Curriculum Graph Visualizer

Dana Oira Toribio

Dr. Kevin Wortman, Advisor

Dr. Kent Palmer, Reviewer

Department of Computer Science

California State University, Fullerton

November 10, 2015

Contents

List of Figures

List of Tables

1 Introduction

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

Lower Division Core (18 units)	
CPSC 120	Introduction to Programming[1]
CPSC 121	Programming Concepts[1]
CPSC 131	Data Structures Concepts
CPSC 223	Object-oriented Programming Language[2]
CPSC 240	Computer Organization and Assembly Language
CPSC 254	Software Development with Open Source Systems
Upper Division Core (28 units)	
CPSC 311	Technical Writing for Computer Science
CPSC 315	Social and Ethical Issues in Computing
CPSC 323	Programming Languages and Translation
CPSC 332	File Structures and Database Systems
CPSC 335	Algorithm Engineering
CPSC 351	Operating Systems Concepts
CPSC 362	Software Engineering
CPSC 440	Computer System Architecture
CPSC 471	Computer Communications
CPSC 481	Artificial Intelligence

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

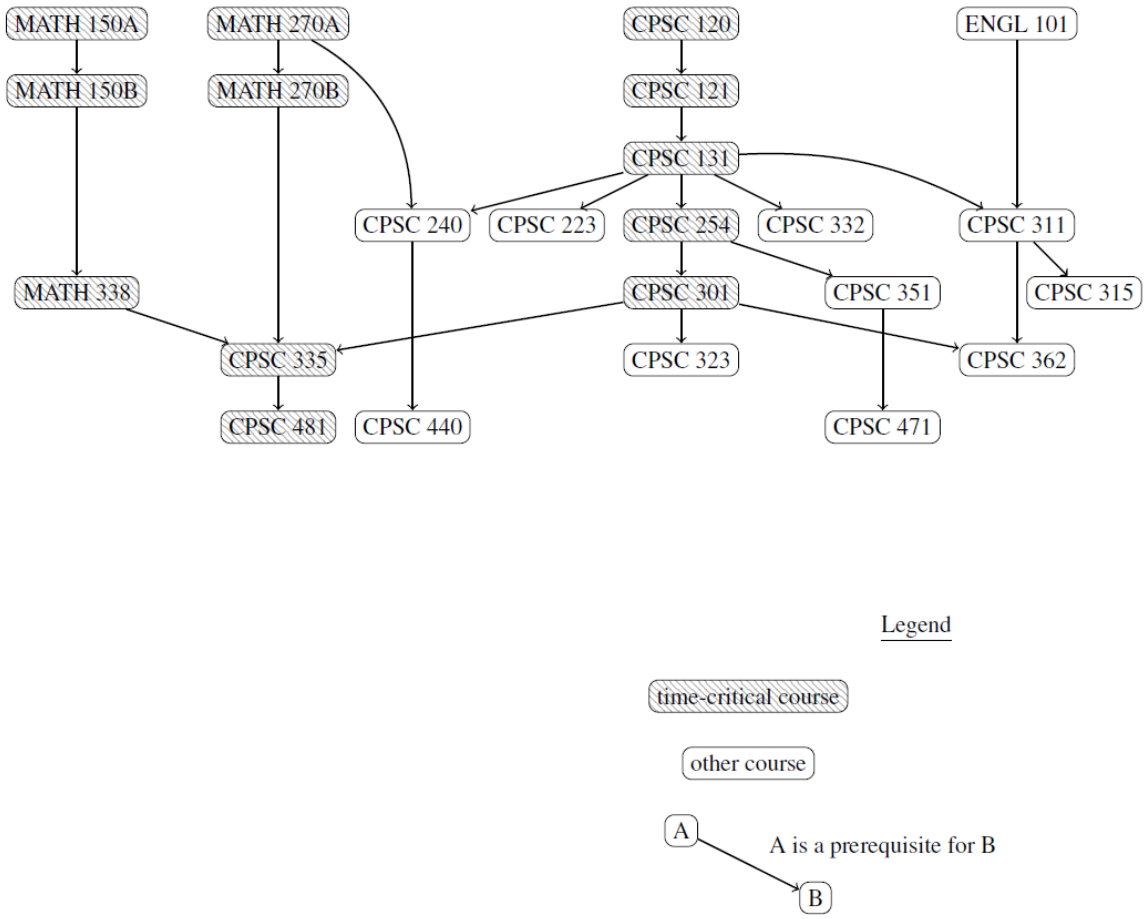


Figure 1 shows a graphical representation of the course curriculum in Figure 1. This figure is also known as a curriculum graph. The curriculum graph is able to show course prerequisites and co-requisites with a legend key to add clarity to the relationship between all the courses. Paired with the course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

