AMERICAN UNIVERSITY OF BEIRUT

GENERATING FEASIBLE UNIVERSITY STUDENT SCHEDULES VIA GRAPH THEORY CONCEPTS

by

MOHAMMAD ALQASIM BASSAM ALZAEEM

A thesis
submitted in partial fulfillment of the requirements
for the degree of Master of Engineering Management
to the Industrial Engineering and Management Program
of the Faculty of Engineering and Architecture
at the American University of Beirut

Beirut, Lebanon
August 2016

AMERICAN UNIVERSITY OF BEIRUT

GENERATING FEASIBLE UNIVERSITY STUDENT SCHEDULES VIA GRAPH THEORY CONCEPTS

by
MOHAMMAD ALQASIM BASSAM ALZAEEM

Approved by:

[illegible]

Dr. Bacel Maddah, Associate Professor Industrial Engineering and Management	Advisor

[illegible]

Dr. Moueen Salameh, Professor Industrial Engineering and Management	Member of Committee

[illegible]

Dr. Hussein Tarhini Assistant Professor Industrial Engineering and Management	Member of Committee

Date of thesis: August 16, 2016

ACKNOWLEDGMENTS

I would like to express my gratitude to Allah (God) for providing me the blessings to complete this work. There are a number of people without whom this thesis might not have been written, and to whom I am greatly indebted.

Foremost, I would like to express my sincere gratitude to my advisor Prof. Bacel Maddah for the continuous support toward my thesis research who was always at hand with a useful piece of advice. I would also like to express my honor to have been advised by Prof. Moueen Salameh and thank him for pointing me into this research direction in the first place. I would also like to thank Prof. Hussein Tarhini for his encouragement and assistance.

Last but not least, I thank my beloved wife for her encouragement, company and unconditional support, and also would like to thank my family and friends for their love and support.

AN ABSTRACT OF THE THESIS OF

Mohammad Alqasim Alzaeem for Master of Engineering Management
Major: Industrial and Financial Engineering

Title: Generating Feasible University Student Schedules via Graph Theory Concepts

This thesis aims to provide an effective method that helps university students in registering several classes having multiple sections each. With many classes to register (reaching 6 or 7 classes) and several sections per class (reaching 8 or 9 sections) this becomes a scheduling problem that is tedious to solve. The difficulty stems from finding available options (schedules) that comply to a mixture of hard constraints imposed by the university and soft constraints related to student's preferences. This thesis describes the design and implementation of a constraint-based personal student course scheduler that utilizes graph theory.

A special graph is constructed which consists of special vertices (referred to as super nodes) that represent the courses offered by the university. Each super node has sub-nodes representing sections of the class. The connectivity is defined among super nodes. Two super nodes are connected by an arc if they have no time conflict. Sub-nodes (sections) that belong to connected super-nodes can be part of the student schedule. This graph theory methodology allows generating all possible schedules for a student accounting for various constraints. We apply our method to generate student course schedules based on the course offerings of the American University of Beirut (AUB). This is developed using a user-friendly PHP application which runs online, and can be easily connected to a university student information system.

CONTENTS

ILLUSTRATIONS

TABLES

CHAPTER I.

INTRODUCTION & BACKGROUND

Recent years have seen an increased application of graph theory and mathematics Chachra, Vinod et al, (1979). The former has proven to be particularly useful to a large number of rather diverse fields. The exciting and rapidly growing area of graph theory is rich in theoretical results as well as applications to real-world problems.

The problem of students registration in university is an increasingly complex process, due to the growing flexibility of course schedules and the interest in including a wide set of constraints and objectives. Any university usually has multiple faculties or schools that are responsible for the academic organization of classes. At the time of registration, the university makes the complete course offerings schedule available to all students, with the information on all courses of each faculty. The students use this information to make their course selection, which is basically a set of compulsory and optional courses from among those offered by their faculty and some other courses from the other faculties ("elective" courses).

In this thesis we present an approach that enables the students to enter their course selection with their own preferences in a user-friendly web-application that uses an algorithm based on graph theory.

The origins of graph theory can be traced back to puzzles that were designed to amuse mathematicians and test their ingenuity. The classic puzzle concerned the bridges of Königsberg, a town in Prussia, which surrounded an island in the River Pregel (the town is now known as Kliningrad on the Pregolya River). Graph theory is considered to have begun in 1736 with the publication of Leonard Euler’s solution to the Königsberg bridge problem.

Many real-life situations can be described by means of a diagram consisting of a set of points with lines joining certain pairs of points. Loosely speaking, such a diagram is what we mean by a graph. Graphs lend themselves naturally as models for a variety of situations. Instances of graphs abound: for example, the points might represent cities with lines representing direct flights between certain pairs of these cities in some airline system, or perhaps the lines represent pipelines between certain pairs of these cities in an oil network. On the other hand, the points might represent factories with lines representing communication links between them. Electrical networks, multiprocessor computers or switching circuits may clearly be represented by graphs.

A. Statement of the Problem

Scheduling student individual courses is usually made by each student at the start of a semester when choosing what courses to register. Manually created schedule is known to contain a lot of conflicts and problems despite the great effort required to form such a schedule.

The process of identifying of the best schedule that meet the hard constrains of the university and the soft constrains based on student preferences can be a very tedious job. Therefore, providing a requisite schedule that meet these constrains will minimize the student dissatisfaction, and saves the student time.

B. Objective of the Study

The purpose of this thesis work is to apply graph theory concepts to generate efficient and conflict free student's personal courses schedule. The work in this research seeks to produce a set of options for each student from which a student can choose the best option to opt-in or register in.

C. General Introduction about graph theory and networks

1. *History of Graph Theory*

The theory of graphs is based on the problem of the Seven Bridges of Königsberg which was firstly formulated by the Swiss mathematician Leonard Euler (1707-1783) the father of graph theory. Königsberg[1] is located at both sides of the river Pregel and includes two large islands, connected to the rest of the city through seven bridges (Fig.1).

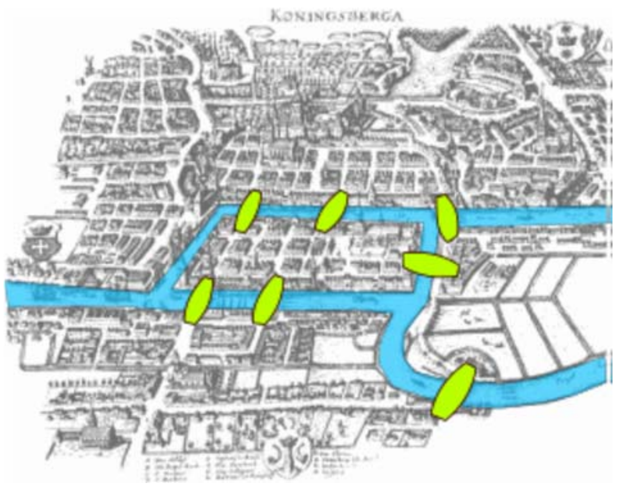


Figure 1 the seven bridges of Königsberg (A view of Königsberg as it was in Euler's day)

[1] During World War II Königsberg was conquered by the soviet Red Army and then integrated into the Soviet Union but its name has been changed to Kaliningrad.

The idea of the problem is to find a path through the city on which each bridge is crossed once and only once. Euler found out, that the key problem to be solved is, to find the sequence of bridges crossed. This enabled him to reshape the problem into a mathematical structure of a graph. Here the different land masses represent the vertices of the graph and the bridges are edges between the vertices (Fig.2).

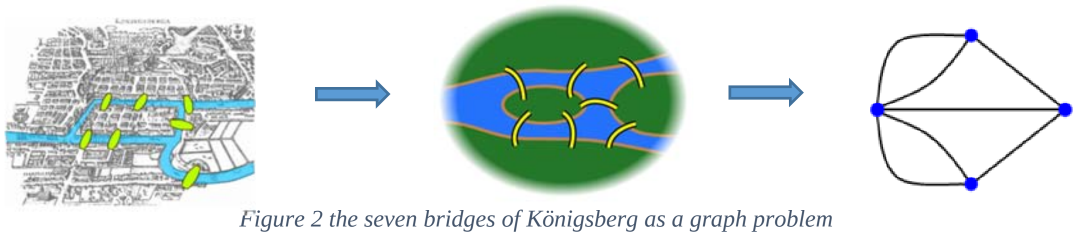


Figure 2 the seven bridges of Königsberg as a graph problem

Euler pointed out that, during any walk, whenever one enters a land mass through a bridge, one has to leave it again over another bridge (since every bridge shall be crossed only once). This means that, if every bridge is crossed only once, the number of bridges touching a land mass has to be even (except for the land masses one defines to be start or finish of the walk). Since all four land masses are touched by an odd number of bridges, this leads to a contradiction. In other words, Euler proved that such a walk (later called a Eularian walk in his honor) exists, if and only if (a) the graph is connected and (b) there exist exactly two or zero vertices with odd degree[2]. By stating and solving this problem, Euler laid down the principles of modern graph theory.

[2] Euler stated that this condition is a necessary for a Eularian walk.

2. ***Graph Theory Definitions***

a. Graph

A graph G consists of a collection *V* of vertices and a collection edges *E*, for which we write *G = (V, E)* Chachra, Vinod et al, (1979). Each edge $v \in E$ is said to join two vertices, which are called its end points. If *e* joins $u, v \in V$, we write *e = (u, v)*. Vertex *u* and *v* in this case are said to be *adjacent*.

The order of a graph is the number of vertices in it, usually denoted |*V*| or |*G*|. The size of a graph is the number of edges in it, denoted |*E*| or ||*G*||. If |*V*| = *0* and |*E*| = *0* the graph is called empty or null. If |*V*| = *1* and |*E*| = *0*, the graph is trivial. If |*V*| >= *1* and |*E*| = *0* the graph is called discrete.

b. Directed Graph

A directed graph or digraph is an ordered pair *D = (V, A)* with V a set whose elements are called vertices or nodes, and *A* a set of ordered pairs of vertices, called arcs, directed edges, or arrows. An arc *a = (x, y)* is considered to be directed from *x* to *y; y* is called the head and *x* is called the tail of the arc; *y* is said to be a direct successor of *x*, and *x* is said to be a direct predecessor of *y* Chachra, Vinod et al, (1979).

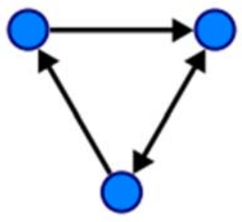


Figure 3 Directed Graph

c. Undirected Graph

An undirected graph is one in which edges have no orientation. The edge *(a, b)* is identical to the edge *(b, a)*, i.e., they are not ordered pairs, but sets *{u, v}* (or 2-

multisets) of vertices. In this thesis we are interested in undirected graphs more than in directed ones Chachra, Vinod et al, (1979).

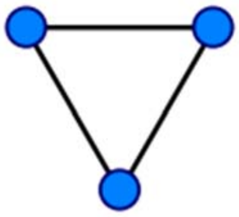


Figure 4 Undirected Graph

3. *Graph Theory Basics*

Graph theory is the study of graphs, whereas graphs are, at the lowest level of abstraction, simple data structures comprised of a set of points that are connected arbitrarily by a set of lines.

A graph is a mathematical structure consisting of two sets *V* and *E*. The elements of *V* are called vertices and the elements of *E* are called edges. Each edge is identified with a pair of vertices. If the edges of a graph *G* are identified with ordered pairs of vertices, then *G* is called a directed graph. Otherwise *G* is called an undirected graph. Our discussions in this thesis are concerned with undirected graphs.

The number of vertices in *V (G)* is called the order of the graph, the number of edges in *E (G)* is called the size *(M)* and the number of edges connecting a vertex *v* to other vertices is called the degree of *v*, K_v. In practice, graphs can be used to model all kinds of networks. Here, vertices represent network elements (such as people, computers, cities, and courses) and edges represent relationships between these network elements (such as friendship bonds, Ethernet connections, railroad connections, and registration without conflict).

4. *Metrics of Graph*

An *adjacency matrix M (G)* is a $n \times n$ matrix (with *n* being the order of the graph) in which $M_{i;j}$ is either *1* (if vertex i is connected to vertex j) or *0* (if otherwise) Chachra, Vinod et al, (1979). This matrix tends to be very sparse what makes it easy to store it quite efficiently in computer memory. The *adjacency list* is a simple list of all vertices of the graph with all vertices to which they are connected next to them. In other words, in a graph adjacency list, the adjacency lists of all vertices are stored. The size of vertex v's adjacency list is equal to its degree k_v.

