starting node and then returns a list of the dominance frontier of the starting node.

The functions utilized in my program for dominating sets are dominating_set(G[,start_with]) and is_dominating_set(G,nbunch). Dominating_set(G[,start_with]) finds a dominating set for graph G. The next is_dominating_set(G,nbunch) is a function that checks if the given list of items nbunch is a dominating set for G if it is the function will return True and if its not it will return false.

Next for the efficiency problem, the subproblems used from NetworkX include efficiency(G,u,v), local_efficiency(G), and global_efficiency(G). The first function efficiency(G,u,v) takes in a graph and two starting nodes u and v. It then gives the inverse of the shortest path of the two nodes u and v. The next function local_efficiency(G) takes in a graph and returns the average local efficiency of the graph. The last function global_efficeincy(G) takes a graph and then returns the average global_efficiency of the graph which I defined above.

The eulerian functions used include is_eulerian(G), eulerian_circuit(G,[,source,keys]), and eulerize(G). The is_eulerian(G) checks whether the graph contains a eulerian circuit. Eulerian_circuit(G[,source,keys]) allows input of a graph G and returns an iterator over edges of

an Eulerian circuit. Euerlerize(G) transforms a graph into a eulerian graph by changing the edges of the graph.

In my web- based program I also use some functions I created on my own. In my first page of my webpage which is saved under index.py. The python cgi creates a script to generate a webpage by printing in a bunch of lines of html code. In the html code I utilized CSS and Javascript to create a more defined user experience and passed in parameters such as csv graph through a browse files button that they can click on the csv file they wish to solve problem fopr and the problem to be solved on the graph through a form. In the form I utilize a dropdown menu in order for the user to select which subproblem they wish to preform on the given graph.

```
print'<form id="filesubmit" enctype="multipart/form-data" action="save_file.py" method="post"><p>'
print'<div class = "text">CSV File:   <input type="file" name="file"></div>'
print '<br><br>'
print'<div id = "change"></div><div id = "description"></div><div id = "input"><input type = "text" name = "parameters"></div>
print'<br><br>'

print'<b><div class = "text">Problem</div><b>'
print'<select name ="function" onchange = "changeParams(this.value)" display = "inline")">'
print'<option disabled = "disabled" selected = "selected"> Select an option. </option>'
print'<option value = "immediate_dominators">Find the Immediate Dominators (Give Starting Node)</option>'
print'<option value = "dominance_frontier">Dominance Frontier (Give Starting Node)</option>'
print'<option value = "dominating_set">Give the Dominating Set (Give Starting Node)</option>'
print'<option value = "is_dominating_set">Is this a Dominating Set? (Give Nodes Seperated by comma)</option>'
print'<option value = "efficiency">Find Efficiency (Give Two Nodes Seperated by a Space)</option>'
print'<option value = "local_efficiency">Local Efficiency</option>'
print'<option value = "global_efficiency">Global Efficiency</option>'
print'<option value = "isEulerian">Is it a Eulerian Graph?</option>'
print'<option value = "eulerian_circuit">Iterate over Eularian circuit</option>'
print'<option value = "eulerize">Euerlize the circuit</option>'
print'</select>'
print'<p><input type="submit" class="button" value="Solve"></p></form>'
print'</div>'
print'</body>'
print'</html>'
```

*Figure 1: Picture of form and parameters being sent through post method to save_file.py*

In the Javascript section, I utilized identification by element ID to find element and change the state of them if the and to change the parameters to be passed in through the form based on the function chosen. For example for the immediate dominators and dominance frontier options

require a node to be passed in with the graph. The webpage will load a text box to enter in the node to find the immediate dominator for or dominance frontier for if that option is chosen. On subproblems that do not require parameters to be passed in besides graph such as local_efficiency or global_ efficiency, the text box is turned invisible since it is not necessary to pass in anything other than the graph to preform the subproblem on.

```
print""<script>function changeParams(val) {
    console.log(val);
     if (val == "immediate_dominators" || val == "dominance_frontier" || val == "dominating_set") {
        document.getElementById("change").innerHTML = "Give Starting Node";
        document.getElementById("input").style.display="inline";
        if(val == "immediate_dominators"){
            document.getElementById("description").innerHTML = "Immediate dominator or idom of a node n is the unique node that strictly dominates n but does not strictly dominate any other node that strictly dominates n. Every node, except the entry node, has an immediate dominator.";
        }

        else if (val == "dominance_frontier"){
            document.getElementById("description").innerHTML = "The dominance frontier of a node d is the set of all nodes n such that d dominates an immediate predecessor of n, but d does not strictly dominate n. It is the set of nodes where d's dominance stops."

        }
        else {
            document.getElementById("description").innerHTML = "A dominating set for a graph with node set V is a subset D of V such that every node not in D is adjacent to at least one member of D"
            }
    }
     else if (val == "is_dominating_set"){
        document.getElementById("change").innerHTML = "Give Set of Nodes ex. 1,2,3";
        document.getElementById("input").style.display="inline";
        document.getElementById("description").innerHTML = "A dominating set for a graph with node set V is a subset D of V such that every node not in D is adjacent to at least one member of D";

        }
     else if ( val == "efficiency"){
        document.getElementById("change").innerHTML = "Give Two Nodes seperated by space ex. 1 3";
        document.getElementById("input").style.display="inline";
        document.getElementById("description").innerHTML = "The efficiency of a pair of nodes is the multiplicative inverse of the shortest path distance between the nodes. Returns 0 if no path between nodes.";
        }
     else if (val == "local_efficiency") {
        document.getElementById("change").innerHTML = "";
        document.getElementById("input").style.display="none";
        document.getElementById("description").innerHTML = "The local efficiency of a node in the graph is the average global efficiency of the subgraph induced by the neighbors of the node. The average local efficiency is the average of the local efficiencies of each node.";
        }
     else if ( val == "global_efficiency"){
        document.getElementById("change").innerHTML = "";
        document.getElementById("input").style.display="none";
        document.getElementById("description").innerHTML = "The average global efficiency of a graph is the average efficiency of all pairs of nodes";
    }
    else if (val == "isEulerian"){
        document.getElementById("change").innerHTML="";
        document.getElementById("input").style.display="none";
        document.getElementById("description").innerHTML = "Checks if graph has Eulerian circuit. Eulerian circuit is a path that includes each oedge of graph exactly once";

    }
```