For development, there are several issues related to graph visualization for a curriculum graph which makes it a challenging endeavor:

Table 2. Problem Domain for Curriculum Graph Visualizer

Area of Expertise	**Research Required**
Fundamental Graph Algorithms	Hierarchical, planar, general
Graph Drawing	Requires computationally complete languages
Human Computer Interfaces (HCI)	Functionality, ease of use, cross-compatibility

Table 2 presents the problem domain. The areas of expertise this issue covers includes fundamental graph algorithms, graph drawing, human computer interfaces and potentially AI planning algorithms.

Fundamental graphing algorithms. Several algorithms are widely used for graph programming problems are hierarchical, planar or general graphing algorithms.

Graph drawing. Graph drawing is a NP-complete problem which requires computationally complete programming languages.

Human Computer Interfaces (HCI). Functionality, ease-of-use and cross-compatibility are important factors for the end-user level with HCI. Without these considerations in developing a program for curriculum graph visualization, the system will not be usable to the end-user.

Table 3. Current Solutions for Graph Programming

Database Models	**Graph Databases**	**Digital Formats**
• Relational • Object-Oriented • Graphical	• Neo4J • MongoDB • TinkerPop3	• DOT • NetworkX • Gremlin • DOODLE

Database Models. Database models available that have been used for graph programming are relational database model, object-oriented database model and graphical database model. *Relational database model* is the most widely-used database model that is applied in various industries outside of academia and computing like business, accounting, marketing and many others. RDBM has known limitations for graph programming due to its extensive use of lists and keys between tables. *Object-oriented database model* has known applications for graph programming like DOODLE. *Graphical database model* is the most recent database model specially made for graph programming. This model uses nodes, properties and edges rather than tables. The most unique part of the graph model is that edges can be identified and labeled, which is useful with creating directed graphs.

Graph Databases. Current database models available for graph programming include Neo4J, MongoDB, and TinkerPop3. *Neo4J* is a widely-used open-source property graph model that is implemented in Java. *MongoDB* uses NoSQL, a document-

oriented database that is implemented in Python and is usable in virtual web applications. Lastly, *TinkerPop3* is a graph database that can be implemented in a wide array of web languages like Python, JavaScript, Java, PHP as well as a functional programming language, Scala. TinkerPop3 utilizes Gremlin, a graphical programming language.

Digital Formats. Current methods digital formats include the following visual languages of DOT programming language, NetworkX, Gremlin and DOODLE. *DOT programming language* is a plain text graph description language. The DOT language defines a graph, but needs external programs to render the DOT language.

Figure 2. Binary Search Tree visualized with DOT and GraphViz

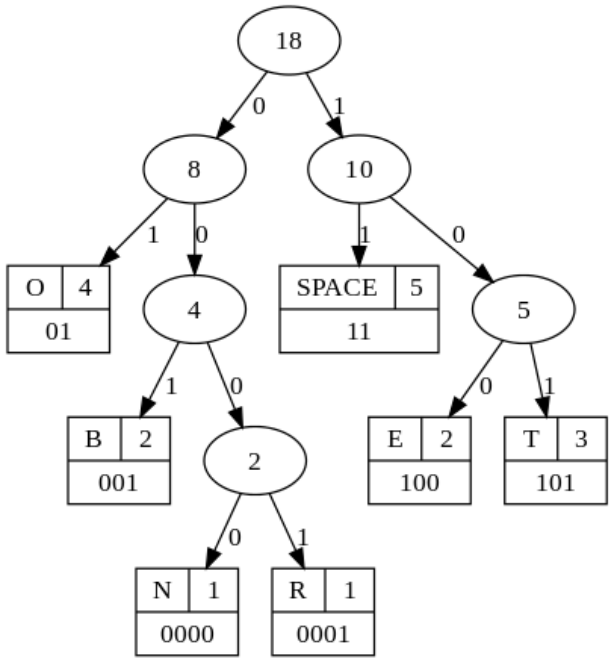


Figure 2 shows a binary search tree generated with GraphViz using the DOT programming language. NetworkX was created as a Python library for studying graphs and networks.

