DEVELOPMENT OF EXAMINATIONS SCHEDULING ALGORITHM USING GRAPH COLOURING

A Case of Sokoine University of Agriculture

Mohamed Abdallah Selemani

M.Sc. (Mathematical Modelling) Dissertation
University of Dar es Salaam
April 2012

DEVELOPMENT OF EXAMINATIONS SCHEDULING ALGORITHM USING GRAPH COLOURING

A Case of Sokoine University of Agriculture

By

Mohamed Abdallah Selemani

A Dissertation Submitted in (Partial) Fulfillment of the Requirement for the Degree of Masters of Science (Mathematical Modelling) of the University of Dar es Salaam

University of Dar es Salaam
April 2012

CERTIFICATION

The undersigned certifies that they have read and hereby recommend for examination a dissertation entitled: ***Development of Examinations Scheduling Algorithm: A Case of Sokoine University of Agriculture***, in partial fulfillment of the requirements for the degree of Master of Science (Mathematical Modelling) of the University of Dar es Salaam.

.....

Dr. E. Mujuni

(Supervisor)

Date: 18-04-2012

.....

Dr. A. R. Mushi

(Supervisor)

Date: 18/04/2012

DECLARATION
AND
COPYRIGHT

I, **Mohamed Abdallah Selemani**, declare that this dissertation is my own original work and that it has not been presented and will not be presented to any other University for a similar or any other degree award.

Signature .....

ACKNOWLEDGMENT

First and foremost I would like to thank Almighty God for keeping me healthy and taking me through my course successfully. I will not attempt the impossible task of mentioning everyone who contributed to make this work a reality, even though each of them deserves special mention. It would be unseeingly, however, not to list few names.

Special thanks to my supervisors Dr. E. Mujuni and Dr. A. Mushi, who generously and tirelessly extended their expertise through advice, criticism, guidance and encouragement, without which this report would not reach this stage, May the Almighty God bless them abundantly.

I am indebted to Management of SUA for making my study possible at the university of Dar es salaam, for their financial support under the Staff Development Program. I am grateful to the Head of Mathematics Department, Prof. ' E. S. Massawe, Programme coordinator, Dr. W. Mahera and other staff members for their valuable advice, support and encouragement during my course.

Much gratitude is extended to my colleagues Halid Lyeme, Michael Mkwizu and Janeth Michael who I worked with tirelessly in times of trouble and joy in completion of this report. Indeed, I am greatly indebted to my beloved parents for their tolerance and understanding during my absence in family matters. The encouragement and moral support continuously refueled my inspiration throughout my study.

Finally, I confirm that none of these individuals bear any direct responsibility for the opinions expressed here in. Any technical errors and weakness that may arise from this report are entirely mine. I take full responsibility and all others absolved.

DEDICATION

I dedicated to my lovely wife Awetu Japhery Millanzi and my son Farhad.

I love you all.

ABSTRACT

Examination Timetabling Problem (ETP) is a real life problem encountered in academic institutions and attracted the attention of reseach communities since 1960's. This study intends to develop a graph based algorithm for ETP at Sokoine University of Agriculture (SUA).

In this study, a Recursive Largest First (RLF) algorithm for graph colouring is used to solve the ETP at SUA. We discuss in detail the step-by-step process that is taken to implement our timetabling-by-graph-coloring procedure, from the assembling of university exam data, to creating an exam conflict graph based on the assembled data, to colouring the conflict graph, to transforming this colouring to a conflict-free timetable, to finally assigning exams to classrooms.

Using the two semester data sets from the case study, computational experiments are conducted based on the developed algorithm and obtained the promising results. Sample Computational results, using actual data provided by students' records office of SUA, are documented.

Our algorithm is very flexible, since it allows the user to define the basic problem data which are courses to examined, courses' enrolment, rooms and their respective capacities.

Its performance is quite satisfactory, considering that it is an algorithm that has to run twice a year for the construction of the examination timetable of a specific educational organization. Some improvements on it such as applying metaheuristics approach with it, will certainly enhance the algorithm's functionality.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER ONE
INTRODUCTION

1.1 General Introduction.

The timetabling problem (TTP) is a typical scheduling problem that appears in every academic institution at least once a year. The problem involves the scheduling of courses or examinations, students and lecturers at a fixed number of time slots and classrooms, subject to a certain number of constraints. The nature of the constraints varies between different instances of the timetabling problem. For example, a student should not be assigned to attend two or more courses simultaneously. In general these constraints are classified into two categories: namely, hard and soft (see, e.g., Burke *et al*, 2004; Mushi, 2004). Hard constrains are conditions that must be satisfied in order to produce legal or feasible timetable, such as avoiding examination collision and room over-sizing. Soft constraints may be tolerable but must be minimized as much as possible. An example of soft constraint is scheduling examinations with the largest number of students earlier.

The university timetabling problem can be grouped into two categories: course (or lecture) timetabling and examination timetabling. These two timetabling problems are fairly similar in some superficial ways, but indeed, the examination timetabling problem differs significantly from course timetabling problem. For example, examination timetable extends to several weeks while course timetable must fit within a week, and repeating throughout the semester. Moreover, in university examinations timetabling it is often considered undesirable for students to have to sit for examinations in two consecutive periods, while in university course timetabling it is often preferred for students to have two or three lectures in consecutive periods.

The University Examination Problem has attracted significant research interest over the years. Despite of more than four decades of research, no polynomial-time algorithm is known to the timetabling problem. This problem is known to be NP-hard (Burke *et al*, 2004; El-mohamed *et al*, 1998). Therefore, according to complexity theory, it is very unlikely to find an optimal algorithm to solve this problem with large input size within reasonable time.