*Figure 2: interactive JavaScript code asking to pass in parameters through the form*

On the next page save_file.py I look at the graph in the function I defined called show(). In this function I iterate through the lines of the csv graph and create a graph object using networkx. In the graph I iterate through each line of the python code and grab each of the values of edges and nodes of the graph. I then add weighted edges and read the problem the user wanted to solve from the index.py file. This is done through a post request from the form sent earlier. With these parameter values sent to save_file.py I have a long list of conditional statements finding what the

subproblem the user wanted solved and in these conditional statements preform a different operation and print the result. I then append different color nodes if the operation requires the identification of several special nodes. The nodes have a default color of green, however, if the subproblem chosen specifically chooses these nodes that exist in the result they are colored blue.

```
if problem =='dominance_frontier':
    if param:
        print '<div class ="text">Dominace Frontier of '+param+':'
    else:
        print'<div class = "text"> Starting node not given</div>'
    print(list(nx.dominance_frontiers(G, int(param))))
    print '</div>'
    for node in G:
        if node in list(nx.dominance_frontiers(G, int(param))):
            color_map.append('blue')
        else:
            color_map.append('green')
elif problem == 'immediate_dominators':
    if param:
        print '<div class ="text">Immediate Dominators of ' + param + ':'
        print(list(nx.immediate_dominators(G, int(param))))
        print '</div>'
    else:
        print'<div class = "text"> Starting node not given</div>'
        print(list(nx.immediate_dominators(G, int(param))))
    for node in G:
        if node in list(nx.immediate_dominators(G, int(param))):
            color_map.append('blue')
        else:
            color_map.append('green')
elif problem =='dominating_set':
    if param:
        print '<div class ="text">Dominating Set of '+param+':'
        print list(nx.dominating_set(G))
        print '</div>'

    else:
        print'<div class = "text"> Starting node not given</div>'
    for node in G:
        if node in list(nx.dominating_set(G,int(param))):
            color_map.append('blue')
        else:
            color_map.append('green')
elif problem == 'is_dominating_set':
    if param:
        nodal = param.split(',')
        intarr = [int(numeric_string) for numeric_string in nodal]
    else:
        print'<div class = "text">Set not given</div>'

    if param:
        print '<div class = "text">Is ' + param + ' a dominating set?:'
        print nx.is_dominating_set(G,intarr)
    print '</div>'
    for node in G:
        if node in intarr:
            color_map.append('blue')
        else:
            color_map.append('green')
# print(list(nx.efficiency(G,0,2)))
```

*Figure 3: Conditional statements in save_file.py show() function checking for subproblem wanted by user sent from form of previous page.*

In the show function, I also create the graph object and append the nodes and edges from the given csv file and from this graph we solve the subproblems through through networkX later on in the code after finding which subproblem was selected.

```
if fileitem.filename :
    #print('test')

    # strip leading path from file name
    # to avoid directory traversal attacks
    fn = os.path.basename(fileitem.filename)
    message = 'The file "' + fn + '" was uploaded successfully'
    with open(fileitem.filename) as csvfile:  # read the csv file, row by row
        reader = csv.reader(csvfile)
        next(reader)
        source = []
        target = []
        value = []
        for row in reader:
            source.append(int(row[0]))
            target.append(int(row[1]))
            value.append(int(row[2]))

    G = nx.DiGraph()
    fig = figure()
    # G = nx.Graph() //uses undirected graph for some subproblems

    # add nodes to the graph
    for i in range(0, np.size(target) + 1):
        G.add_node(i)

    # add weightededges to the graph
    for i in range(np.size(source)):
        G.add_weighted_edges_from([(source[i], target[i], value[i])])
    pos = nx.spring_layout(G)
    # graph is loaded completely
    UG = nx.Graph(G)
    print'<div class="big">'
    color_map = []
```

*Figure 4: Code showing the opening of csv file and appending to the graph object G=nx.DiGraph*

In this code listed above you will also see a color_map =[]. This is defined to keep track of the colors of the nodes. As I mentioned above, special nodes will have different colors and this color_map helps keep track of special nodes to change their colors by appending different color nodes to the list when they are special.

# V. User Interface

My webpage has a very intuitive appealing user interface utilizing CSS JavaScript and HTML. Using CSS and HTML I created a navigational bar on the top of the page as is required for any respectable webpage. At the top of the page I list my name and project with big bold letters to draw attention with “Jason Chuang Graph Problem” In my navigation bar I kept relatively simple with only two tabs available of Problem Solver and Results. In my navigation bar the highlighted current active tab is highlighted.