D. Special Graph super node and sub-nodes

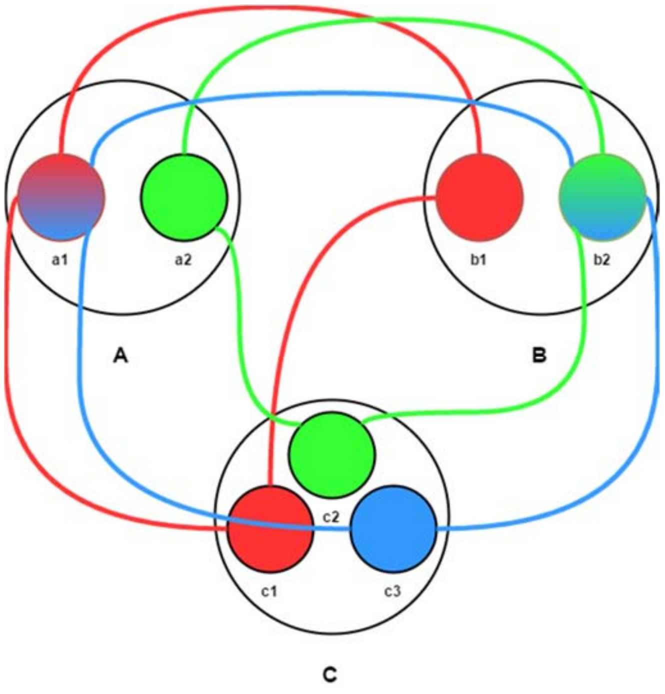


Figure 5 Special Graph Super Node and Sub-Node

E. Adjacency Matrix

Let $V_G = \{v_1 \ldots v_n\}$ be ordered. The adjacency matrix of G is the $n \times n$ matrix M with entries $M_{ij} = 1$ or $M_{ij} = 0$ according to whether $v_iv_j \in G$ or $v_iv_j \notin G$.

For instance, the graph in Figure 6 has an adjacency matrix in Table 1. Notice that the adjacency matrix is always symmetric (with respect to its diagonal consisting of zeroes).

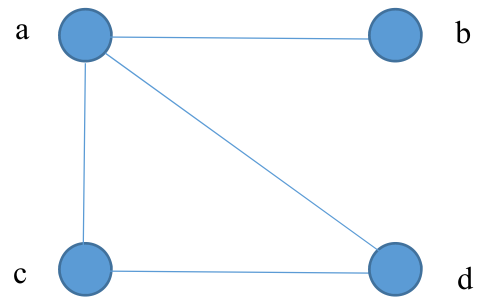


Figure 6 Graph G

	a	b	c	d
a	0	1	1	1
b	1	0	0	0
c	1	0	0	1
d	1	0	1	0

Table 1 Adjacency Matrix of Graph G

A graph has usually many different adjacency matrices, one for each ordering of its set V_G of vertices. The above result is obvious from the definitions.

In our case we can represent the courses sections as nodes and the adjacency matrix entries is 1 if two sections can be registered together without time conflict, else it is 0 if there any time conflict between the two sections.

F. Sub graphs

A graph H is a sub graph of a graph G, denoted by $H \subseteq G$, if $V_H \subseteq V_G$ and $E_H \subseteq E_G$.

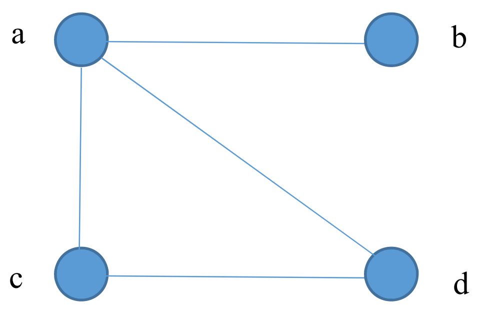


Figure 8 Graph G

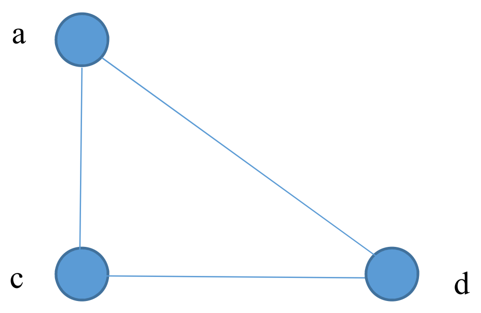


Figure 7 Sub-Graph of G

G. Complete Sub graph (clique)

A Complete Sub graph (clique), C in an undirected graph $G = (V, E)$ is a subset of the vertices, $C \subseteq V$, such that every two distinct vertices are adjacent Chachra, Vinod et al, (1979). This is equivalent to the condition that the sub graph of G induced by C is complete.

1. ***Maximum Clique***

A maximum clique is a clique of the largest possible size in a given graph. Figure 9 present an example of a maximum clique.

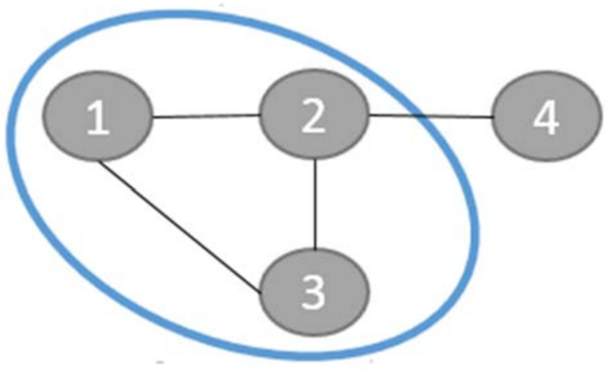


Figure 9 Maximum Clique

2. *Maximal Clique*

A maximal clique is a clique that cannot be extended by including one more adjacent vertex. Figure 10 shows an example of this clique.

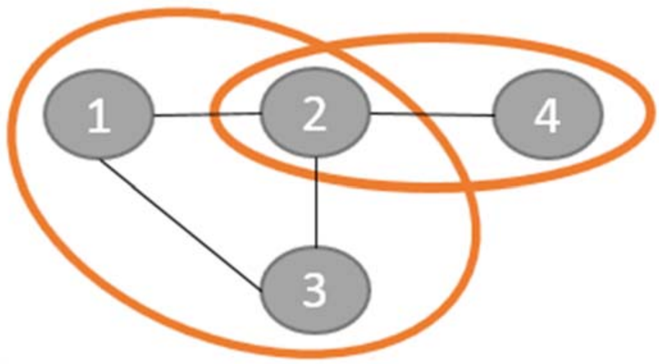


Figure 10 Maximal Cliques

H. Degrees of vertices

Let $v \in G$ be a vertex in a graph G. The neighborhood of v is the set $N_G(v) = \{u \in G \mid vu \in G\}$. The degree of v is the number of its neighbors or the number of edges that come out from a vertex: $(v) = |N_G(v)|$. If $d_G(v) = 0$, then v is said to be isolated in G, and if $d_G(v) = 1$, then v is a leaf of the graph. The minimum degree and the maximum degree of G are defined as $\boldsymbol{\min \{d_G(v) \mid v \in G\}}$ and $\boldsymbol{\max \{d_G(v) \mid v \in G\}}$.

CHAPTER III

LITERATURE REVIEW

Timetabling in educational systems is a broad research topic with a very strong relation to graph theory which plays a major role in timetabling problems Schroeder, Roger, (1973). The construction of timetabling is one of the main research area that management science addresses in educational systems which has three main categories: University Courses Timetabling, Exam Timetabling, and Student Sectioning White, Gregory, (1987). A significant amount of research effort has developed powerful methods that utilize graph theory to solve the problem regarding University Course Timetabling and Exam Timetabling. However, the Students Sectioning problem has received the least attention, since it is regarded as a sub problem of timetabling.

Wren, Anthony, (1995) defines: "Timetabling is the allocation, subject to constraints, of given resources to objects being placed in space time, in such way as to satisfy as nearly as possible a set of desirable objectives."

A. University Course Timetabling

University Course Timetabling problem is the process of assigning course's lectures to time slots, rooms, teachers and other resources, subject to multiple constraints. The large scale of universities usually makes solving such problem so complicated that it is solved by many experts from different faculties and departments coming from all over the university. Because of the complexity of this problem, it is common for the universities to use the previous year solution when solving the problem.

University Course Timetabling has two major approach to solve it: Post Enrolment Course Timetabling and Curriculum-based Course Timetabling. The former utilizes the student enrollment data for each student to determine the course timetable in such a way that allows all student to attend all their registered courses without time clashes Lewis, Rhydian et al, (2007).

The second approach is the Curriculum-based Timetabling that assign courses lectures to time slots based on curriculum which states that a combination of courses are required to satisfy a degree in a given major. This approach constructs the weekly classes' timetable where conflict between courses are based on the curriculum approved by the educational institute Di Gaspero, (2007) Carter, Michael, (2000).

University course timetabling is subject to multiple constraints that are categorized in two main types: hard constraints and soft constraints. A list of some of the most common constraints is:

Hard constraints:

1) No student or teacher can be in two different places at the same time.

2) Only one class can occupy a room in any time slot.

3) Any room must satisfy all class requirements.

Soft constraints:

1) The preferences of time.

2) The preferences of room.

3) Students schedule compactness.

Many works uses a two steps method to solve the University course timetabling. The first step for creating the timetable and the second step is to improve the constructed timetable. Azlan, et al, (2013) the focus was on the first step to produce a good initial solution to start from, and the initial solutions were generated by implementing graph coloring heuristic to the Curriculum-based course timetabling problem.

B. Exam Timetabling

Exam Timetabling is the process of assigning a number of exams to a limited number of time slots, where each course has one event represented by the exam. The main goal of the exam timetabling is to avoid conflict in each student's exam timetable and to make sure that each student has enough time to prepare for each exam Schaerf, Andrea, (1999) Qu, R. Burke, (2009). Although, there are many similarity between university course timetabling and exam timetabling, but the main difference between the two is that university course timetabling aim to generate a compact schedule, whereas exam timetabling tend to provide more spread timetable to allow students to have sufficient time to prepare for the exam. Also, there is only one exam per course, and it is possible to have more than one exam in a single room. Following are some of the common constraints used in Exam Timetabling:

Hard constraints:

1) No student or teacher can be in two different places at the same time.

2) There must be enough room capacity for each examination period.

Soft constraints:

1) Spreading examination period for student examination schedule.

2) The preferences of time (early, late...).

3) Consecutive exams for any student to be minimum (Back to Back).

4) Dividing the exam on multiple rooms.

Many research utilize graph based methods to address the exam timetabling problem. Rahman, Syariza Abdul, (2014). Two graph coloring heuristics with an adaptive linear combination of heuristics for solving exam timetabling problems.

C. Student Sectioning

Student sectioning is the process of assigning students to sections Feldman, Ronen et al, (1989). In the previous timetabling problems we can see that we are assigning lectures or exam to a time slot, whereas in the problem of student sectioning we assign students to sections. A section is a course instance i.e. a copy of the same course but each section of a given course has its own room, teacher, and time. Student sectioning problem respects the individual student's preferences while assigning students to course sections Kingston, Jeffrey, (2013). A list of the common student sectioning constraints are:

Hard constraints:

1) No student can attend two courses having a time conflict.

2) Each section has limitation on size.

Soft constraints:

1) Keeping the number of course section minimum.

2) Balancing course sections.

Most of the research studies concerning student sectioning address this problem as sub-problem of University courses timetabling and not a dedicated problem. Many technique were used to overcome this problem Carter, Michael, (2000) Carter describes a demand driven timetabling in which student preference of courses are used to construct a timetabling which satisfies as many student preferences as possible.

Sönmez, Tayfun et al, (2010) Introduce a bidding system where students make bids on the courses they want to register in. Based on the bids created by the students a class is assigned to room that has the enough capacity to hold the students requesting to register in a given course.

Müller, et al, (2010) solve the student sectioning problem as part of university timetable by applying it in two phases the first is during the constructing of the university course timetabling, and the second is after the university course timetabling is created.

