Approval Cover Page

Curriculum Graph Visualizer

Dana Oira Toribio

Dr. Kevin Wortman, Advisor

Dr. Kent Palmer, Reviewer

Department of Computer Science

California State University, Fullerton

May 6, 2016

Contents

[1 Introduction 5](#_Toc450300634)

[1.1 Problem Description 5](#_Toc450300635)

[1.2 Project Objectives 7](#_Toc450300636)

[1.3 Development Environment 8](#_Toc450300637)

[1.4 Operational Environment 8](#_Toc450300638)

[2 Requirements Description 9](#_Toc450300639)

[3 Design Description 10](#_Toc450300640)

[4 Implementation 10](#_Toc450300641)

[5 Test and Integration 11](#_Toc450300642)

[6 Installation Instructions 14](#_Toc450300643)

[7 Operating Instructions 14](#_Toc450300644)

[8 Recommendations for Enhancements 16](#_Toc450300645)

[7 Acknowledgement 16](#_Toc450300646)

[10 Biliography 17](#_Toc450300647)

List of Figures

[**Figure 1.** Undergraduate Computer Science Curriculum Graph 5](#_Toc450308822)

[**Figure 2.** Translation between studyplan.txt to studyplan.py. 10](#_Toc450308823)

[**Figure 3.** CGV download on GitHub 15](#_Toc450308824)

[**Figure 4.** Run command prompt 15](#_Toc450308825)

[**Figure 5.** Successful command prompt launch 16](#_Toc450308826)

[**Figure 6.** Successful directory change 16](#_Toc450308827)

[**Figure 7.** Run command for CGV in console 17](#_Toc450308828)

List of Tables

[**Table 1.** Computer Science Core Curriculum 4](#_Toc450308817)

[**Table 2.** Project Schedule 12](#_Toc450308818)

***Abstract* – The graph academic curricula is complex and can be difficult to visualize it. This program demonstrates the ability to take a set of course curriculum data, user inputs to provide study plan suggestions and convert it to a graph visualization using Python GraphViz.**

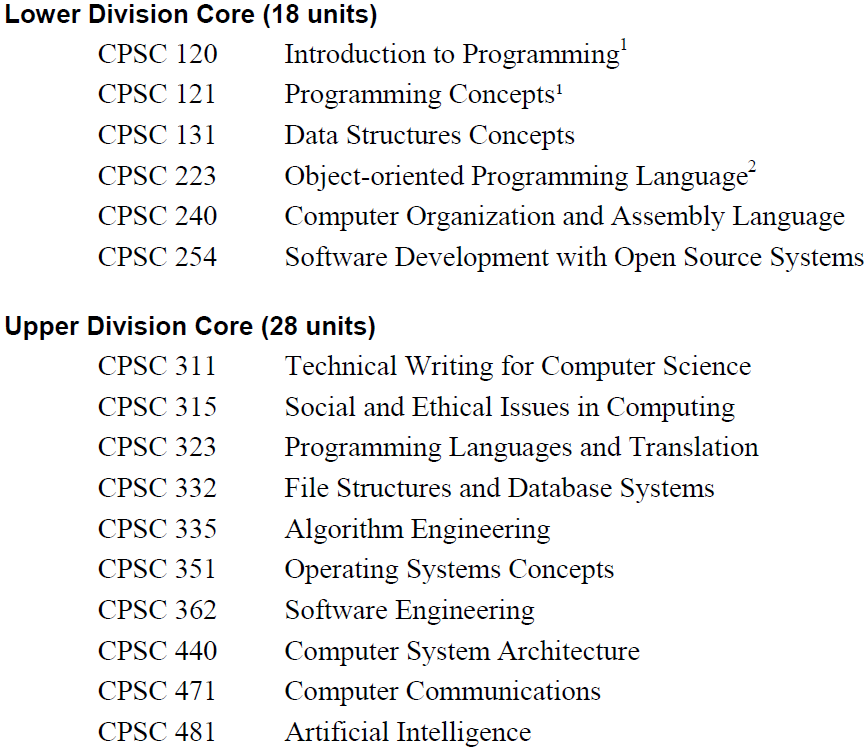
***Keywords* – interactive graph drawing, graph programming, graph visualization, directed graphs**