*Figure 5: Webpage title page and navigational bar*

After thinking about color schemes and best available options, I decided to go with a fun and professional color scheme utilizing essentially four hexadecimal colors for the color scheme of the entire website. By utilizing colors that are compatible I am able to create a holistic experience in which the user recognizes the colors being utilized and is trained to see the colors and recognize what they are doing. The four colors primarily used are #4ABDAC, #FC4A1A, #F7B733, and #DFDCE3. These colors are extremely appealing to the eyes and complement each other extremely well.

```
  background-color: #4ABDAC;
  border:none;
  color:white;
  padding: 15px 32px;
  text-align:center;
  text-decoration:none;
  display:inline-block;
  font-size:18px;
  font-family: Georgia;
  margin:4px 2px;
  cursor:pointer;
  }

.button:hover {
    background-color:#F7B733;

  }
"""
print""".topnav {
  overflow: hidden;
  background-color: #4ABDAC;
}

.topnav a {
  float: left;
  color: black;
  text-align: center;
  padding: 14px 20px;
  text-decoration: none;
  font-family: Georgia;
  font-size: 17px;
   #FC4A1A

}

.topnav a:hover {
  background-color: #F7B733;
  color: black;
}

.topnav a.active {
  background-color: #FC4A1A;
  color: white;

  }
.header {
    font-size:40px;
    color:black;
    font-family:Georgia;
    display:block;
    background:#DFDCE3
    }
```

*Figure 6: CSS snippet of User Interface*

By providing similar colors throughout my entire webpage, I am able to provide a user experience that is agreeable and increase user experience. As you can see from the snippet above I made all the hoverable effects #F7B733. By doing so the user will become accustomed and learn that when they are hovering over something it is the Sunshine color of #F7B733. I also made sure that all of these colors were incorporated into all pages of the webpage. If you look closely at the color of the webpage I changed the color of the entire background of the webpage to be an appealing clean look of #DFDCE3.

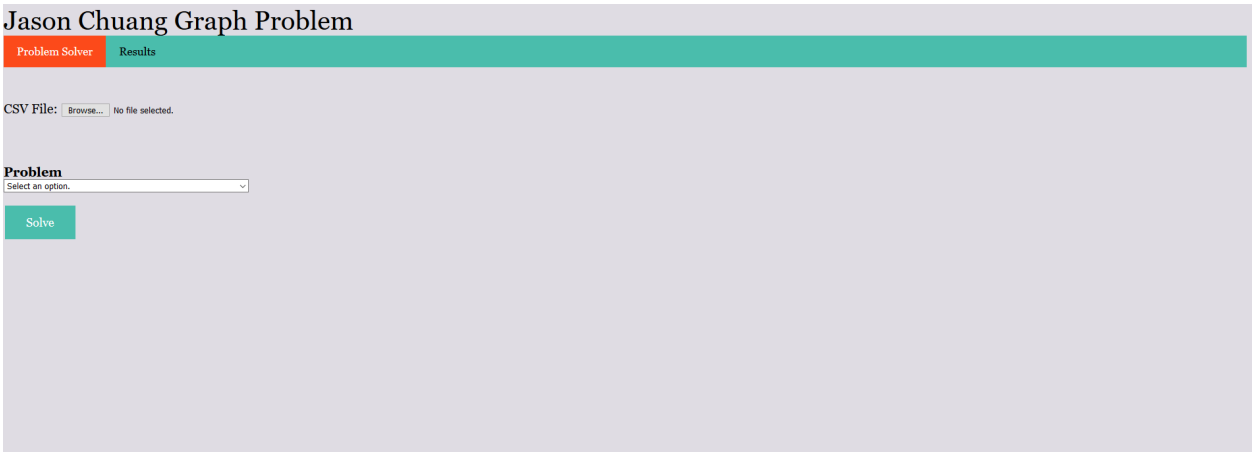



*Figure 7: Webpage user interface*

In the example of my home page shown above I ensured the color scheme is uniform and specifically only those 4 specific colors mentioned earlier.

After selecting an option for the problem to be solved, a text box asking for parameters will appear. For example, if Dominating Set or Immediate Dominators is selected a text box asking for the starting node will appear. Along with a text box asking for a prompt, when any of the choices are selected, a description of the subproblem will be given for each option.

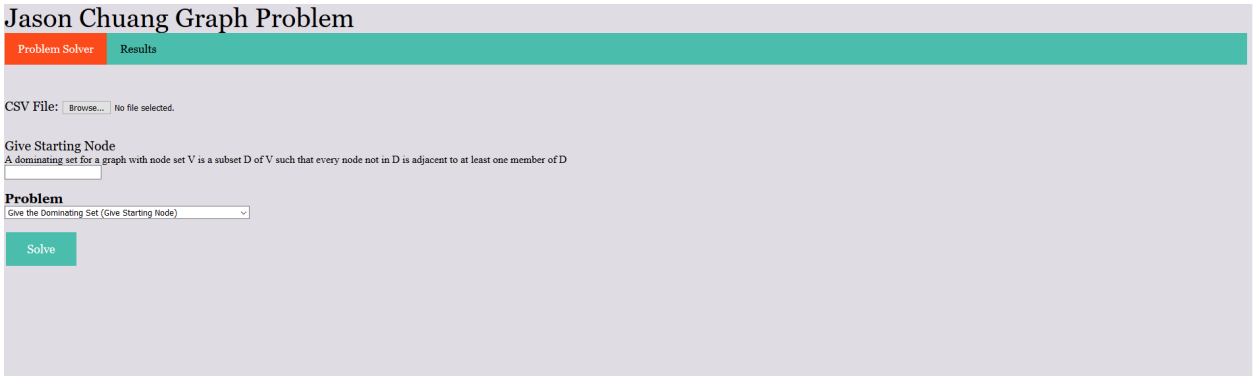



*Figure 8: Problem selection along with text box asking for parameters and a short description of the problem selected.*

The different text appearing from form selection is done using a lot of JavaScript code utilizing html ID. By identifying specific HTML by there ID we were able to make them disappear by using display-block: none to make them seem like they are not there to the webpage user. When they are there, but just invisible.