Feldman and Golumbic (1990) introduce priorities on constraints in the student sectioning problem indicating which schedules are preferred over others by the student. A number of algorithms are then presented for finding the best schedule minimizing violation of student priorities. Sampson and Weiss (1995) furthered the idea of accommodating student preferences in both the timetabling and sectioning problems by introducing a heuristic approach based on each student's priority ordering of the courses and sections they wished to enroll in.

As mentioned above we notice that student preferences has the least attention, and it’s not addressed as specific problem by itself. Even though, students are the largest in number in any educational institute. Also any of the previous solutions require significant modification, or changing the current university course timetabling solution completely to take into consideration the student preferences.

In this thesis, we utilize graph theory concepts to represent university courses and course's sections as a special graph, which enable us to provide an add-on to any current university course timetabling solution to take into account student's preferences and help them in easing the process of registering their desired courses without the hassle of constructing their personal course schedule every semester.

CHAPTER II.

METHODOLOGY

In order to solve the student personal course schedule problem, a student needs to identify the desired courses first then to check each course sections for conflict while trying to take into account their personal preferences while keeping a conflict free schedule all that in a manual fashion.

This kind of schedule problem revolves about allocation of resources in an effective way for the student.

Graph theory techniques were used in generating all the possible options for a student. A special graph was constructed which consists of special vertices (referred to as super nodes) that represent the courses offered by the university (see chapter 1). Each super node has sub-nodes representing sections of the class. The connectivity is defined among sub nodes if two sub nodes are connected by an edge (they have no time conflict). Sub-nodes (sections) that have no time conflict can be taken by the student. This graph theory methodology allows generating all possible schedules for a student accounting for various constraints.

Because the student personal course schedule problem constraints analysis is very important in solving this problem. During the requirements analysis several constraints were identified. After analyzing those constraints, it was divided into two groups according to the effect of those constraints to the final possible solutions. Violating of some constraints will affect the solution directly and violating of some other constraints will be affecting the quality of the possible solutions (based on student's preferences).

Similar to other studies which were carried out to solve scheduling problem, the two groups of constraints were named as hard constraints and soft constraints. Hard constraints are the constraints, which must be satisfied to get feasible solution for use in practice and soft constraints are used to evaluate the quality of the solution. So soft constraints are not compulsory but are desired to be satisfied as much as possible.

The main objective is to find a conflict-free timetable for each student. Some students are considered part-time students because they can only attend classes in the morning or only in the afternoon, mainly due to work commitments. In this case, there is a virtual course constructed to prevent selecting classes in non-available periods. The rest of the students are full-time. As most of them would like to attend classes in the morning so a virtual course can be assigned to the afternoon periods to prevent selecting classes in the non-preferred periods.

For any student we know the course offerings provided by the university, and we want to build a set of section assignments producing the best possible personal schedules, according to the constraints chosen by the student.

The construction of the available set of personal schedule for any student may be described as follows:

1. Building a non-directed special graph $G = (V, E)$ which consists of special vertices V (referred to as super nodes) that represent the courses offered by the university. Each super node has sub-nodes representing sections of the class. The connectivity E is defined among sub nodes. Two sub nodes are

connected by an edge if they have no time conflict. Sub-nodes (sections) that have no time conflict can be taken by the student. The preferences of the student can be considered when building their graph *G* by including virtual courses as super nodes to prevent having schedule with these non-preferred periods.

2. Using the algorithm of Bron and Kerbosh (1973) to enumerative over the student special graph to generate independent sets (schedules) from graph *G*. These sets (schedules) are feasible. Section assigned to a schedule where at most one section (sub-node) of each course (super-node) with no time conflicts between sections. The original algorithm is modified to look only for maximum degree of complete and independent schedule, that is, sets containing exactly one section of each course. In the enumerative process store at each time only those sets of the maximum degree obtained so far.

3. At the end of Step 2 we have a set of solutions that contains all the possible personal schedule for a given student.

…

A. The Algorithm

1. *The Bron–Kerbosch Algorithm*

The Bron–Kerbosch algorithm (1973) is a widely used algorithm for finding all maximal cliques in a graph. It is a recursive backtracking algorithm which is easy to understand, and has been shown to work well in practice. A recursive call to the Bron–Kerbosch algorithm provides three disjoint sets of vertices *R, P*, and *X* as arguments, where *R* is a (possibly non-maximal) clique and $P \cup X = N(R)$ are the vertices that are adjacent to every vertex in *R*. The vertices in *P* will be considered for inclusion in clique *R*, while those in *X* must be excluded from the clique; thus, within the recursive call, the algorithm lists all cliques in $P \cup R$ that are maximal within the sub-graph induced by $P \cup R \cup X$. The algorithm chooses a candidate *v* in *P* to add to the clique *R*, and makes a recursive call in which *v* has been moved from *R* to *P*; in this recursive call, it restricts *X* to the neighbors of v, since non-neighbors cannot affect the maximality of the resulting cliques.

When the recursive call returns, *v* is moved to *X* to eliminate redundant work by further calls to the algorithm. When the recursion reaches a level at which *P* and *X* are empty, *R* is a maximal clique and is reported. To list all maximal cliques in the graph, this recursive algorithm is called with *P* equal to the set of all vertices in the graph and with *R* and *X* empty. The algorithm is presented next.

Algorithm 1: Bron-Kerbosch algorithm

```
BronKerbosch(P, R, X)
1: if $P \cup X = /0$ then
2: report R as a maximal clique
```

```
3: end if
4: for each vertex v ∈ P do
5: BronKerbosch(P∩ N(v), R∪ {v}, X ∩ N(v))
6: P ← P\ {v}
7: X ← X ∪ {v}
8: end for
```

On the first call *R* and *X* are set to *∅*, and *P* contains all the vertexes the graph. *R* is the temporary result, *P* the set of the possible candidates and *X* the excluded set. *N(v)* indicates the neighbors of the vertex *v*.

The algorithm can be described as following: Pick a vertex *v* from *P* to expand. Add *v* to *R* and remove its non-neighbors from *P* and *X*. Then pick another vertex from the new *P* set and repeat the process. Continue until *P* is empty. Once *P* is empty, if *X* is empty then report the content of *R* as a new maximal clique (if it is not then *R* contains a subset of an already found clique). Now backtrack to the last vertex picked and restore *P*,*R* and *X* as they were before the choice, remove the vertex from *P* and add it to *X*, then expand the next vertex. If there are no more verses in *P* then backtrack to the superior level.

2. ***Modification of Bron-Kerbosch Algorithm***

This modified version of Bron-Kerbosch algorithm is more efficient since it help reduce the number of recursive calls made hence having less time to generate all maximal cliques in a given undirected graph *G (E, V)*. In this approach the algorithm will use the nodes which have a degree of at least *(k-1)* and *k* is the desired number of vertices in any maximal clique found. And this happen when the algorithm iterate over the only the set of vertices in set *P* which have a degree of at least *(k-1)*, and will be removed from set *P* if any node has a degree less than *(k-1)*.

Using this modified version of the Bron-Kerbosch algorithm we can set the number of the desired degree of each vertex and traverse over the vertices that will most likely will be in a maximal clique.

The modified algorithm can be described as following: Pick a vertex v that has a degree at least *(k-1)* from *P* to expand else remove this vertex from *P* and continue the loop. Add *v* to *R* and remove its non-neighbors from *P* and *X*. Then pick another vertex from the new *P* set but that only has a degree at least equal to *(k-1)* else remove this vertex from the new *P* and repeat the process. Continue until *P* is empty. Once *P* is empty, if *X* is empty then report the content of *R* as a new maximal clique (if it's not then *R* contains a subset of an already found clique). Now backtrack to the last vertex picked and restore *P*,*R* and *X* as they were before the choice, remove the vertex from *P* and add it to *X*, then expand the next vertex. If there are no more vertexes in *P* then backtrack to the superior level. The modified algorithm is presented next.

Algorithm 2: Modified Bron-Kerbosch algorithm

```
MBronKerbosch(P, R, X)
1: if $P \cup X = /0$ then
2: report R as a maximal clique
3: end if
4: for each vertex $v \in P$ do
5: if v's degree $\geq k-1$ then
6: MBronKerbosch($P \cap N(v)$, $R \cup \{v\}$, $X \cap N(v)$)
7: $P \leftarrow P \setminus \{v\}$
8: $X \leftarrow X \cup \{v\}$
```

```
9: else
10: P ← P\ {v}
11: end if
12: end for
```

3. *Modified Version Efficiency Demonstration*

Our modification of the Bron-Kerbosch algorithm has helped us reduce the time complexity required to find all maximal cliques in a given undirected graph *G(E,V)* by reducing the recursive calls needed to find all maximal cliques. In the following example we illustrate how the modified version minimize the required recursive calls compared to the original Bron-Kerbosch algorithm.

In this example we have a simple undirected graph *G (E, V)* that consist of four nodes *{1, 2, 3, 4}*.

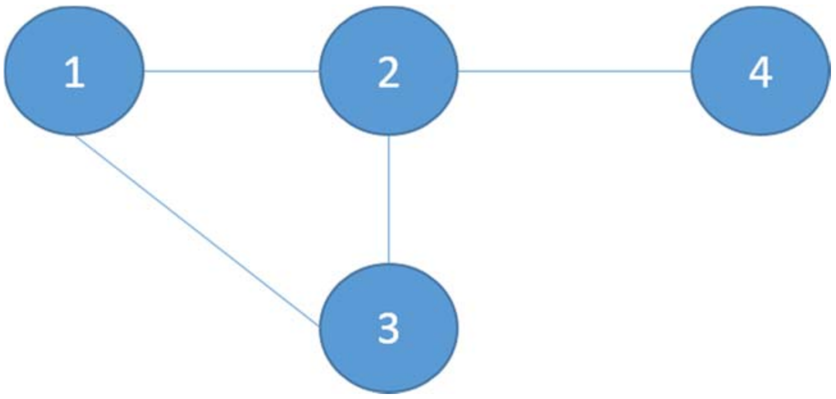


Figure 11 Undirected Graph G (E, V)

First we will show the recursive calls made using the original Bron-Kerbosch algorithm that will call from now on Algorithm 1, after that we will present our modified version of Bron-Kerbosch algorithm and we will refer to it as Algorithm 2.

a. Algorithm 1 Bron-Kerbosch Recursive Calls

- ***Start with***
    - ***R = {}, P = {1, 2, 3, 4}, X = {}***
- ***BK ({}, {1,2,3,4}, {})***
    - ***v = {1}***
    - ***BK ({1}, {2,3}, {})***
        - ***v = {2}***
        - ***BK ({1,2}, {3}, {})***
            - ***v = {3}***
            - ***BK ({1,2,3}, {}, {})***
            - ***Maximal Clique Found {1,2,3}***
        - ***v = {3}***
        - ***BK ({1,3}, {}, {2})***
        - ***Duplicate***
    - ***v = {2}***
    - ***BK ({2}, {3,4}, {1})***
        - ***v = {3}***
        - ***BK ({2,3}, {}, {1})***
        - ***Duplicate***
        - ***v = {4}***
        - ***BK ({2,4}, {}, {})***
        - ***Maximal Clique Found {2,4}***
    - ***v = {3}***

        - ***<u>BK ({3}, {}, {1,2})</u>***
        - ***Duplicate***
        - ***v = {4}***
        - ***<u>BK ({4}, {}, {2})</u>***
        - ***Duplicate***

By observing the number of the recursive calls made by Algorithm 1 indicated by the underlined BK() calls we can see that for such small graph the Algorithm 1 is making 10 recursive calls to find two maximal cliques the first is clique {1,2,3} and the second is clique {2,4}.

b. <u>Algorithm 2 Modified Bron-Kerbosch Recursive Calls</u>

First we set the desired degree value which for this example will be (k = 3).