# 1 Introduction

## 1.1 Problem Description

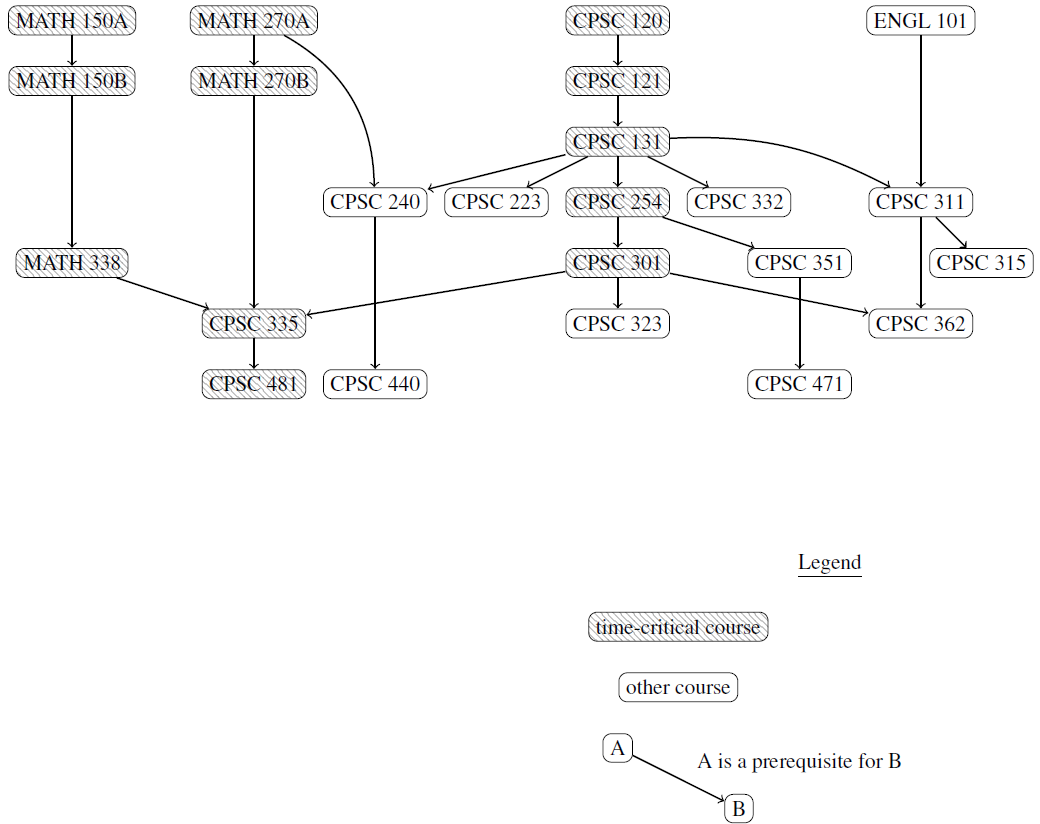
In academia, a course curriculum is available which lists courses and electives required for completion of a degree program. Often times, a course curriculum is simply given as a list of courses.

**Table 1.** Computer Science Core Curriculum



Shown in **Table 1** is the core curriculum for the Computer Science major at California State University Fullerton from the 2010 student handbook. What this format does not clarify with is visualizing and understanding the relationship between all the courses. Underlying information like prerequisites, co-requisites and the progression of the curriculum in a semester-to-semester basis is difficult to visualize from a simple list of courses.

**Figure 1.** Undergraduate Computer Science Curriculum Graph



**Figure 1** shows a graphical representation of the course curriculum in Figure 1. This figure is also known as a curriculum graph. The curriculum graph is able to show course prerequisites and co-requisites with a legend key to add clarity to the relationship between all the courses. Paired with the course list from Figure 1, the curriculum graph gives a better understanding of the course progression.

With the contrast of the course listing and the curriculum graph, it's evident that the graph of courses and their prerequisites is complex, and students and faculty can both struggle to visualize it.

## 1.2 Project Objectives

The objective of this project is to create a curriculum graph which will take the input of curriculum graph data of courses names, prerequisites and locations to output a graph drawing. The graph drawing will organize its output to minimize overlapping edges, organize the graph by suggesting courses by semester, the courses in the graph by tracks and to indicate completed courses.