*Figure 8: JavaScript code conditional statements to find subproblem selected.*

Although the JavaScript took a lot of work, it helped in defining each subproblem and making user experience much more intuitive and helpful to the user in understanding what the function does and what items are needed to pass into the parameters.

On the next page after selecting a subproblem along with selecting a input parameter if necessary, the output will be displayed on the next page. In the output I listed what the input if there was one, and the resultant output of the subproblem. I also display a picture of the graph of the output. If for example the problem was dominating set, I would highlight the dominating set on the picture of the graph in blue instead of the basic green.

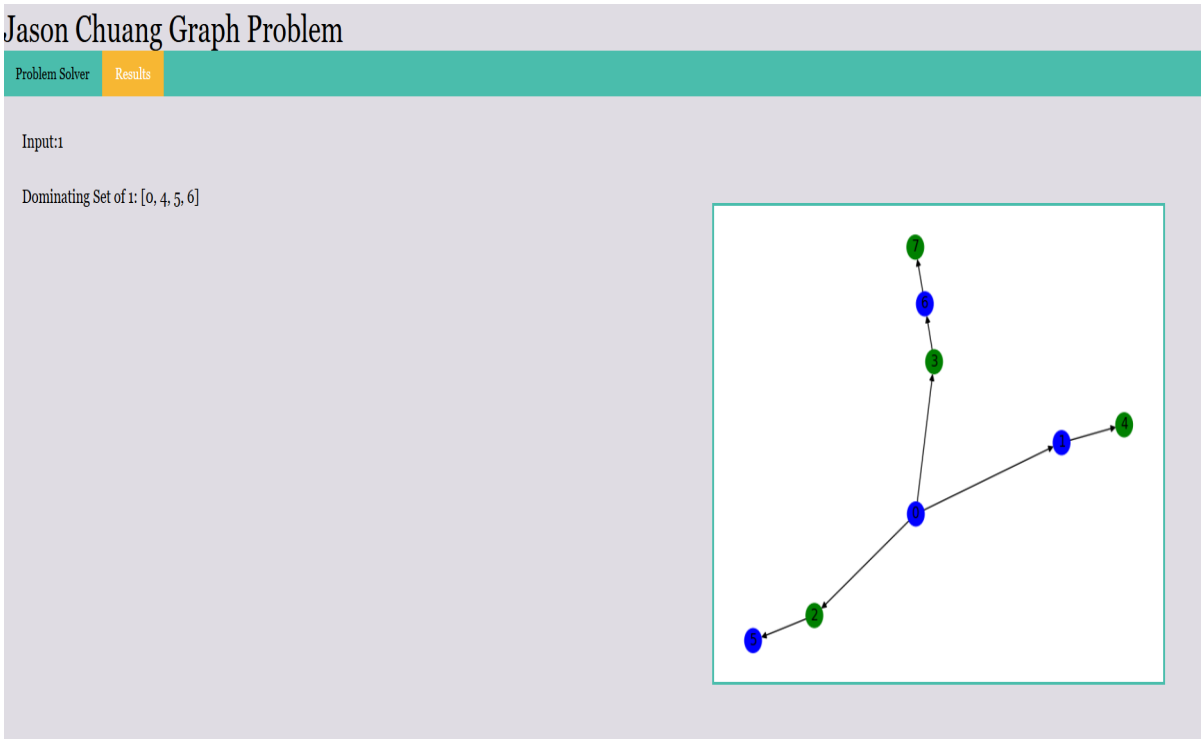



*Figure 9: Output of problem with input*

In the output of the subproblem along with the graph as you can see in the picture above, the results of the dominating set are highlighted in blue on the graph to show a difference from the base color green. Also notice the continue color scheme of the entire page similar to the previous one, along with the same navbar and the background of the webpage color. If the user wishes to go back to the main page after this subproblem, through the navbar, they are able to click on problem solver to solve another graph problem as they wish. Similar to before, the color of the current active tab on the navigation bar is highlighted.

If the user fails to input a graph or when the user fails to give an input parameter when one is required the next page will ask them to enter one. This is so if the user forgets or accidently clicks on the next page they will not be panicked at a blank screen and helps inform the user of why the error happened.