- ***Start with***
    - ***R = {}, P = {1, 2, 3, 4}, X = {}***
- ***<u>BK ({}, {1,2,3,4}, {})</u>***
    - ***v = {1}***
    - ***<u>BK ({1}, {2,3}, {})</u>***
        - ***v = {2}***
        - ***<u>BK ({1,2}, {3}, {})</u>***
            - ***v = {3}***
            - ***<u>BK ({1,2,3}, {}, {})</u>***
            - ***Maximal Clique Found {1,2,3}***
        - ***v = {3}***

        - ***<u>BK ({1,3}, {}, {2})</u>***
        - ***Duplicate***
    - ***v = {2}***
    - ***<u>BK ({2}, {3,4}, {1})</u>***
        - ***v = {3}***
        - ***<u>BK ({2,3}, {}, {1})</u>***
        - ***Duplicate***
    - ***v = {3}***
    - ***<u>BK ({3}, {}, {1,2})</u>***
    - ***Duplicate***

By setting *k* equal to 3, we can see that the number of the recursive calls made by Algorithm 2 indicated by the underlined BK() calls using the same graph used by Algorithm 1 the Algorithm 2 is making 8 recursive calls to find one maximal cliques that is clique {1,2,3}. That is less recursive calls and therefore less complexity time. Applying this algorithm on a larger undirected graph will have huge complexity reduction compared to Algorithm 1.

4. *Applying Modified Bron-Kerbosch Algorithm on Student Course Schedule*

Finding all maximal Cliques of size *K* (all complete sub graphs) in an undirected graph using the algorithm 457 by Bron and Kerbosch (1973). Using graph theory concepts to represent the selected courses as graph where the courses are its vertices and the availability to register two courses without conflict is the edge between these two vertices. The availability of two course to be registered in the same schedule can be represented using the adjacency matrix. In this case the adjacency matrix will denote the adjacent courses sections i.e. two courses sections are adjacent if the two courses (nodes of the graph) are connected. In other words, both courses sections can be registered without time conflict in the same schedule.

Then, finding all possible courses schedules by detecting all complete sub-graphs ("cliques") of size *k* Bron, and Kerbosch, (1973) from the original graph as shown in Figure 1.

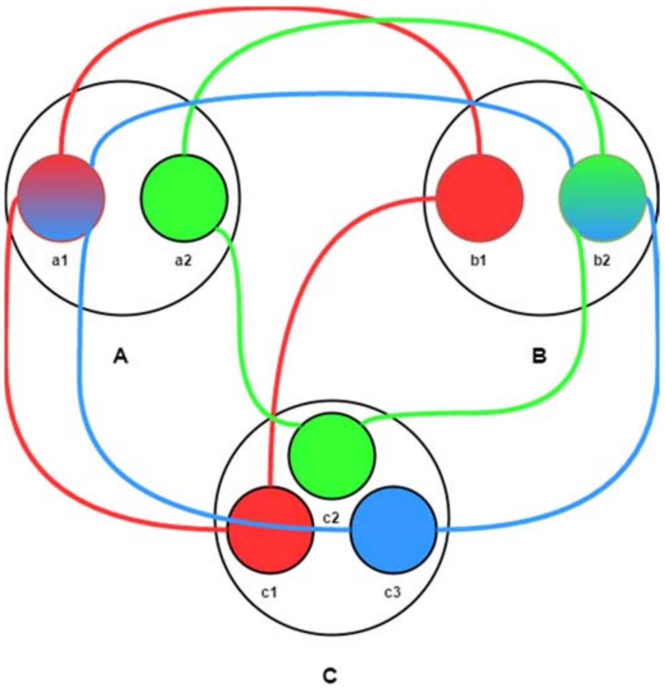


Figure 12 Sub-graphs (cliques) example

In the Figure 12 example we can see three complete sub-graphs (cliques) colored as (red {a1, b1, c1}, green {a2, b2, c2}, and blue {a1, b2, c3}) where each of the sub-graph is of **size *k*** (***k*=3** in this example).

In order to find the best student personal courses schedule for a given set of courses. The method has to list all *K*-size sub-graph (*K*-clique), where every complete sub-graph (clique) has two properties that every two vertices in a sub-graph is connected (adjacent), and number of vertices in the sub-graph is equal to ***K***.

B. Super Graph

A super graph *G* consists of a collection *(V)* of vertices super node where each have one or more sub-nodes *(v)* and a collection edges *E*, for which we write $G = (V, E)$. Each edge $e \in E$ is said to join two vertices, which are called its end points. If *e* joins v_i, $v_j \in V$, we write $e = (v_i, v_j)$. Vertex (sub-node) v_i and v_j in this case are said to be adjacent.

Example of super graph as shown below:

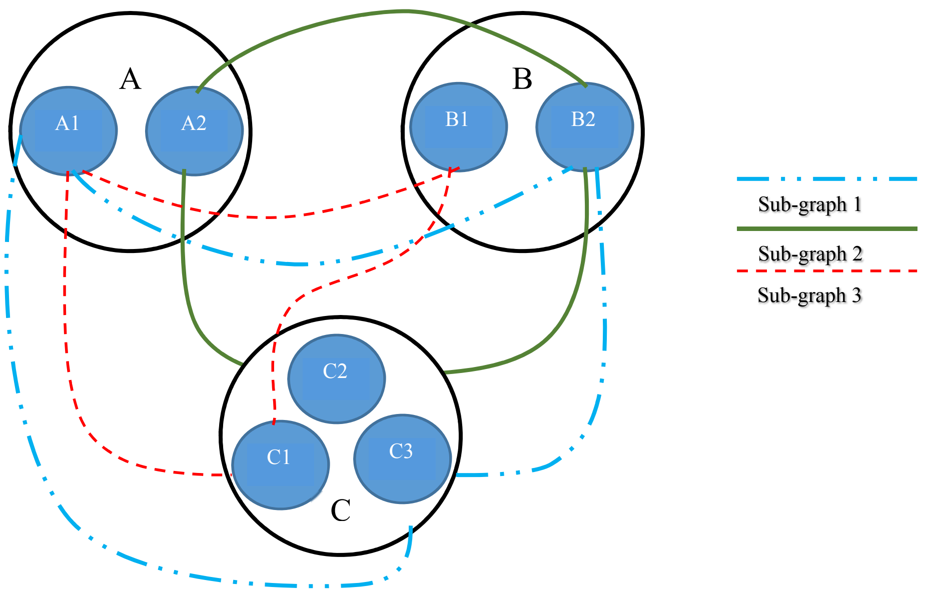


Figure 13 Super Graph

1. ***Super Node***

A super node is the vertices in super graph that contain at least one or more sub-node. From the example above we have three super node {A, B, and C}.

2. ***Sub-Node***

A sub-node is a vertices contained in a super node of a super graph. From the above example we have two sub-nodes is super node A that are {A1, A2}, likewise in super node B we have two sub-nodes {B1, B2}, and in super node C we have three sub-nodes {C1, C2, and C3}.

3. ***Connections in Super Graph***

Two super node V_1, V_2 with sub-nodes $v_i \in V_1, v_j \in V_2$ are said to be connected if there are at least one connection between their sub-nodes (v_i, v_j) we write $e = (v_i, v_j)$.

4. ***Types of Node's Degrees***

- Degree of super node: the number of edges coming out of a super node.
- Degree of sub-node: the number of edges coming out of a sub-node.
- Degree of sub-node with other super nodes: the number of edges connecting a sub-node to other super nodes. The value range between zero and the number of sub-nodes in the connected super node where zero means no connecting between the sub-node and the super node.

The relations between different types of degrees are as follows.

$$\text{Degree of super node } V = \sum_{i \in V} Degree\ of\ sub\text{-}node\ i$$

For example, in Figure 13 we have:

$$\text{Degree of super node } A = \sum_{sub\text{-}node\ Ai}^{2} Degree\ of\ sub\text{-}node\ Ai$$

$$\text{Degree of super node } A = Degree\ of\ sub\text{-}node\ A1 + Degree\ of\ sub\text{-}node\ A2$$

$$\text{Degree of super node } A = 4 + 2 = 6$$

C. Super Graph Adjacency Matrix

Super Graph Adjacency Matrix: Let $V_G = \{v_1 \ldots v_n\}$ be graph. The adjacency matrix of Super Graph *G* is the *n* × *k* matrix where *k* number of all super nodes and *n* number of all sup-nodes in all super nodes *M* with entries *with entries Mij = Degree of sub-node with other super nodes or Mij = (-) i.e. (undefined)*..

For instance, the Super Graph below has a Super Graph Adjacency matrix next to it.

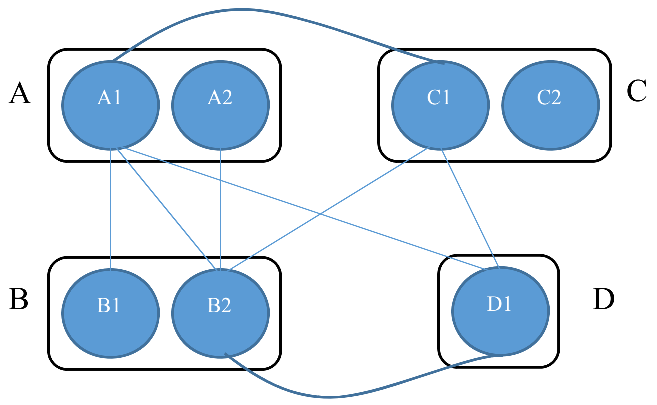


Figure 14 Super Graph G

Super Node	Sub-node	**A**	**B**	**C**	**D**	**Degree of sub-node**	**Degree of Super node**
A	**A1**	-	2	1	1	4	5
	A2	-	1	0	0	1	
B	**B1**	1	-	0	0	1	5
	B2	2	-	1	1	4	
C	**C1**	1	1	-	1	3	3
	C2	0	0	-	0	0	
D	**D1**	1	1	1	-	3	3

Table 2 Super Graph Adjacency Matrix of Super Graph G

From the above example we have $k = 4$ (k number of all super nodes {A, B, C, D}) and $n = 7$ (n number of all sup-nodes in all super nodes {A1, A2, B1, B2, C1, C2, D1}).

A Super Graph G has many different adjacency matrices, one for its sub-nodes vertices which is called (sub-nodes adjacency matrix), and another for the adjacency between sub-nodes and super nodes that is (Super Graph Adjacency Matrix).

The difference between the two adjacency matrixes is that the (Super Graph Adjacency Matrix) can reduce the preprocessing time needed be reducing the number of sub-nodes that will go into the (sub-nodes adjacency matrix).

For that we need to define a new type of sub-node degree the (degree of sub-node with other super nodes). By finding the (degree of sub-nodes with other super nodes) we can remove the sub-nodes with the degree of zero from the (sub-nodes adjacency matrix) which in turn reduces the dimensions of the sub-node adjacency matrix resulting in less processing time for finding solutions.

1. *Advantage of Using Super Graph Adjacency Matrix*

The previous example will act as the explaining vehicle to illustrate the benefit of using the super graph adjacency matrix.

The sub-node adjacency matrix for the above example can be constructed as **n × n** matrix where (*n* number of all sup-nodes in all super nodes) *M* with entries M_{ij} = 1, M_{ij} = 0 or M_{ij} = (-) i.e. (undefined) according to whether $v_iv_j \in$ Super Graph *G* or $v_iv_j \notin G$.

	A1	**A2**	**B1**	**B2**	**C1**	**C2**	**D1**
A1	-	-	1	1	1	0	1
A2	-	-	0	1	0	0	0
B1	1	0	-	-	0	0	0
B2	1	1	-	-	1	0	1
C1	1	0	0	1	-	-	1
C2	0	0	0	0	-	-	0
D1	1	0	0	1	1	0	-

Table 3 Sub-Node Adjacency Matrix

Now using the previous Super Graph Adjacency matrix we can exclude the sub-nodes with one or more zero occurrence in there degree with other super nodes.

From the super graph adjacency matrix we find that the following sub-nodes should be deleted from the sub-node adjacency matrix which are {A2, B1, and C2} because all of them has at least one zero degree in their super graph adjacency matrix.

The resulted sub-node adjacency matrix will be as follow:

	A1	**B2**	**C1**	**D1**
A1	-	1	1	1
B2	1	-	1	1
C1	1	1	-	1
D1	1	1	1	-

Table 4 Reduced Sub-Node Adjacency Matrix

The new sub-node adjacency matrix has reduced in dimension which reduce the search time required to find all the solutions.

CHAPTER IV

RESULTS

We applied our algorithm on the course offerings data provided form (AUB). Using the super graph and the super node adjacency matrix concepts we have developed a web based application that can be used by students or teachers to generate personal courses schedules tailored to their preferences. Also this application can be integrated with any web based student information system in most university very easily.