Figure 3. Directed Graph visualized with NetworkX

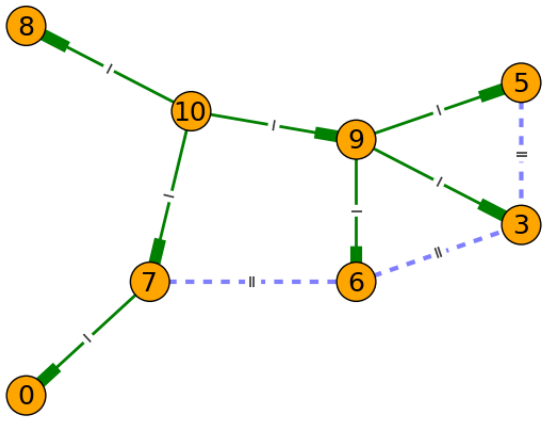
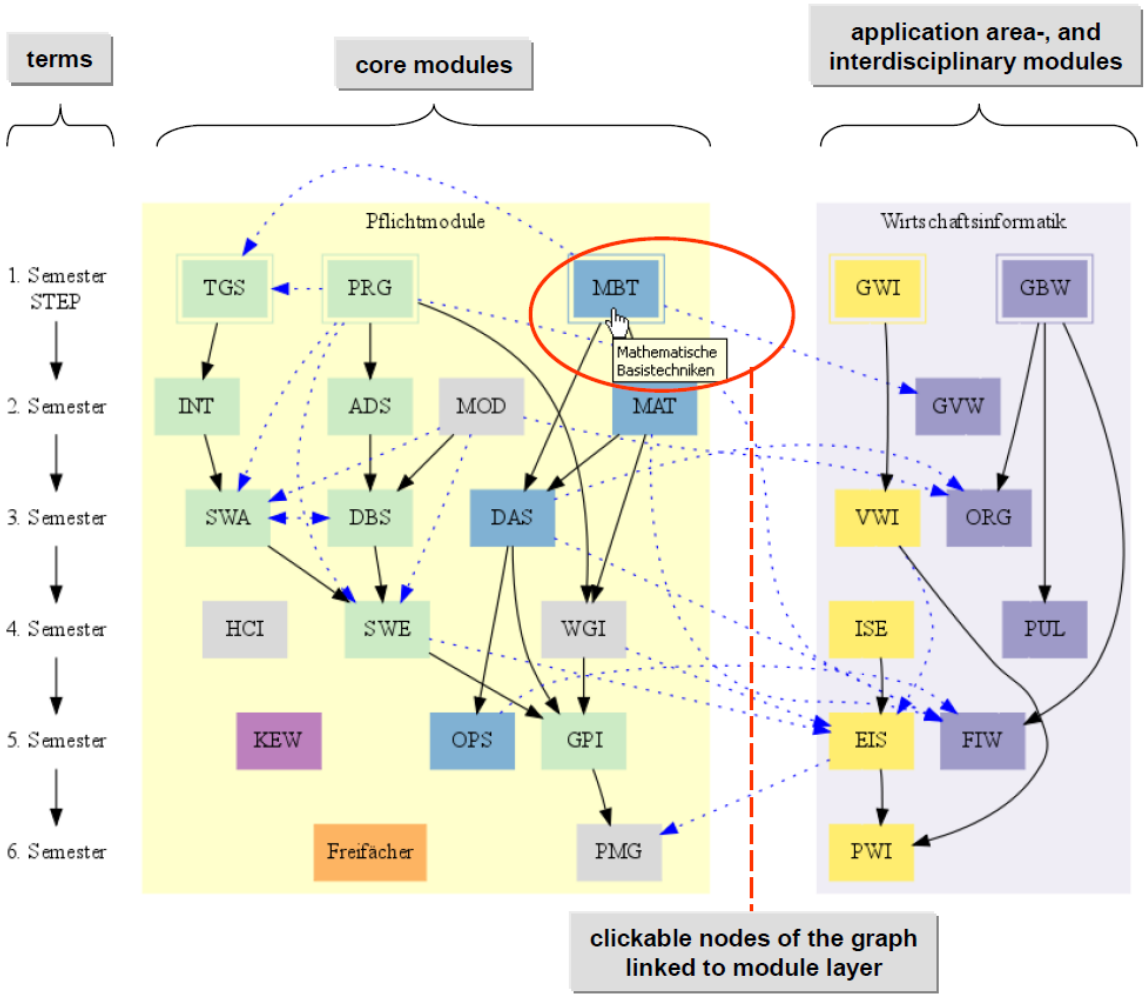