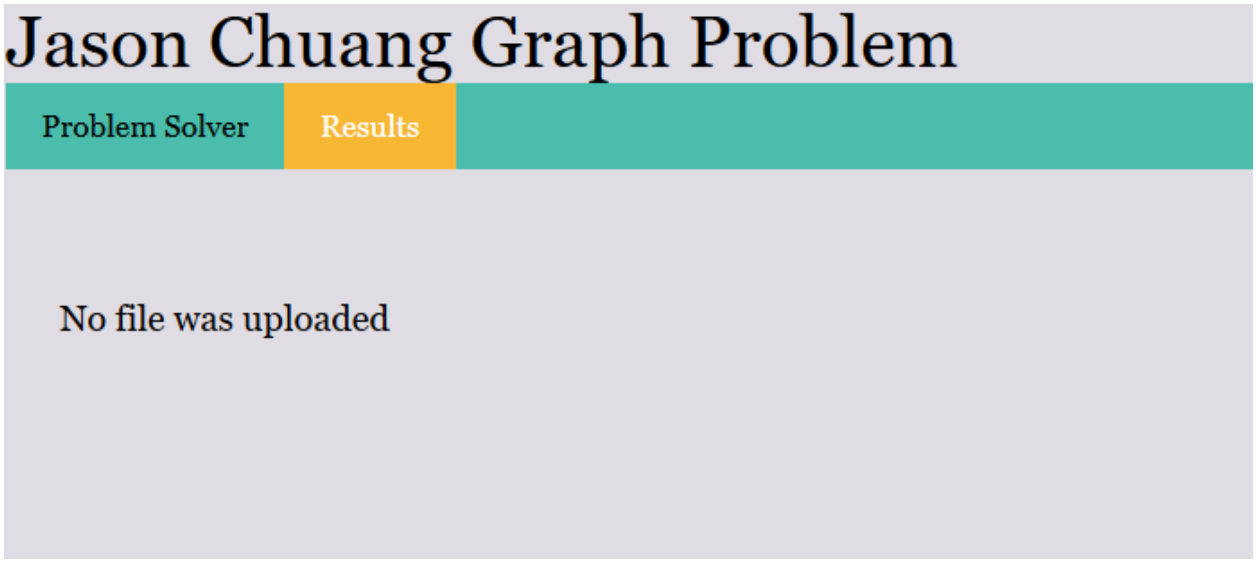



*Figure 10: No file uploaded*

The user interface of my webpage is extremely intuitive, logical, and appealing. By improving the user interface I was able to take a glance into the world of design and color schemes. My webpage features a clean design with little mess or clutter and is a very minimalist design.

## VI. Test Plan

In order to test my project I utilized a white-box testing methodology since I am aware of the design and how it works. To begin with, I utilized a bunch of test cases for each input and tried some edge cases or even user mistakes such as user forgetting to put an input csv file or user forgetting to input a parameter that is needed. However, with knowledge of the nodes and number of nodes in the csv file is important. By testing every edge case and all possible user inputs and errors, I plan to make my application foolproof for the user to destroy or mess with. To begin testing I tried every combination of user selected subproblems in index.py and tested if

each selection in the form had the proper inputs. If the subproblem requires additional input besides the graph, a textbox would appear and ask them about it, as well as personal descriptions of each subproblem when they are selected. After testing the forms from index.py its time to test if we get the proper results in the output save_file.py. After testing simple cases with each subproblem and seeing that the correct graph is displayed as well as the correct input and output is displayed. After that we will try every cases, such as testing the true and false of every Boolean subproblem output. I will also try testing strange inputs and seeing what kind of outputs they give. I will also try changing the csv file into more complex, but logical graphs.

# VII. Results

In the end after much testing, my web application failed to consider if a starting node for immediate dominators and dominance frontier is given for a node that doesn't exist in the csv file. However, it does not crash my application. On the output page of save_file.py it gives an output of nothing since the input does not make sense. So if faulty user input is considered then my program will not output anything but it does not explicitly give an error from tomcat or anything else. It seems that after testing my own application that I forgot to account for different csv files. When reading the user file in the first page of index.py I should have saved the file the user uploaded into the server and then used that file in the next page, however I just used the name which I have been testing it with which is the basic file graph.csv. The python code on save_file.py grabs that file that already exists in the server. Therefore if trying to use a different file my program fails to produce an output. Since it only recognizes graph.csv by the file already in the server. So when using a different csv file my webpage doesn't produce an output page. But using the base file works well and if you manually input a file into the server and then select it in

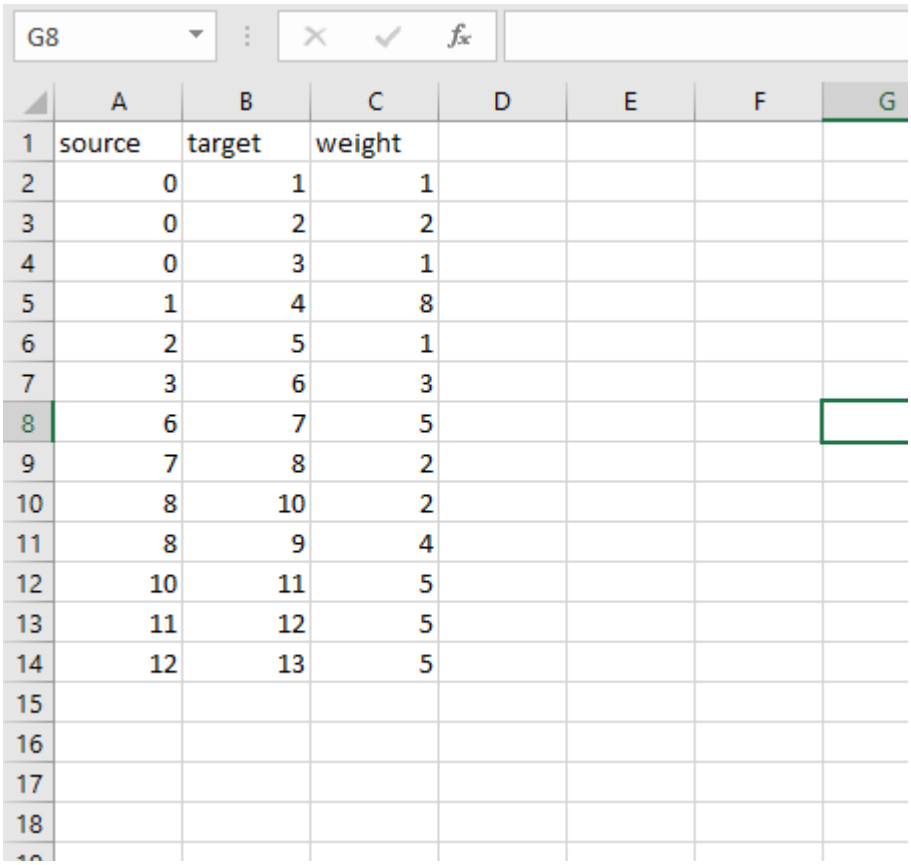
the index.py. I could have done this by using python to just read the file and save the file into the server. This was a mistake on my part