The final results of the project will include an (1) Algorithm Design Document which detail the rationale behind the algorithm selection, included in this document, and (2) a command line application that is able to perform all the tasks aforementioned in the Objectives:

1. *Create a graph drawing.* The user will input course information which the program will use to generate a graph drawing that is readable. The graph drawing will include the following attributes below.
2. *Organize graph into tracks.* Since there are several core tracks as well as elective tracks within the major, the program will aim to generate the curriculum graph’s output to be organized by these tracks. In general, the core tracks will be categorized based on its root node. Courses in the same track will be organized within the same column.
3. *Minimize overlapping edges.* Decreasing overlapping edges is a best practice for graph drawings. Due to the nature of course organization into strata and tracks, overlapping edges will likely be unavoidable. This project will prioritize strata and track organization before edge minimization.
4. *Indicate complete courses.* Indicating course completion may be done by coloring the completed courses’ nodes in a different color. This feature will be helpful with tracking completed courses and tracking which courses are to be done.

The main goal of the final demonstration of the project is to efficiently show the program’s full range of functionality.

## 1.3 Development Environment

Hardware

The hardware used to develop the project includes the following:

1. Central Processing Unit: Intel Core i5-3570K 3.4GHz
2. Random Access Memory: 8GB
3. Graphical Card: PNY NVIDIA GeForce 560 Ti Fermi 1GB
4. Operating System: Windows 10

Software

The software used in the development of this program includes the following:

1. Programming Languages: Python 3.5.1 (v3.5.1:37a07cee5696), DOTS
2. Graph Visualization: GraphViz 2.38
3. Python GraphViz Package: GraphViz 0.4.10
4. Editor: Sublime Text Build 3013

## 1.4 Operational Environment

Hardware

The hardware needed to run this program includes:

1. Central Processing Unit: 1.5GHz or faster processor
2. Random Access Memory: 1GB
3. Screen Resolution: 1024x768 pixels
4. Operating Systems: Windows 7/8/10, Linux, Mac OS X v10.9/10.10/10.11

Software

The software needed to run this program includes:

1. Programming Languages: Python 3
2. Graph Visualization: GraphViz 2
3. Graph Viewer: Adobe Acrobat Reader

# 2 Requirements Description

2.1 External Functions

1. Educational institution’s curriculum specifications.
   1. Unit limitations
   2. Prerequisite requirements
2. GraphViz and Python GraphViz formatting requirements.

2.2 External Interfaces

User Interfaces

The user interfaces shall follow in the following requirements:

1. The program shall be executed through the command line.

In the interest of time, a command line execution for the program suffices to the demonstration of the projects functions.

Hardware Interfaces

The hardware interfaces shall follow the following requirements:

1. The program shall respond to keyboard input.
2. The graphical card shall generate a graph to the screen.

No actual hardware was created in the development of this process. To simplify the project, the only hardware interaction required keyboard and mouse navigation to access the command line prompt.

Software Interfaces

The CGV utilizes several external interfaces:

1. *Programmatic Interface:* The CGV shall access functions of the external software through a library (Python GraphViz 0.4.10, Python os module, Python re module, Python sys module).
2. *Graph Drawing Interface:* The CGV shall access graph drawing capabilities through a visualization software (GraphViz 2.38).

# 3 Design Description

3.1 Architecture

In progress.

3.2 Internal Functions

There are two main files involved in the curriculum input (CIF) to graph generation:

1. *cgv.py, Curriculum Graph Visualizer:* This file guides the main events for the creation of the Study Plan list.
   1. Input: curriculum.txt
   2. Output: studyplan.txt
2. *cgc.py, Curriculum Graph Converter:* This file does the translation functions from the CIF to generate a Python GraphViz executable code
   1. *Input:* studyplan.txt
   2. *Output:* studyplan.py

The following files are generated by the user:

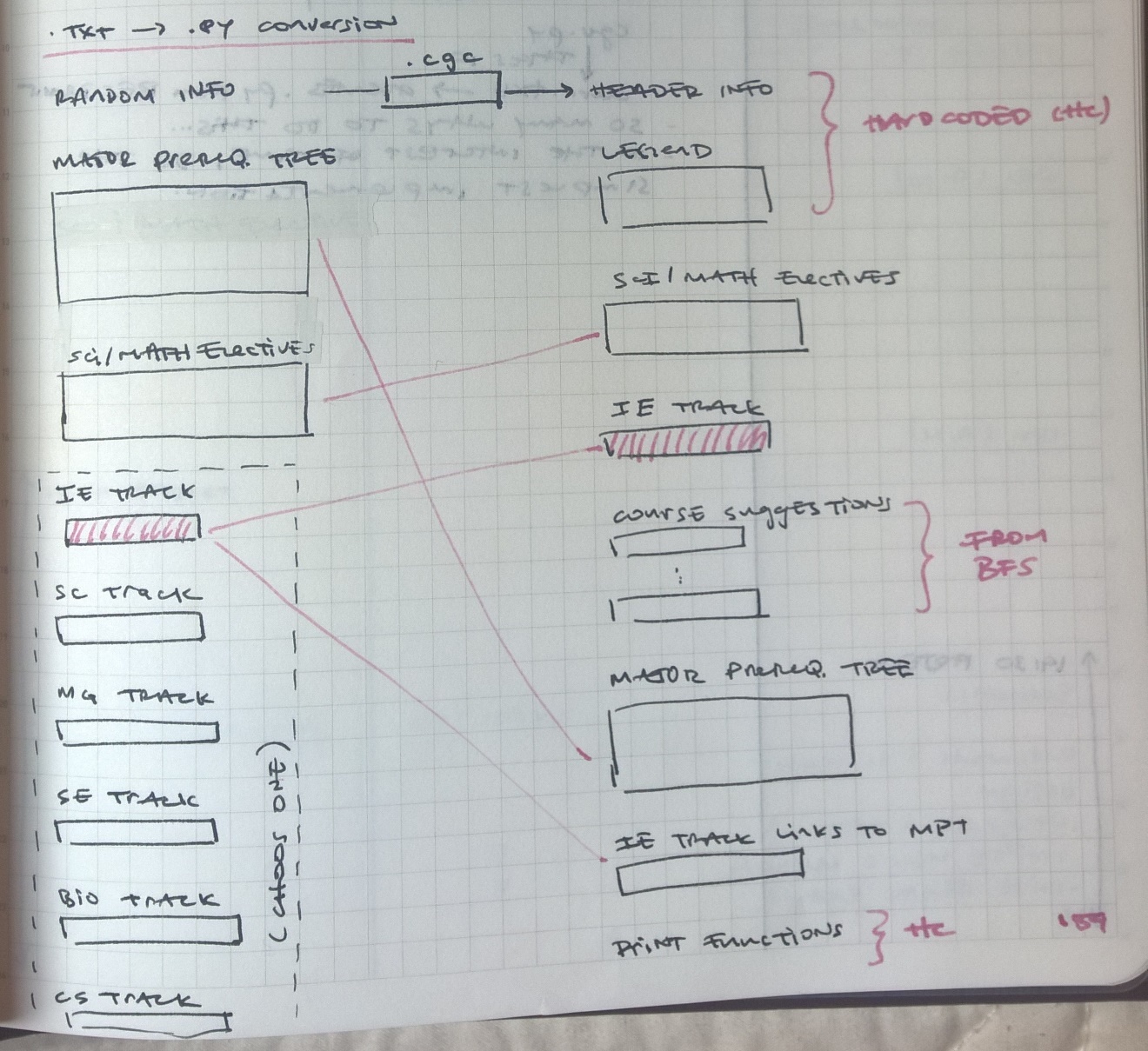
1. *curriculum.txt:* The curriculum input file (CIF) that is has a major’s entire curriculum, prerequisites and track electives.

The following describes the files generated by cgv.py and cgc.py:

1. *studyplan.txt:* An updated CIF which includes only required courses and elective track courses specified by the user.
2. *studyplan.py:* A translation of studyplan.txt to Python GraphViz, which launches the graph rendering.

Draft photo. To be updated with a computer graphic.

**Figure 2.** Translation between studyplan.txt to studyplan.py.



3.3 Internal Interfaces

To elaborate.

* Python GraphViz
* Regular Expressions Module
* OS Module
* System Module

# 4 Implementation

4.1 Organization of Source File Structure

**cgc.py, Curriculum Graph Visualizer**

The source file structure is located in cgv.py.