The data that was used in the test phase was a real course offering data provided by the registrar office in AUB.

A. Modified Vs Original Versions of Bron-Kerbosch Algorithm

Due to the relatively small number of selected courses (between 4-8 courses). And the increasing performance of today's computational power. We cannot see a huge difference between the modified and the original versions of Bron-Kerbosch algorithm. Although, we did a comparison that shows an enhancement in performance for the modified version over the original version of Bron-Kerbosch algorithm.

We have compared our modified version against the original one using the time required to generate all maximal cliques given the same date set. Although the difference is very small, but the modified version shows an enhancement over the original version of Bron-Kerbosch algorithm.

For example, we have tested the required time for generating all possible schedule given the same number of courses to both versions of the Bron-Kerbosch algorithm and we recorded the needed time which was in millisecond (ms) in the following table we show the result of the performance comparison while processing 2, 3, 4, 5, 6, 7 courses. Then we show the process time for each type of algorithm.

Number of Courses	Original Algorithm Time (ms)	Modified Algorithm Time (ms)	Number of Solutions
2	194	185	4
3	229	216	8
4	357	346	25
5	368	358	48
6	574	521	72
7	596	552	24

Table 5 Modified vs Original version of Bron-Kerbosch Algorithm

Figure 15 shows the results graphically. In Figure 15 we notice that both algorithms have a small difference when the number of input courses is small, when the number of courses used as input increases we can see the difference is increasing.

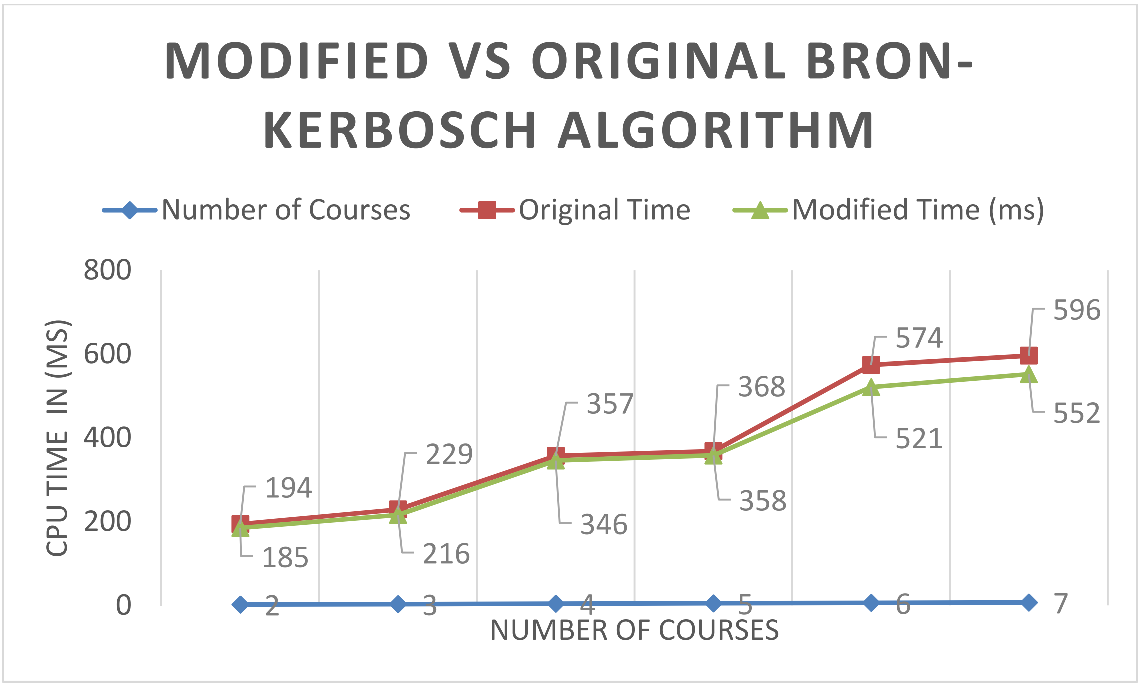


Figure 15 Modified vs Original version of Bron-Kerbosch Algorithm

B. Test Data

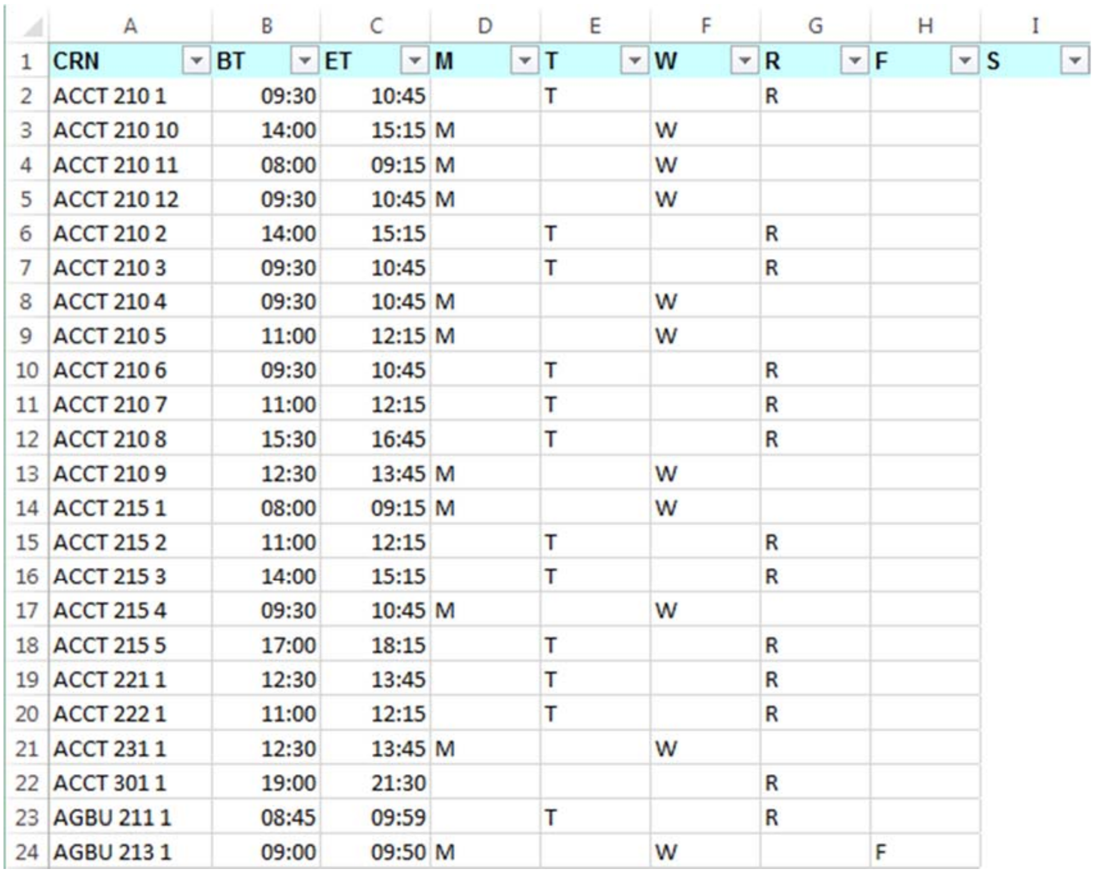
We have used real courses offerings data that was provided by the registrar office in AUB to guarantee the effectiveness of the proposed method and the speed of the algorithm in face of a huge real data.

1. *Course Offerings Data*

The provided test data was exported as an Excel file that contains 2515 record of course offerings that has the following columns:

- Course CRN which is a unique combination of characters and numbers that identifies a particular course and section offered for the term.
- Beginning Time (BT) which is the class start time of the course section that has a specific CRN.
- End Time (ET) which is the class end time of the course section that has a specific CRN.
- Week days which are the days that the class of the course section is giving on each week throughout the semester.

A sample of the course offerings data is showing in Table 6:



	A	B	C	D	E	F	G	H	I
1	CRN	BT	ET	M	T	W	R	F	S
2	ACCT 210 1	09:30	10:45		T		R		
3	ACCT 210 10	14:00	15:15	M		W			
4	ACCT 210 11	08:00	09:15	M		W			
5	ACCT 210 12	09:30	10:45	M		W			
6	ACCT 210 2	14:00	15:15		T		R		
7	ACCT 210 3	09:30	10:45		T		R		
8	ACCT 210 4	09:30	10:45	M		W			
9	ACCT 210 5	11:00	12:15	M		W			
10	ACCT 210 6	09:30	10:45		T		R		
11	ACCT 210 7	11:00	12:15		T		R		
12	ACCT 210 8	15:30	16:45		T		R		
13	ACCT 210 9	12:30	13:45	M		W			
14	ACCT 215 1	08:00	09:15	M		W			
15	ACCT 215 2	11:00	12:15		T		R		
16	ACCT 215 3	14:00	15:15		T		R		
17	ACCT 215 4	09:30	10:45	M		W			
18	ACCT 215 5	17:00	18:15		T		R		
19	ACCT 221 1	12:30	13:45		T		R		
20	ACCT 222 1	11:00	12:15		T		R		
21	ACCT 231 1	12:30	13:45	M		W			
22	ACCT 301 1	19:00	21:30				R		
23	AGBU 211 1	08:45	09:59		T		R		
24	AGBU 213 1	09:00	09:50	M		W		F	

Table 6 Course Offerings Data Sample

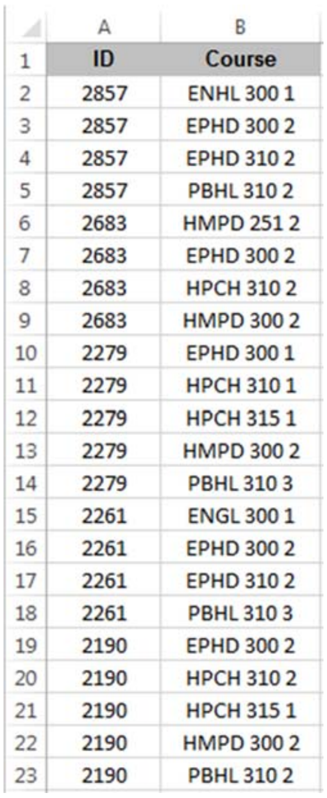
2. ***Validation Data***

The test data provided from AUB has also a validation data sample of about 100 case of real student course schedules without providing any personal information and after changing the student ID to different combination of numbers to keep the student identity anonymous.

The structure of the validation data excel file is as follow:

- ID which is the encrypted student ID for student identity protection.
- Course which is the CRN of the class of the course section that the student is registered in a given semester.

A sample of the validation data is shown in Table 7:



	A	B
1	ID	Course
2	2857	ENHL 300 1
3	2857	EPHD 300 2
4	2857	EPHD 310 2
5	2857	PBHL 310 2
6	2683	HMPD 251 2
7	2683	EPHD 300 2
8	2683	HPCH 310 2
9	2683	HMPD 300 2
10	2279	EPHD 300 1
11	2279	HPCH 310 1
12	2279	HPCH 315 1
13	2279	HMPD 300 2
14	2279	PBHL 310 3
15	2261	ENGL 300 1
16	2261	EPHD 300 2
17	2261	EPHD 310 2
18	2261	PBHL 310 3
19	2190	EPHD 300 2
20	2190	HPCH 310 2
21	2190	HPCH 315 1
22	2190	HMPD 300 2
23	2190	PBHL 310 2

Table 7 Validation Data Sample

C. Hard Constraints

In the proposed method for generating courses schedules, the course schedule needs to meet some constraints in order to be a possible solution for a user inquiry. Meeting these constraints result in a schedules where can be used in real world. These type of constraints guarantee a safe logic of the output of the proposed method. The hard constraints that are followed in our method are:

- No student can attend two courses having a time conflict
- Each section has a limitation on size
- No student can attend two sections of the same course

D. User Constraints (Soft Constraints)

The main advantage of our proposed method is providing user (student, teacher) the ability of finding the best course schedule that best fit their preferences. Our proposed method try to do that by creating virtual courses that have all the student preferences then trying to find the schedules that meet these choices.