Figure 3 shows a directed graph using NetworkX with directed edges represented using a solid rectangle. *Gremlin* is a graph traversal language that can also be used as a virtual machine. Figure 6 shows a graph created using Gremlin language and visually formatted in GraphML. Gremlin is also used in the TinkerPop3 database. The last digital format is *DOODLE*, which is utilizes object-oriented databases.

Several institutions have been working on modeling a curriculum as a graph as this subject is a great point of interest in academia. The following pages will present two examples of similar solutions for the curriculum graph visualization problem.

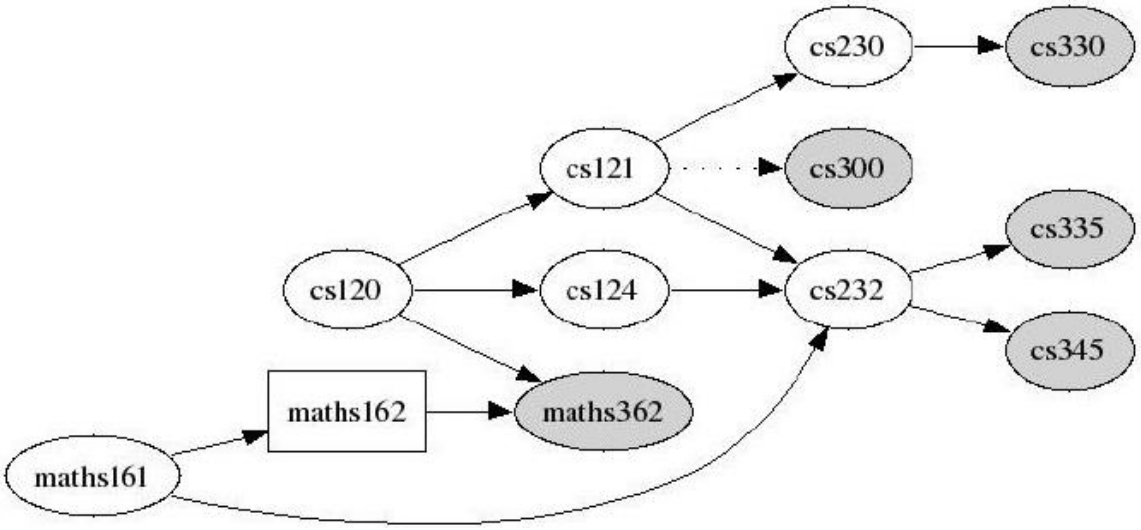
Figure 4. ActiveCC Visualization of Curriculum Structure[6]



At the University of Vienna, a program called ActiveCC uses Wiki-supported graph visualization for curriculum graph visualization [6]. Their graphing process includes three layers for modeling: curriculum layer, module layer and course layer.

Their research and application may have relevance to the Curriculum Graph Visualizer project as ActiveCC generates directed graphs and organization by strata on a semester-to-semester basis as seen in **Figure 4**.

Figure 5. CurricVis visualization of the Computer Science Minor[4]



At Ball University in Indiana, a program called CurricVis presents a prototype used to generate visualizations of Computer Science curricula [4]. It is implemented in SWI-Prolog with a three-tier software architecture:

1. Knowledge Base
2. Visualization Generation
3. User Interface

Their architecture is of relevance to the Curriculum Graph Visualizer since the steps to develop their prototype are similar to the objectives set out for this project.