1. Read in Curriculum Input File (CIF).
2. Find elective tracks.
3. Print electives to console.
4. Read in one (1) user input for elective track.
5. Read courses in user’s study plan, which include:
   1. Required courses
   2. Elective track courses
6. Writes courses to studyplan.txt file.
7. Create study plan tree of user’s selected track.
8. Console prints list of courses in study plan.
9. Take in user input for completed courses.
10. Update study plan tree with completed courses,
11. Write suggestions to studyplan.txt.
12. Run cgc.py

**cgc.py, Curriculum Graph Converter**

The source file structure for cgc.py:

1. Open studyplan.txt to read.
2. Open study plan.py to write.
3. Read studyplan.txt line by line.
4. Update the following attributes with the Python GraphViz translation.
   1. req\_electives
   2. trk\_electives
   3. completed\_courses
   4. suggested\_courses
   5. core\_courses
   6. elective\_prereqs
5. Write resulting translation to studyplan.py.
6. Run studyplan.py

4.2 Reference List of Files

1. *curriculum.txt:* Curriculum Input File (CIF)
2. *cgv.py:* Curriculum Graph Visualizer
3. *cgc.py:* Curriculum Graph Converter
4. *studyplan.txt:* Modified CIF
5. *studyplan.py:* Translated studyplan.txt to Python GraphViz

# 5 Test and Integration

Please advise if this detail is too verbose or unnecessary for the final draft.

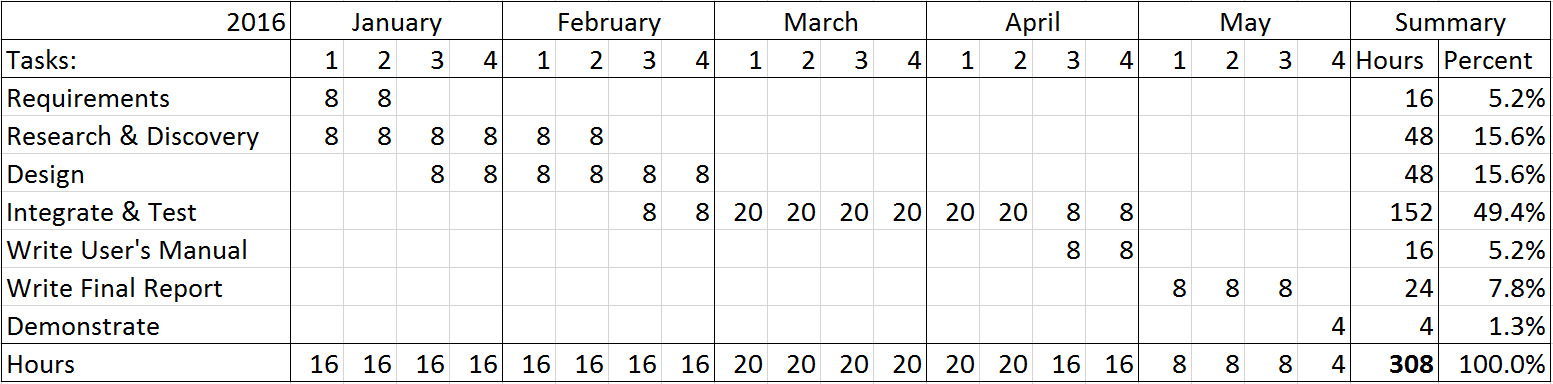
5.1 Plan

Since the project was created by one person, there were no formal daily standup meetings. Daily progress was recorded internally as reference. Testing was be done individually and bug fixes were part of the development phase.

The integration phase resulted in the deliverable of the completed project as a command line executable application.

**Table 2** shows the proposed project schedule. The project will begin in early January through the duration of the semester, allowing approximately 308 hours of work over the course of 20 weeks.

**Table 2.** Project Schedule



There is some overlap with the tasks in that the next task may be worked on as the preceding task is reaching completion. To comply with the SCRUM time-boxing of two-week long sprints, a new task typically starts every two (2) weeks aside from the Integration portion where the bulk of development will occur. There will be a total of five (5) iterations, beginning the first week of every month. Each iteration will have two (2) sprints each, giving a total of twenty (20) sprints for the entire duration of the project.

Requirements gathering was given 16 hours since many of the requirements having been gathered before the time of writing the proposal. If less time is needed for requirements gathering, the extra time can be used for research and discovery of algorithm patterns and testing since the Algorithm Phase is expected to be the most difficult portion of the project.