1. *Virtual Course*

A virtual course is a constraint where user can set to reflect its preferences on the final result schedules. When a user prefer to put a constraints on something like day for instance we make a temporary class section which has the user preferences of the day and time and we add it to the pool of selected courses and then the algorithm would not generate any schedule that has any conflict between the real courses and the virtual ones which result in a conflict free schedules while meeting the user desired preferences at the same time.

Finally, all the virtual courses that are set by the user after generating the set of possible schedule will be removed i.e. these constraints (virtual courses) are user based.

a. Virtual Course Structure

Each virtual course must have the following items in it that must be set in order to be a valid virtual course:

- Virtual course name: it act as an identification ID for user to distinguish virtual courses if there are multiple ones.
- Virtual course days: it is the days of the week that this instance of virtual course will be held or active.

- Virtual course time: it is the time of the day in which the virtual course instance will be active.

An example for the virtual course is presented in Figure 16:

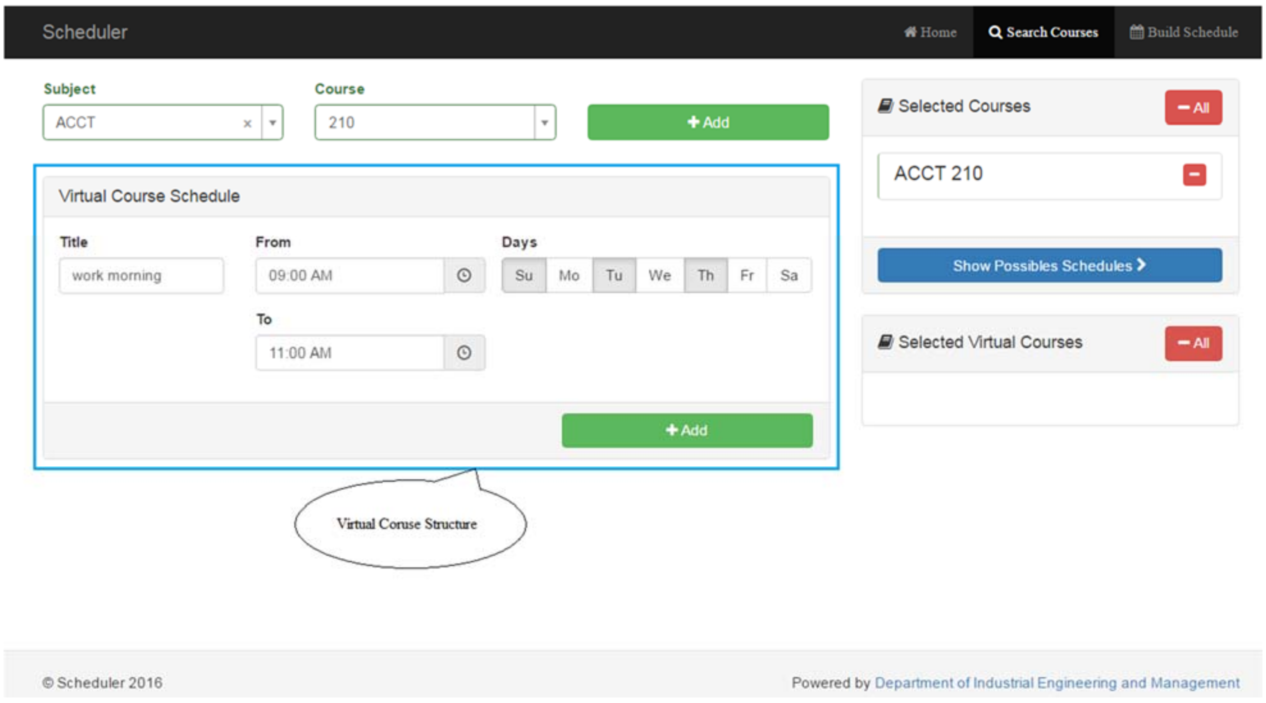


Figure 16 Virtual Course Structure

1. ***Examples on Method Validity***

a. <u>Example 1</u>

From the validation data in Table 7. We have used the registered courses by student number 2857 the registered courses are {ENHL 300 1, EPHD 300 2, EPHD 310 2, PBHL 310 2}. We only need the CRN without the section number which is the last number in the CRN. To validate the proposed method the results should contain one schedule that matched the schedule of student number 2857.

After selecting the desired courses in the Search Course screen we generate the result schedules and we have 18 possible solutions where the schedule number 4 is the same as student number 2857 schedule shown in the following screenshot.

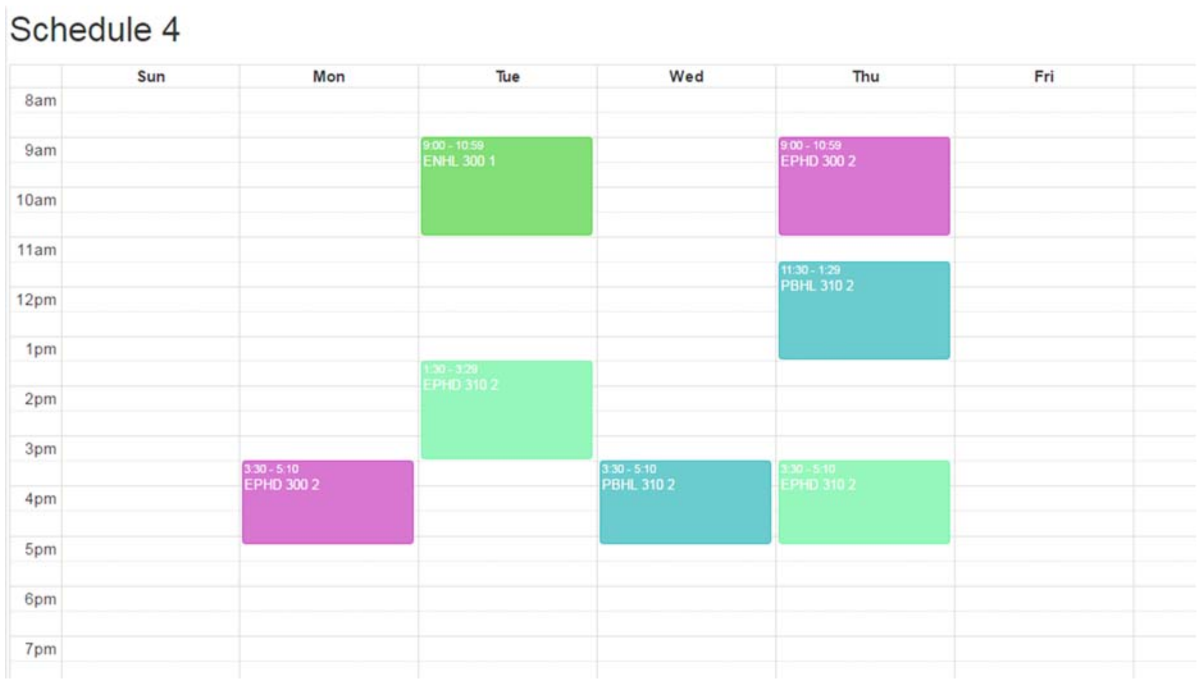


Figure 17 Example 1 Schedule

b. <u>Example 2</u>

From validation data in Table 7. We have used the registered courses by student number 2683 the registered courses are {HMPD 251 2, EPHD 300 2, HPCH 310 2, HMPD 300 2}. To validate the proposed method the results should contain one schedule that matched the schedule of student number 2683.

After selecting the desired courses in the Search Course screen we generate the result schedules and we have 8 possible solutions where the schedule number 8 is the same as student number 2683 schedule shown in the following screenshot.

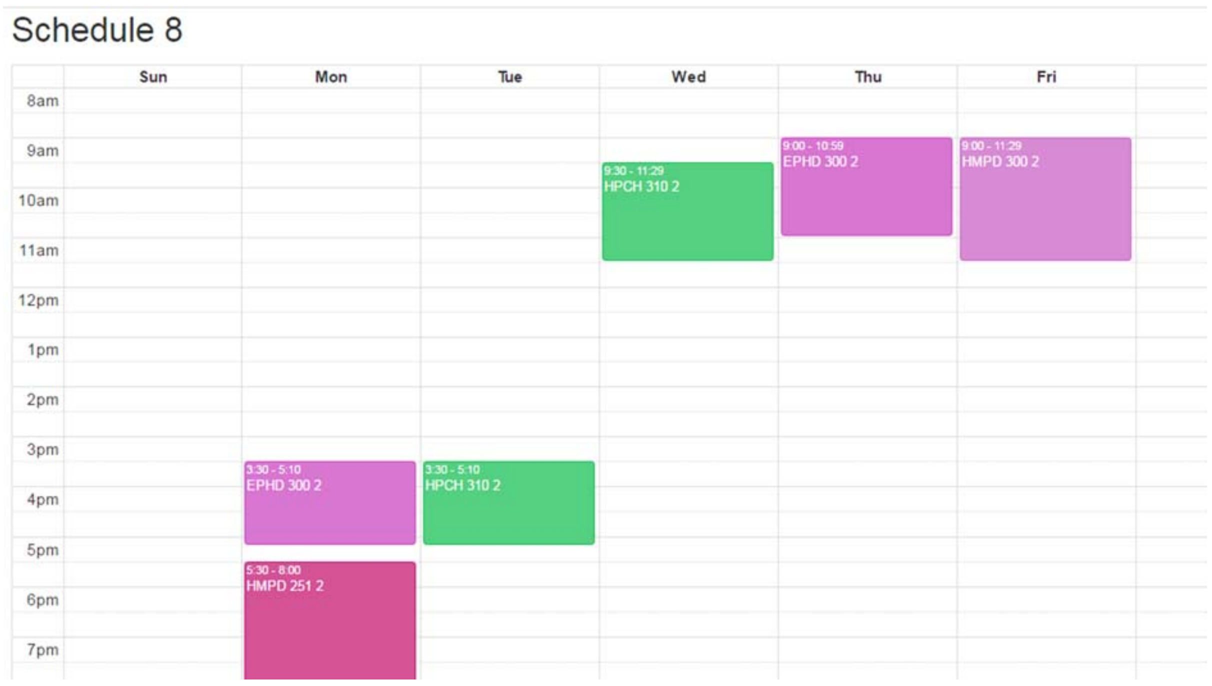
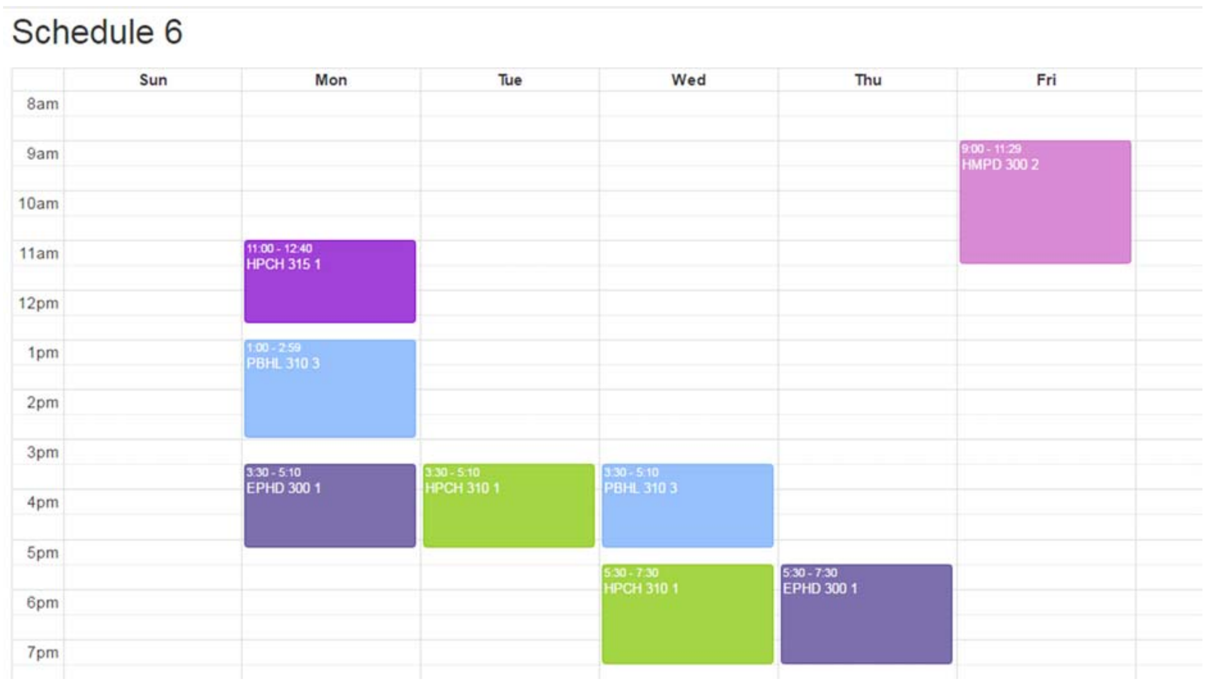


Figure 18 Example 2 Schedule

c. <u>Example 3</u>

From validation data in Table 7. We have used the registered courses by student number 2279 the registered courses are {EPHD 300 1, HPCH 310 1, HPCH 315 1, HMPD 300 2, PBHL 310 3}. To validate the proposed method the results should contain one schedule that matched the schedule of student number 2279.