A potential solution to solve this problem is to model the curriculum as a directed graph where each course is a vertex and each prerequisite requirement or co-requisite is a directed edge. Creating a system that is widely available would bring considerations of making a web application that can be used on a desktop, mobile or tablet environment.

2 Objectives

The basic requirements for the proposed system is to (1) model the curriculum as a directed graph, (2) implement the curriculum graph as an online web application to allow teachers and students to access it online. The necessary knowledge for the proposed system includes fundamental graph algorithms, graph drawing and human computer interfaces (HCI).

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 into strata by suggesting courses by semester, the courses in the graph by tracks and to indicate completed courses. The curriculum graph drawing will be made into a web application to be submitted as a final deliverable.

3 Activities

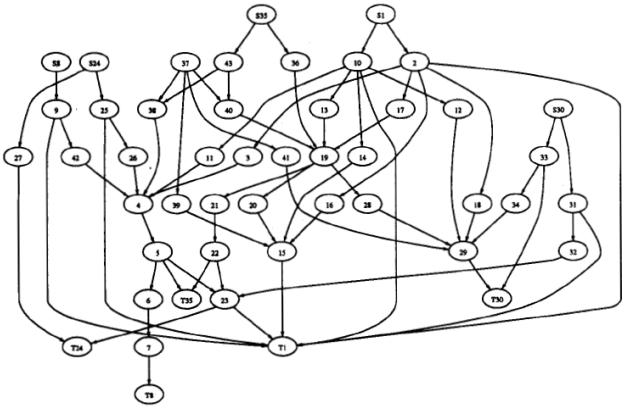
The project's activities will be conducted in two phases. Phase 1 will include the Algorithm Phase, to determine which graph drawing algorithms will be implemented into the program. The second phase will be the Development Phase, in which the development of the program will occur.

3.1 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. Some points of interest for graphing algorithms include graphing techniques by Gansner et al's research on directed graph generation and optimal ranking [3].

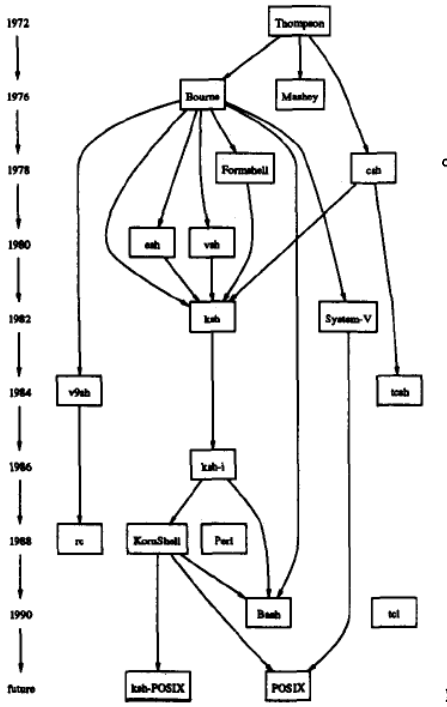
Figure 6. Gansner et al Digraph Generation[3]



(a)

```
digraph world_dynamics {
        size="6,6";
        S8 -> 9; S24 -> 27; S24 -> 25; S1 -> 10; S1 -> 2; S35 -> 36;
        S35 -> 43; S30 -> 31; S30 -> 33; 9 -> 42; 9 -> T1; 25 -> T1;
        25 -> 26; 27 -> T24; 2 -> 3; 2 -> 16; 2 -> 17; 2 -> T1; 2 -> 18;
        10 -> 11; 10 -> 14; 10 -> T1; 10 -> 13; 10 -> 12;
        31 -> T1; 31 -> 32; 33 -> T30; 33 -> 34; 42 -> 4; 26 -> 4;
        3 -> 4; 16 -> 15; 17 -> 19; 18 -> 29; 11 -> 4; 14 -> 15;
        37 -> 39; 37 -> 41; 37 -> 38; 37 -> 40; 13 -> 19; 12 -> 29;
        43 -> 38; 43 -> 40; 36 -> 19; 32 -> 23; 34 -> 29; 39 -> 15;
        41 -> 29; 38 -> 4; 40 -> 19; 4 -> 5; 19 -> 21; 19 -> 20;
        19 -> 28; 5 -> 6; 5 -> T35; 5 -> 23; 21 -> 22; 20 -> 15; 28 -> 29;
        6 -> 7; 15 -> T1; 22 -> 23; 22 -> T35; 29 -> T30; 7 -> T8;
        23 -> T24; 23 -> T1;
}
```