The project’s development were conducted in two phases. Phase 1 included the Algorithm Phase, to determine which graph drawing algorithms will be implemented into the program. The second phase will be the Development Phase, in which the development of the program occured.

Phase I – Algorithm

The Algorithm Phase for the first phase of the project includes research activities to decide which algorithms will be chosen for the implementation of the graph drawings. The two activities in the Algorithm Phase includes Algorithm Discovery and Algorithm Testing.

*Algorithm Discovery.* The Algorithm Discovery portion of this activity will be continued research on graphing algorithms.

*Algorithm Testing*. This activity will include testing of graphing algorithms to determine their suitability against the project’s objectives. Possible algorithms that can be tested can be graphing algorithms found during Algorithm Discovery. Algorithms can also be created from scratch if graphing algorithms found in the Algorithm Discovery activity are not sufficient.

The resulting output of this Algorithm Phase will be (1) selected algorithms to be implemented in Phase II and (2) an Algorithm Design Document which will include pseudocode, an algorithm snippet from the project’s source and a brief description of the rationale behind the algorithm selection. The Algorithm Design Document is found in section **5.2 Results**.

Phase II – Development

The Development Phase follows the Algorithm Phase. This phase includes common software engineering development activities including software design, software development and quality assurance testing.

With the time constraint limiting the entire project to the duration of a semester, the development process will loosely follow the Agile methodology using SCRUM. The project was time-boxed into 2-week sprints beginning in January. Each sprint begins with a meeting with the project advisor to discuss the project progress and if the sprint’s goals are reasonably appointed. The following meeting followed up on the project’s progress and demonstrated any working samples from that sprint’s development.

Testing

Testing was done prior to development and occurred during the research phase. In order to understand the workflow of how the project would be executed, Ideal Results were created as part of the testing data. The Ideal Results were used to prototype the expected output of a proper graph based on several of the computer science elective tracks.

Test cases are found in the test folder. Viewing the Python (.py) test case file in a programming editor will give the ideal Python GraphViz code needed to generate the ideal graph output. Running the Python file will generate the ideal graph output.

5.2 Results

Phase I – Algorithm

Algorithm Discovery

In Progress

* Tree Implementation
* Hash Table for Tree Update
* Breadth First Search

Algorithm Testing

In Progress

Algorithm Design Document

In Progress

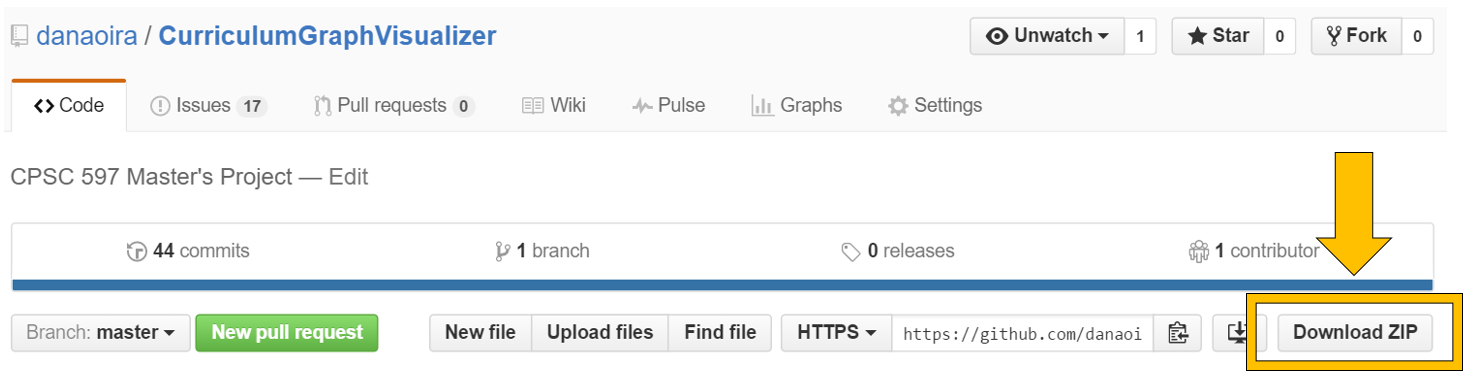
Phase II – Development

In Progress

# 6 Installation Instructions

1. Download CGV package from GitHub.

**Figure 3.** CGV download on GitHub



1. Extract CGV files into any folder.
2. Install Python.
3. Install GraphViz.

# 7 Operating Instructions

Creating a Curriculum Input File (CIF)