After selecting the desired courses in the Search Course screen we generate the result schedules and we have 48 possible solutions where the schedule number 6 is the same as student number 2279 schedule shown in the following screenshot.


Schedule 6

	Sun	Mon	Tue	Wed	Thu	Fri
8am						
9am						9:00 - 11:29 HMPD 300 2
10am						
11am		11:00 - 12:40 HPCH 315 1				
12pm						
1pm		1:00 - 2:59 PBHL 310 3				
2pm						
3pm		3:30 - 5:10 EPHD 300 1	3:30 - 5:10 HPCH 310 1	3:30 - 5:10 PBHL 310 3		
4pm						
5pm				5:30 - 7:30 HPCH 310 1	5:30 - 7:30 EPHD 300 1	
6pm						
7pm						

Figure 19 Example 3 Schedule

CHAPTER V

CONCLUSION & FUTURE WORK

A. Conclusion

In this work a practical, real-word problem of student personal schedule has been addressed. Most of the students in each university manually go through hard time preparing their personal course schedule every semester. Instead of wasting time in creating the course schedule, students can spend more time and effort on something else. Solving the student personal schedule problem the hard constraints were not the only constraints considered. All soft constraints were also considered. Results therefore shows a feasible solution to the problem that can be used in real life.

B. Future Work

- Addressing the issue of having consecutive courses in the resulted schedules as a soft constraint.
- Enable the student of defining number of courses per day in the generated schedules.
- Integrating our application with existing student management systems like the Banner in AUB.
- Enable the student to register the selected schedule automatically to save time.

- Enhancing the algorithm so it provides the first n-feasible solutions only without traversing over all solutions to save time if a student prefers.

CHAPTER VI

APPENDIXE

A. Application Usage Steps

1. *Step 1 Application Dashboard*

In Figure 20 a student can choose to search for courses (Search Courses) or to edit already build schedules (Build Schedule).

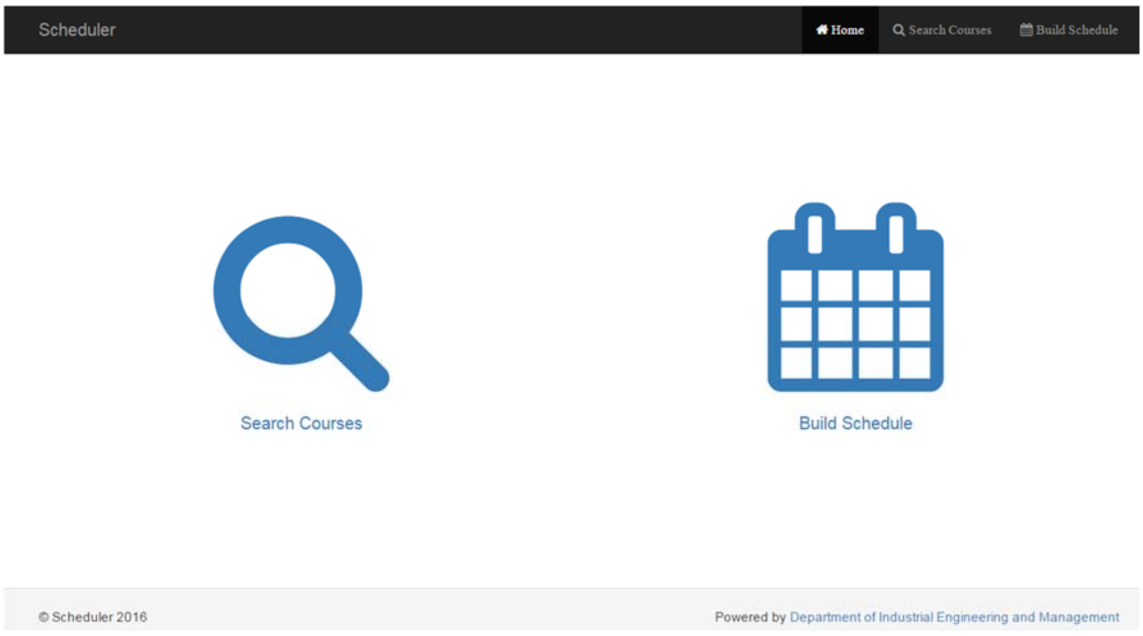


Figure 20 Application Dashboard

2. *Step 2 Application Search Screen (Subject Selection)*

In Figure 21 a student can choose the required course subject or the course department.

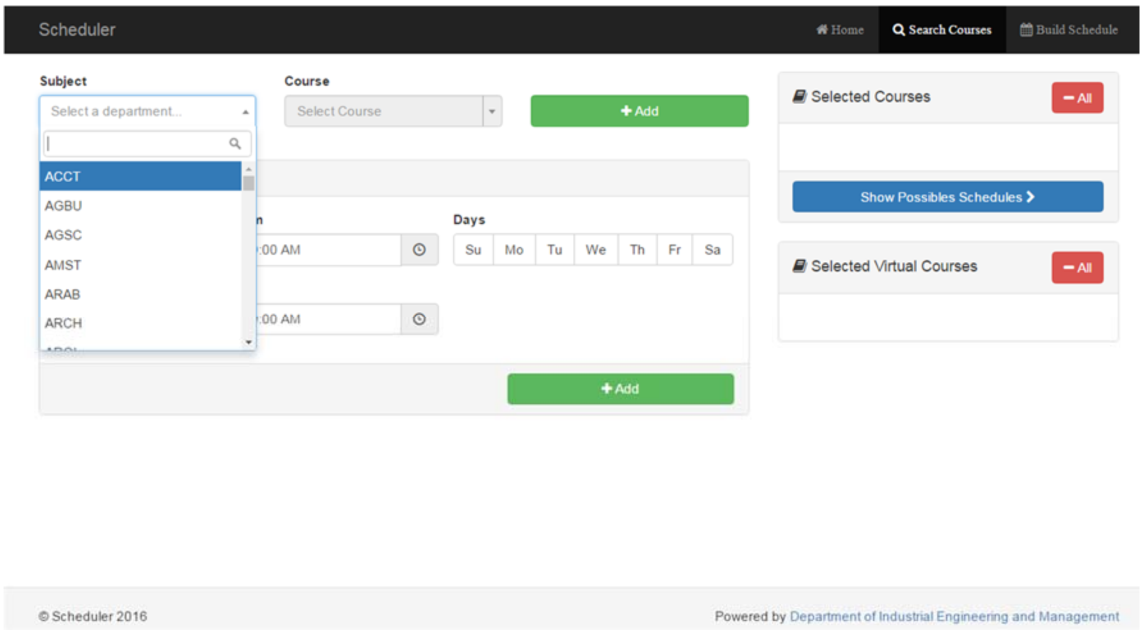


Figure 21 Application Search Screen (Subject Selection)

3. *Step 3 Application Search Screen (Courses Selection)*

In Figure 22 a student can select any course of the previously subject.

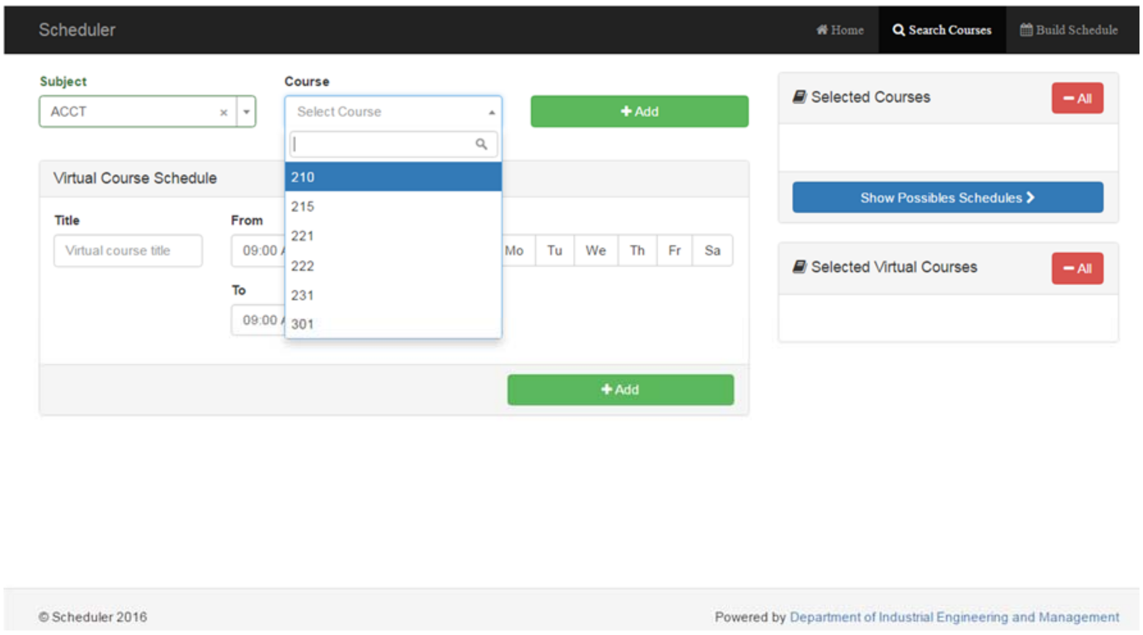


Figure 22 Application Search Screen (Course Selection)

4. ***Step 4 Application Search Screen (Adding Selected Course)***

In Figure 23 a student can add the previously selected course to its set of selected courses.

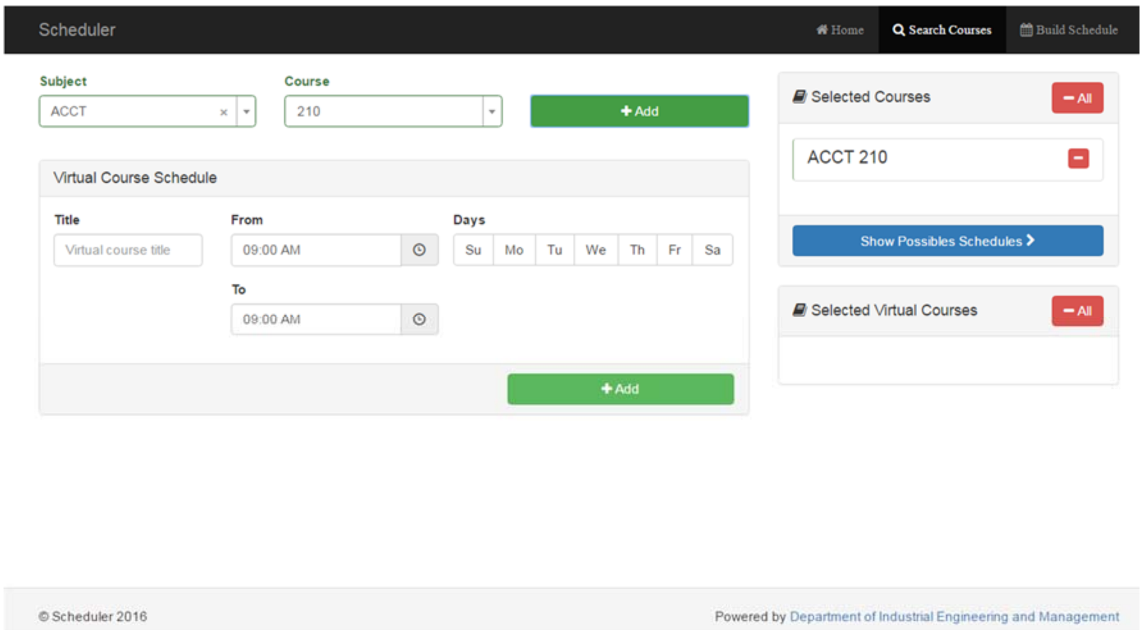


Figure 23 Application Search Screen (Adding Selected Course)

5. ***Step 5 Application Search Screen (Virtual Course)***

In Figure 24 a student can (optionally) create his preferences as a virtual course where he need to input a name, beginning time, end time, and days for the desired virtual course. In Figure 25 we can see a created virtual course named (work morning) with beginning time (09:00 AM) and end time (11:00 AM), and days (Su, Tu, Th).