Figure 7. Gansner et al Optimal Rank Generation[3]




```
digraph shells {
        size="7,8";
        node [fontsize=24, shape = plaintext];
        1972 -> 1976 -> 1978 -> 1980 -> 1982 -> 1984 -> 1986 -> 1988
                -> 1990 -> future;

        node [fontsize=20, shape = box];
        { rank = same;  1976 Mashey Bourne; }
        { rank = same;  1978 Formshell csh; }
        { rank = same;  1980 esh vsh; }
        { rank = same;  1982 ksh "System-V"; }
        { rank = same;  1984 v9sh tcsh; }
        { rank = same;  1986 "ksh-i"; }
        { rank = same;  1988 KornShell Perl rc; }
        { rank = same;  1990 tcl Bash; }
        { rank = same;  "future" POSIX "ksh-POSIX"; }

        Thompson -> {Mashey Bourne csh}; csh -> tcsh;
        Bourne -> {ksh esh vsh "System-V" v9sh}; v9sh -> rc;
              {Bourne "ksh-i" KornShell} -> Bash;
        {esh vsh Formshell csh} -> ksh;
        {KornShell "System-V"} -> POSIX;
        ksh -> "ksh-i" -> KornShell -> "ksh-POSIX";
        Bourne -> Formshell;

                /* 'invisible' edges to adjust node placement */
        edge [style=invis];
        1984 -> v9sh -> tcsh ; 1988 -> rc -> KornShell;
        Formshell -> csh; KornShell -> Perl;
}
```

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.

3.1 Phase II – Development

The Development Phase follows the Algorithm Phase. This phase will include 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 spring semester, the development process will loosely follow the Agile methodology using SCRUM. The project will be time-boxed into 2-week sprints beginning in January. Each sprint will begin with a meeting with the project advisor to discuss the project progress and if the sprint's goals are reasonably appointed. The following meeting will follow-up on the project's progress and demonstrate any working samples from that sprint's development.

Since the project will be created by one person, there will be no daily standup meetings. Daily progress may be recorded internally as reference for the developer. Testing will also be done individually and bug fixes will be part of the development phase.

The development phase will result in the deliverable of the completed project as a web application.

4 Environment

The environment used to develop the project includes the following hardware and operating system:

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.

The following software will be used in the development of the program:

1. Frameworks: Django, Bootstrap.
2. Programming Languages: Python, DOTS.
3. Web Languages: HTML/HTML5, CSS/CSS3, JavaScript/jQuery.
4. Database: MongoDB (NoSQL, JSON).
5. Graph Visualization: GraphViz or NetworkX.

5 Project

The final results of the project will include an (1) Algorithm Design Document which detail the rationale behind the algorithm selection and (2) a working web 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 strata by semester.* The graph output will be expected to organize its solution into courses by semester. An appropriate drawing will contain with no more than 4 courses (12 units) of computer science courses per semester and at least 1 courses (3 units) of computer science courses per semester. Courses in the same semester will appear in the same row on the graph.

3. *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.

4. *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.

5. *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.

6 Schedule

Table 4 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 4. Project Schedule

2016	January				February				March				April				May				Summary	
Tasks:	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	Hours	Percent
Requirements	8	8																			16	5.2%
Research & Discovery	8	8	8	8	8	8															48	15.6%
Design			8	8	8	8	8	8													48	15.6%
Integrate & Test							8	8	20	20	20	20	20	20	8	8					152	49.4%
Write User's Manual															8	8					16	5.2%
Write Final Report																	8	8	8		24	7.8%
Demonstrate																				4	4	1.3%
Hours	16	16	16	16	16	16	16	16	20	20	20	20	20	20	16	16	8	8	8	4	**308**	100.0%

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.

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.

8 References

[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