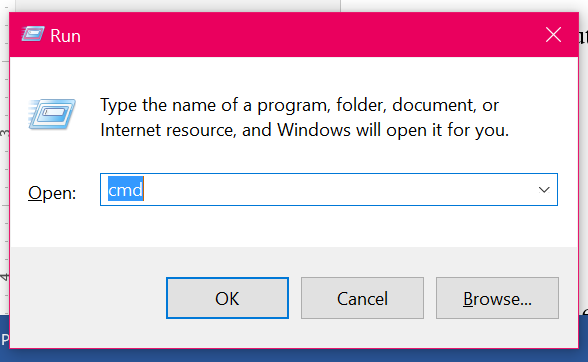
In the CGV folder, there will be a file called ‘sample\_curriculum.txt’. This file will detail the formatting instructions a user must use in order for the CGV to work.

The CGV can read in any plain text file format as the CIF document.

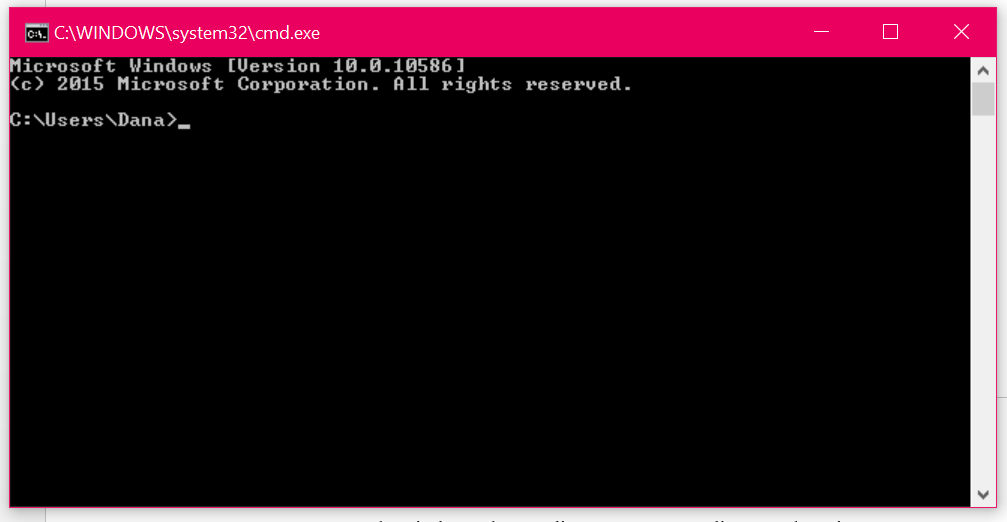
Execute CGV to Generate a Study Plan

1. Create a CIF.
2. Open a command prompt.
   1. Press Windows key + R and type “cmd”.

**Figure 4.** Run command prompt

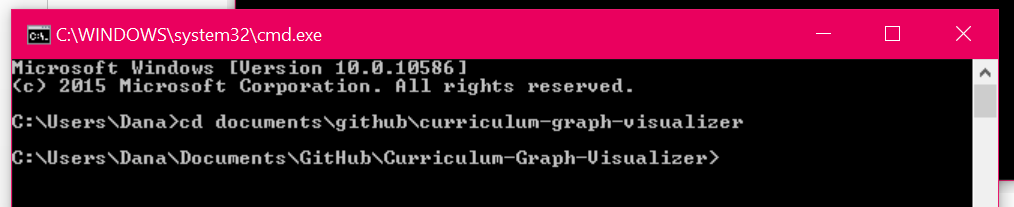


**Figure 5.** Successful command prompt launch



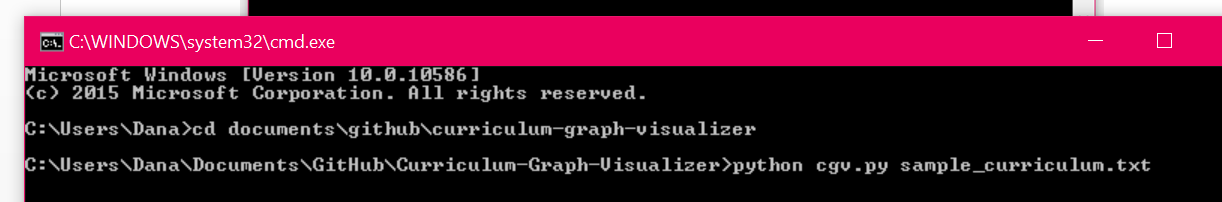
1. In console window, change directory to CGV directory location.
   1. Input string: cd [DIRECTORY PATH]
   2. Press Enter.