## VIII. Partner Testing

In testing my partner, Yan Zhang's, graph problem I plant to test a variety of inputs and utilize black box testing after reading description of what each function does since I do not know the design. From this I am will be able to compare the expected output to the results of the output from Yan Zhang's problem. Only through extensive testing will I be able to tell if each of the functions as expected and I must produce edge cases such as incorrect input and different csv files.

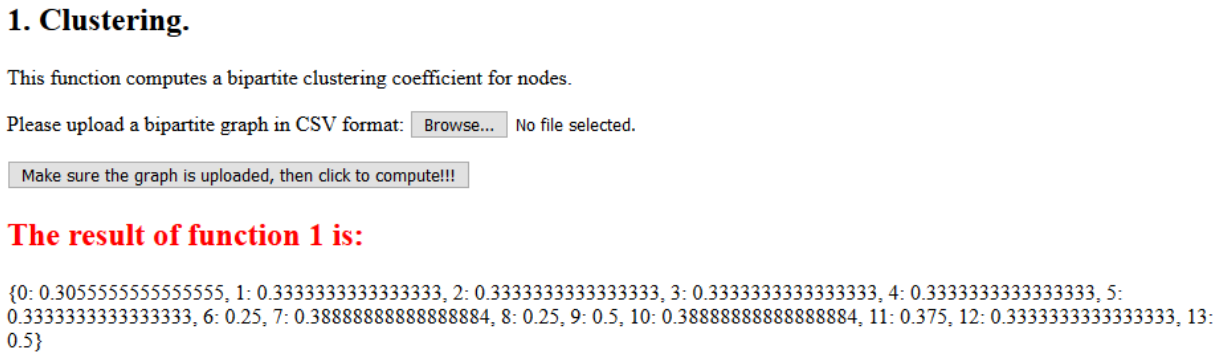
## IX. Partner testing Results

Testing the functions of my partner, it seems very intuitive layout and he has labeled all the important necessary information, but I feel like its too much text and it is a lot of reading. I appreciate the red text to highlight importance of the text. It seems when trying to run many of the functions with just the default values, a Tomcat error occurs. This can be seen in the Configuration model, Havel, Hakimi Graph, Reverse Havel Hakimi graph, Alternating Havel Hakimi Graph, Random Graph. For the generator part of my partner's lab it seems that only the first and last function work which are the Complete bipartite graph and the Gnmk random graph. The Clustering functions all seem to work with any size or correctly formatted csv file. I made my own csv file for testing purposes and added more nodes to see if the webpage would register them and they webpage did.



| | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| 1 | source | target | weight | | | | |
| 2 | 0 | 1 | 1 | | | | |
| 3 | 0 | 2 | 2 | | | | |
| 4 | 0 | 3 | 1 | | | | |
| 5 | 1 | 4 | 8 | | | | |
| 6 | 2 | 5 | 1 | | | | |
| 7 | 3 | 6 | 3 | | | | |
| 8 | 6 | 7 | 5 | | | | |
| 9 | 7 | 8 | 2 | | | | |
| 10 | 8 | 10 | 2 | | | | |
| 11 | 8 | 9 | 4 | | | | |
| 12 | 10 | 11 | 5 | | | | |
| 13 | 11 | 12 | 5 | | | | |
| 14 | 12 | 13 | 5 | | | | |
| 15 | | | | | | | |
| 16 | | | | | | | |
| 17 | | | | | | | |
| 18 | | | | | | | |

*Figure 11: test csv file used*



**1. Clustering.**

This function computes a bipartite clustering coefficient for nodes.

Please upload a bipartite graph in CSV format: Browse... No file selected.

Make sure the graph is uploaded, then click to compute!!!

**The result of function 1 is:**

{0: 0.3055555555555555, 1: 0.3333333333333333, 2: 0.3333333333333333, 3: 0.3333333333333333, 4: 0.3333333333333333, 5: 0.3333333333333333, 6: 0.25, 7: 0.38888888888888884, 8: 0.25, 9: 0.5, 10: 0.38888888888888884, 11: 0.375, 12: 0.3333333333333333, 13: 0.5}

*Figure 12 : results for clustering of test csv file*

As you can see from the results you can see that it registered the newly added nodes from the test file and correctly found the results. This was reflected in the other functions in which entering a file was necessary.

The covering functions also seem to work with given csv files. My partners program works well with different and new csv files. The centrality functions all seem to work as well. The result of the redundancy is hard to test since an input of a bipartite graph is needed which I have no idea how to implement or provide in csv file. The results for the testing can be seen below for generation can be seen below.