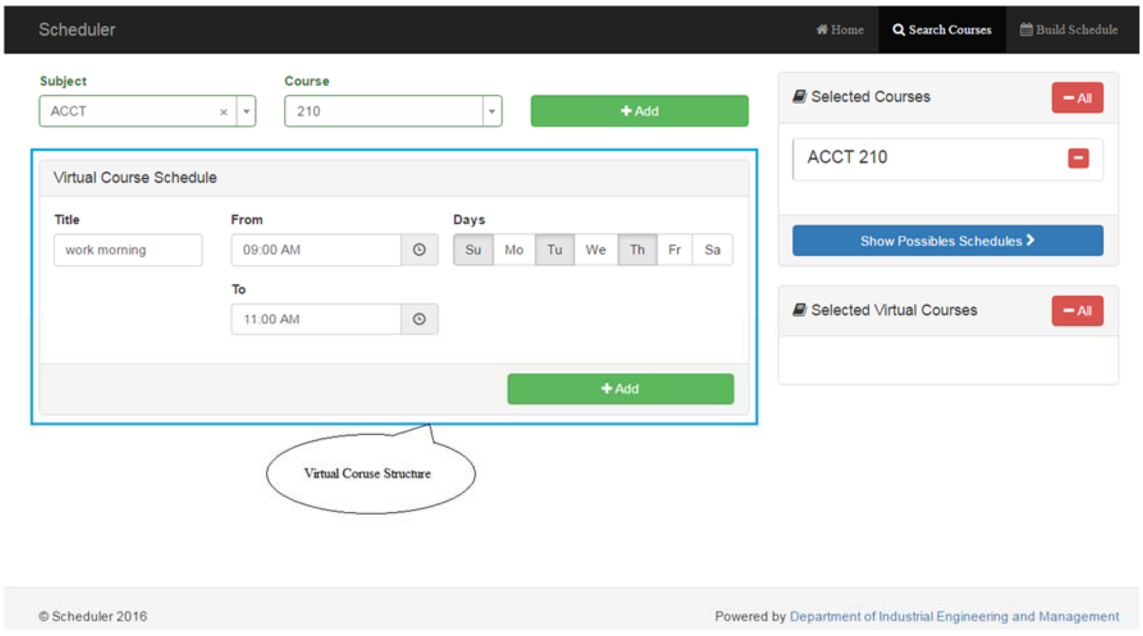
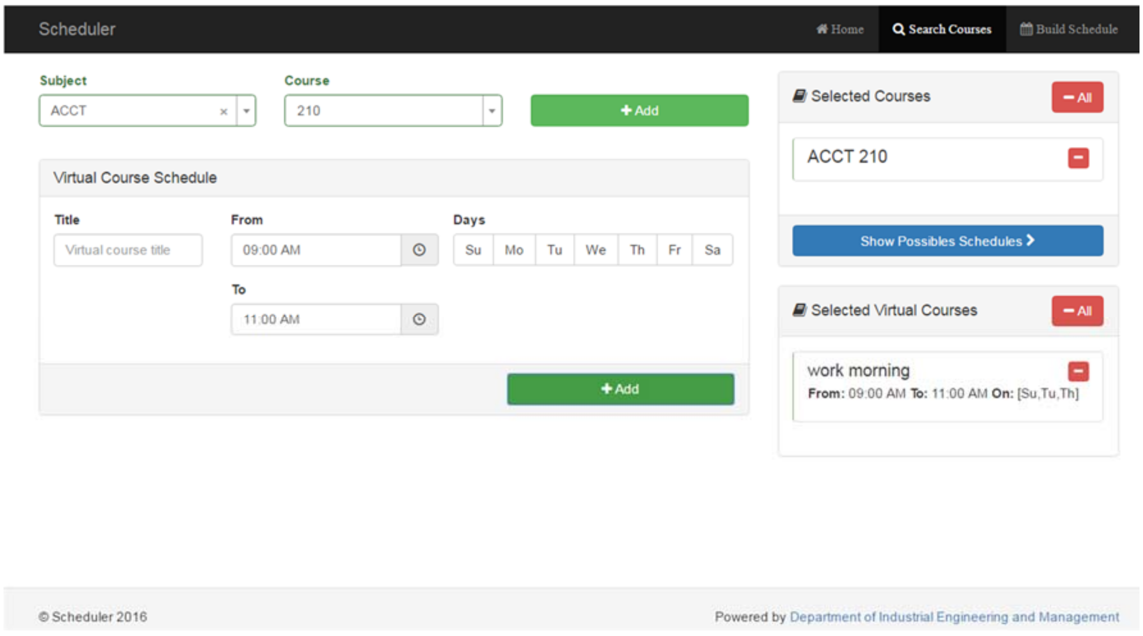


Figure 24 Application Search Screen (Creating Virtual Course)



work morning
From: 09:00 AM **To:** 11:00 AM **On:** [Su,Tu,Th]

Figure 25 Application Search Screen (Adding Virtual Course)

6. ***Step 6 Application Result Screen***

In Figure 26 a student can choose one of the alternative schedules to register in. For example, in Figure 26 we see 9 possible schedules that a student can choose from.

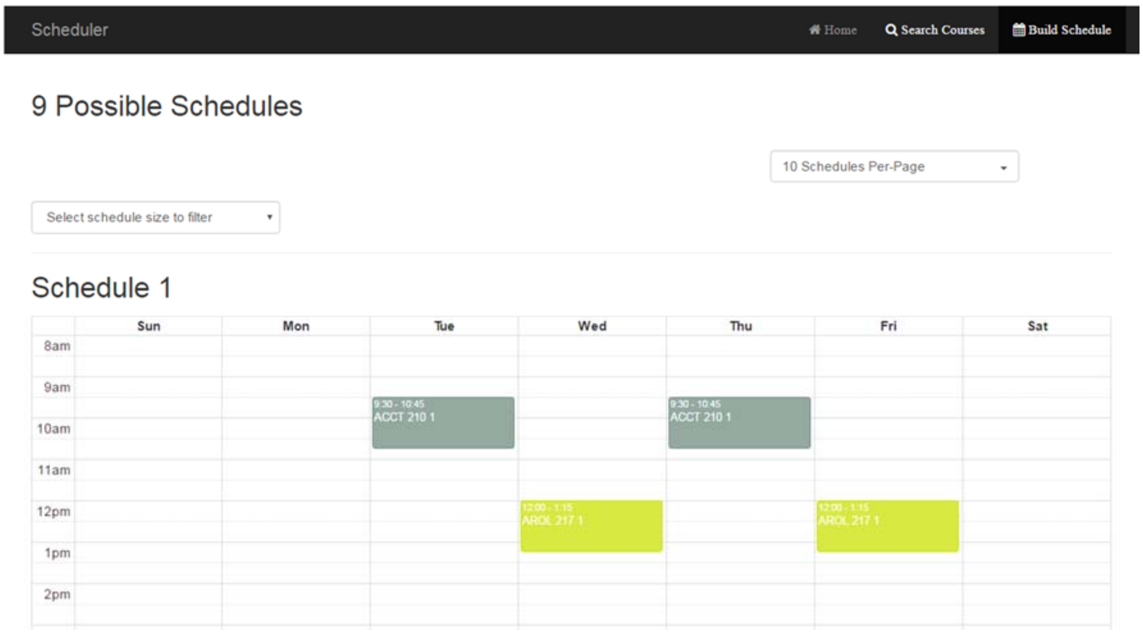


Figure 26 Application Result Screen

REFERENCES

Schroeder, Roger G. "A survey of management science in university operations." Management Science 19, no. 8 (1973): 895-906.

White, Gregory P. "A survey of recent management science applications in higher education administration." Interfaces 17, no. 2 (1987): 97-108.

Wren, Anthony. "Scheduling, timetabling and rostering—a special relationship?" In Practice and theory of automated timetabling, pp. 46-75. Springer Berlin Heidelberg, 1995.

Lewis, Rhydian, Ben Paechter, and Barry McCollum. *Post enrolment based course timetabling: A description of the problem model used for track two of the second international timetabling competition*. Cardiff Business School, 2007.

Di Gaspero, Luca, Barry McCollum, and Andrea Schaerf. *The second international timetabling competition (ITC-2007): Curriculum-based course timetabling (track 3)*. Technical Report QUB/IEEE/Tech/ITC2007/CurriculumCTT/v1. 0, Queen’s University, Belfast, United Kingdom, 2007.

Carter, Michael W. "A comprehensive course timetabling and student scheduling system at the University of Waterloo." In Practice and theory of automated timetabling III, pp. 64-82. Springer Berlin Heidelberg, 2000.

Azlan, Amirah, and Naimah Mohd Hussin. "Implementing graph coloring heuristic in construction phase of curriculum-based course timetabling problem." In *Computers & Informatics (ISCI), 2013 IEEE Symposium on*, pp. 25-29. IEEE, 2013.

Schaerf, Andrea. "A survey of automated timetabling." *Artificial intelligence review* 13, no. 2 (1999): 87-127.

Qu, R., Burke, E. K., Mccollum, B., Merlot, L. T., & Lee, S. Y. (2009). A survey of search methodologies and automated system development for examination timetabling. Journal of Scheduling, 12(1), 55-89.

Rahman, Syariza Abdul, Andrzej Bargiela, Edmund K. Burke, Ender Özcan, Barry McCollum, and Paul McMullan. "Adaptive linear combination of heuristic orderings in constructing examination timetables." *European Journal of Operational Research* 232, no. 2 (2014): 287-297.

Feldman, Ronen, and Martin Charles Golumbic. "Constraint Satisfiability Algorithms for Interactive Student Scheduling." In IJCAI, pp. 1010-1016. 1989.

Kingston, Jeffrey H. "Educational timetabling." In *Automated Scheduling and Planning*, pp. 91-108. Springer Berlin Heidelberg, 2013.

Carter, Michael W. "A comprehensive course timetabling and student scheduling system at the University of Waterloo." In *Practice and theory of automated timetabling III*, pp. 64-82. Springer Berlin Heidelberg, 2000.

Sönmez, Tayfun, and M. Utku Ünver. "Course bidding at business schools."*International Economic Review* 51, no. 1 (2010): 99-123.

Müller, Tomas, and Keith Murray. "Comprehensive approach to student sectioning." *Annals of Operations Research* 181, no. 1 (2010): 249-269.

R. Feldman and M. C. Golumbic. Optimization algorithms for student scheduling via constraint satisability. The Computer Journal, 33(4):356{364, 1990.

Scott E. Sampson and Elliott N. Weiss. Increasing service levels in conference and educational scheduling: A heuristic approach. Management Science, 41(11):1816{1825, 1995.

Chachra, Vinod, Prabhakar M. Ghare, and James Mendon Moore. "Application of graph theory algorithms." *NORTH-HOLLAND PUBL. CO., NY, 1979, 422* (1979).
Wikipedia. Seven Bridges of Königsberg. Online, accessed 9 December 2015. URL https://en.wikipedia.org/wiki/Seven_Bridges_of_K%C3%B6nigsberg.
Bron, Coenraad, and Joep Kerbosch. "Finding all cliques of an undirected graph (algorithm 457)." Commun. ACM 16, no. 9 (1973): 575-576.

Framinan, Jose M., and Rubén Ruiz. "Architecture of manufacturing scheduling systems: Literature review and an integrated proposal." European Journal of Operational Research 205, no. 2 (2010): 237-246.

Chachra, Vinod, Prabhakar M. Ghare, and James Mendon Moore. "Application of graph theory algorithms." NORTH-HOLLAND PUBL. CO., NY, 1979, 422 (1979).