**Figure 6.** Successful directory change



1. Type CGV execution command in console.
   1. Format: python cgv.py [CIF file]
      1. python: Command to run the file using Python.
      2. cgv.py: Runs the CGV file.
      3. [CIF File]: The input file for the course curriculum.
         1. For convenience, save CIF is in the same directory as cgv.py.

**Figure 7.** Run command for CGV in console



1. Select one (1) elective track.
2. Input completed courses.
   1. Acceptable inputs:
      1. *Comma separated list:* CPSC 121, CPSC 131, MATH 120
      2. *Single-line input*
3. Press Enter twice to generate the study plan.

# 8 Recommendations for Enhancements

To elaborate.

* Graphical User Interface
* Web Application
* File upload for CIF
* CIF tutorial GUI

# 7 Acknowledgement

I would like to thank my mentors, Dr. Kevin Wortman (Advisor) and Dr. Kent Palmer (Reviewer), for the guidance and inspiration for the proposal portion of this project as well as the continued collaboration during the project’s creation in the upcoming semester.

# 10 Biliography

1. Agrawal, V.K.; Earnest, J.; Gupta, S.K.; Tegar, J.P.; Mathew, S.S., "Outcome based engineering diploma curriculum - 2012 Gujarat experiment," in *Frontiers in Education Conference, 2013 IEEE* , vol., no., pp.1864-1870, 23-26 Oct. 2013
2. Azlan, A.; Hussin, N.M., "Implementing graph coloring heuristic in construction phase of curriculum-based course timetabling problem," in *Computers & Informatics (ISCI), 2013 IEEE Symposium on* , vol., no., pp.25-29, 7-9 April 2013
3. Gansner, E.R.; Koutsofios E.; North S.C.; Vo K., “A Technique for Drawing Directed Graphs,” in *IEEE Transactions on Software Engineering, 1993, vol. 19, no. 3, pp. 214 -230, March 1993*
4. Gestwicki, P., "Work in progress - curriculum visualization," in *Frontiers in Education Conference, 2008. FIE 2008. 38th Annual* , vol., no., pp.T3E-13-T3E-14, 22-25 Oct. 2008
5. Hosobe, H., "Analysis of a high-dimensional approach to interactive graph drawing," in *Visualization, 2007. APVIS '07. 2007 6th International Asia-Pacific Symposium on* , vol., no., pp.93-96, 5-7 Feb. 2007
6. Kabicher, S.; Motschnig-Pitrik, R., "Coordinating Curriculum Implementation Using Wiki-supported Graph Visualization," in *Advanced Learning Technologies, 2009. ICALT 2009. Ninth IEEE International Conference* on , vol., no., pp.742-743, 15-17 July 2009
7. McCreary, C.L.; Chapman, R.O.; Shieh, F.-S., "Using graph parsing for automatic graph drawing," in *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on* , vol.28, no.5, pp.545-561, Sep 1998
8. Rodgers, P.J., "A graph rewriting programming language for graph drawing," in Visual Languages, 1998. Proceedings. 1998 IEEE Symposium on , vol., no., pp.32-39, 1-4 Sep 1998
9. Rusu, A.; Crowell, A.; Petzinger, Bryan; Fabian, A., "PieVis: Interactive Graph Visualization Using a Rings-Based Tree Drawing Algorithm for Children and Crust Display for Parents," in *Information Visualisation (IV), 2011 15th International Conference on*, vol., no., pp.465-470, 13-15 July 2011
10. Slim, A.; Heileman, G.L.; Kozlick, J.; Abdallah, C.T., "Employing Markov Networks on Curriculum Graphs to Predict Student Performance," in *Machine Learning and Applications (ICMLA), 2014 13th International Conference on* , vol., no., pp.415-418, 3-6 Dec. 2014