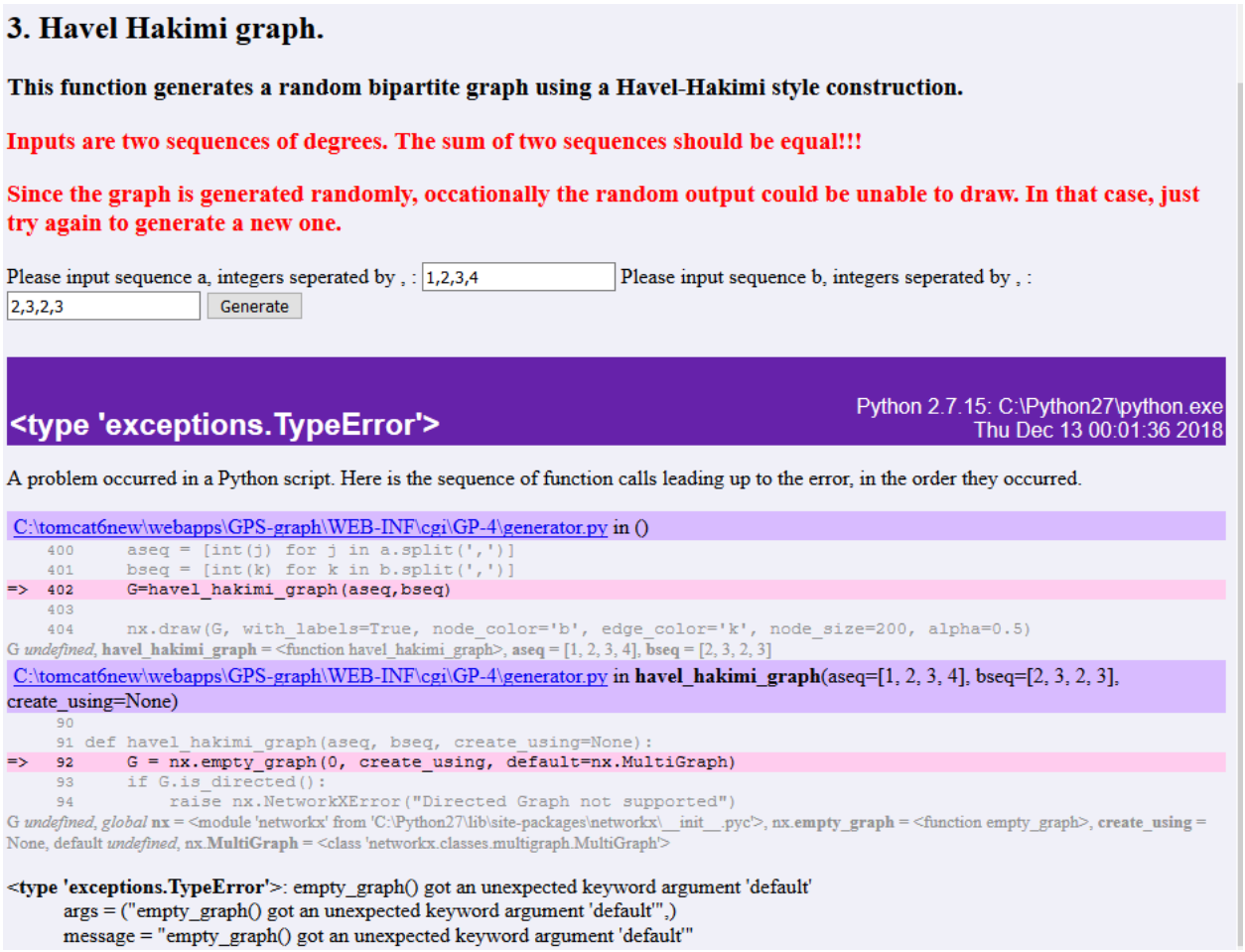



*Figure 13: Havel Hakimi graph error*

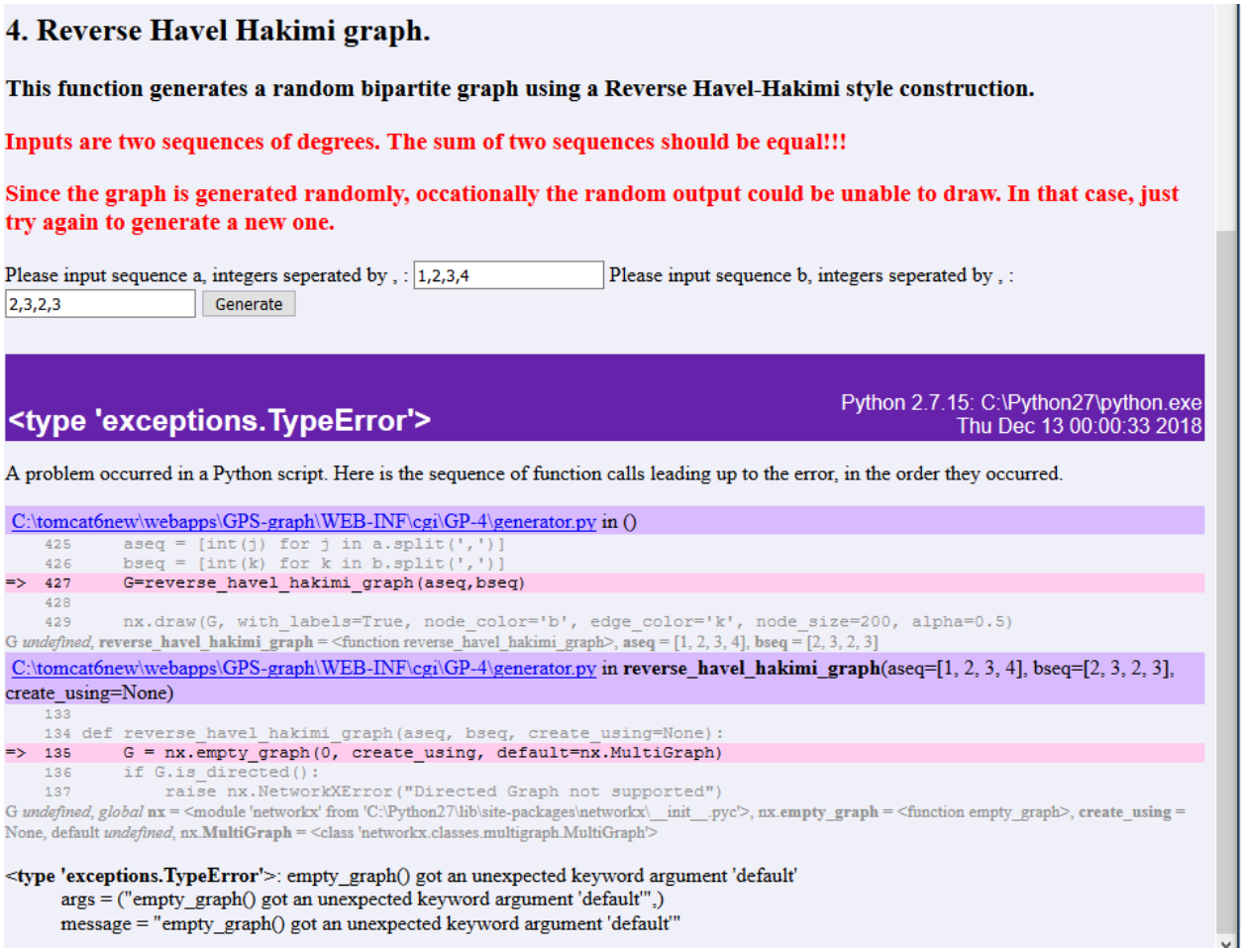



*Figure 14: Reverse Havel Hakimi graph error*

## 8. Gnmk random graph.

**This function generates a random bipartite graph given n, m, k.**

**Inputs are 3 integers n, m, and k, the number of edges**

**The number of edges, k, should not be larger than n*m/2, and should not be too small for two parts of the bipartite graph to connect!!!**

Please input n, the number of nodes in the first set: 8 Please input m, the number of nodes in the seconde set: 8 Please input k, the number of edges: 35 Generate

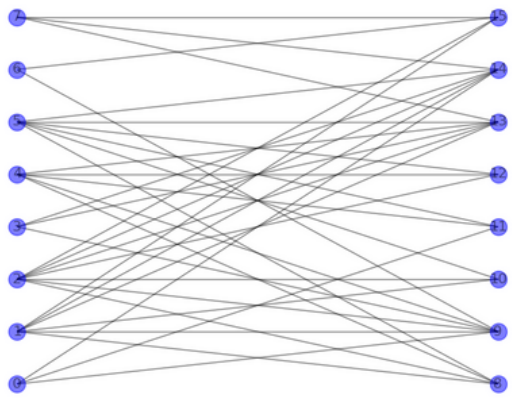


*Figure 15: Gnmk random graph working*

# Redundancy

Go back to index.

## 1. Redundancy.

This function computes the node redundancy coefficients for the nodes in the bipartite graph G

Please upload a bipartite graph in CSV format: Browse... No file selected.

Make sure the graph is uploaded, then click to compute!!!

**The result of function 1 is:**

**Cannot compute redundancy coefficient for a node that has fewer than two neighbors.**

**Please input a new bipartite graph!**

None

*Figure 16: Redundancy function unable to compute due to necessary*

# X. Summary and Discussion

In the end, I don't agree with the project as I feel I did not really learn much or anything that would stick as it was all ephemeral. I learned a little into webpage design and was reminded on how to work with HTML and CSS and JavaScript. I was able to create a, in my opinion, great webpage. I was really satisfied with my product and in the end I am happy that my webpage looks so nice. Despite how nice it looks and how well it works with the given csv file, I forgot to account for different csv files since uploading to csv server was something I forgot about. Because of this, my webpage isn't very inclusive of other csv files. I learned some ephemeral terms related to graphs such as dominators, dominating sets, Eulerian, and efficiency. In these terms I learned some of the real-world application for these terms and the only one that made sense to me was efficiency. I also disagree with the timing aspect of this project in relation to the small amount of time given to this project and not to more relevant topics such as in class material. Overall this project seems quite boring and just tedious not very much intellectually stimulating material. I would much rather look at data structures and algorithms in the industry or how software engineering is used in the industry as I believe that would be more relevant and important. In the end it was cool to be able to make a webpage but overall I think it was kind of a waste of time in that nothing new was really learned.

# XI. References

"Dominance¶." *NetworkX - NetworkX*, networkx.github.io/documentation/stable/reference/algorithms/dominance.html.

"Dominating Sets¶." *NetworkX - NetworkX*, networkx.github.io/documentation/stable/reference/algorithms/dominating.html.

"Dominator (Graph Theory)." *Wikipedia*, Wikimedia Foundation, 12 Apr. 2018, en.wikipedia.org/wiki/Dominator_(graph_theory).

"Efficiency." *Wikipedia*, Wikimedia Foundation, 17 Oct. 2018, en.wikipedia.org/wiki/Efficiency.

"Efficiency¶." *NetworkX - NetworkX*, networkx.github.io/documentation/stable/reference/algorithms/efficiency.html.

"Eulerian Path." *Wikipedia*, Wikimedia Foundation, 29 Nov. 2018, en.wikipedia.org/wiki/Eulerian_path.

"Eulerian¶." *NetworkX - NetworkX*, networkx.github.io/documentation/stable/reference/algorithms/euler.html.