Because of the importance of this problem, from both practical and theoretical point of view, numerous studies have been conducted (see. e.g., Abdullah, 2006; Burke *et al*, 1994; Mushi, 2004). The main difference between various studies is the set of assumptions and constraints taken into consideration. Due to the complexity of the problem, most of these studies concentrates on the heuristic algorithms which try to find an approximate solution. Some of these include Tabu Search (White and Xie, 2001), Simulated Annealing (Mushi, 2007; El-mohamed *et al*, 1998), Graph Colouring Algorithm (Malkawi *et al*, 2008; Reid, 2009) and Integer Programming Model (Mushi, 2004).

In this study, we develop a graph-colouring based algorithm for university examinations scheduling at Sokoine University of Agriculture.

1.2 Statement of the Problem

The construction of an examination timetable is a common problem for all institutions of higher education. Quite often it is done with the limited help of a simple administration system, and usually involves taking the previous year's timetable and modifying it. As the difficulty of the problem increases, due to a large number of students, courses, exams, rooms and invigilator constraints, an automated timetabling system that can produce feasible and high quality timetables is often required (Cowling, et al, 2002). Like any other university in the world, SUA faces the problem of scheduling both courses and examinations in

every semester. Meanwhile, the commercial software called CELCAT is used to facilitate timetabling at SUA. However, this software does not make a timetable rather it gives a base for search and provides the choice of the possible solutions (Mkandawile, 2004). This implies that timetabling is time consuming and tiresome activity. Therefore this research intends to formulate a graph-colouring based algorithm for an automation of examination scheduling at SUA.

1.3 Current Examination Timetable at SUA

At SUA, the examination period is currently set to be three weeks with two examination sessions per day. Each week consists of five days i.e. Monday to Friday, making a total of 30 exam periods. Despite of the long exam period, in existing exam timetable, the exams for most of groups are not evenly distributed over this exam period. For example, it may happen for a candidate to seat for ten (10) exams within three weeks in the following ratios; 1:5:4 or 1:4:5 instead of 3:4:3 or 4:3:3. Also, there is inefficient of room allocation because in some exams sessions some rooms are not used while some candidates have exams in those particular exam sessions.

As stated in 1.2, currently, a software package called CELCAT is used to detect collisions, but the timetable has to be created manually. This task takes two up to three weeks, and still we cannot be sure that the resulting timetable is the best possible. Our goal is to develop an algorithm which will reduce this period by automatically generating a timetable which can then be imported into the timetable software for other changes where necessary.

SUA is composed of four faculties, each having various departments under it. Courses are offered by each department for students. Generally, courses within the university are designated as compulsory or optional. In most cases, students are permitted to "borrow" approved courses from other departments within

and/or outside their faculty in order to make up the required minimum number of course units. Therefore, there are some courses which are offered to more than a thousand students at the same time.

A room can be assigned more than one examination during the same timeslot depending on the availability of space. On the other hand, an examination that has large number of candidates than a single room can hold, are split into more than one room depending on the size of the course and capacity of available space. Allocation of invigilator is done by department after the release of timetable. The constraints applied in this study are as follows ;
Hard Constraints are:,

- No student may have two examinations in the same timeslot.
- At each timeslot, the university seat capacity must be respected.

Soft Constraints are:,

- No student(s) should have to take two examinations in adjacent time slots.
- No student(s) should have to take two examinations on the same day.
- Examinations with large number of candidates should be scheduled earlier.

1.4 Research Objectives

The general objective of this study is to provide a mechanism for automatic examination-schedule generation that achieves fairness, and minimizes the examination period at the SUA. In particular the specific objectives are:

i. To develop and analyze a graph-colouring based algorithm for examination scheduling for SUA.

ii. To suggest a better examination timetable at SUA.

1.5 Significance of the Study

The goal of the examination timetabling process can be taken to be that of producing the feasible timetable of the highest possible quality. The scheduling of examinations in higher learning institutions is known to be highly constrained problem. Many universities are increasing number of student enrolments into a wider variety of courses and an increasing number of combined degree courses. This is contributing to the growing challenge of constructing examination timetable within a short limited period of time. The use of software (CELCAT) for searching purpose has been weakened by the increase in complexity of Examination Timetabling Problem. Upon completion of this study, the formulated algorithm will be used to produce an examination timetable in a shorter period of time. Also the study will add more knowledge to the existing literature and will act as support for further research on examination timetabling.

1.6 Dissertation Organization

This dissertation consists of five chapters. Chapter 1 presents the general introduction, statement of problem, objectives, hypotheses and significance of the study. The remainder of this dissertation is organised in the following way:
Chapter 2 introduces the TTP and gives a brief literature review the problem. Basic graph theoretical concepts, complexity theory and some common methods on solving TTP are presented. Methods included are integer programming (IP), tabu search (TS), simulated annealing (SA) and Graph Based algorithms. Chapter 3 describes the methodology for solving our problem, and how it can be used to solve the problem. Analysis and Computational results are presented in chapter 4. Chapter 5 gives some conclusions and direction for future research.

CHAPTER TWO
LITERATURE REVIEW

2.1 Introduction

This chapter presents a brief review on some techniques used in solving timetabling problems especially with regard to the examination timetable. Various approaches have been presented in solving the problem. Morever, we present the basics of complexity theory and graph colouring problem for comprehensive purpose of our problem.

2.2 Graph Theoretical Definitions

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* or simply *graph*. Our discussions in this study are concerned with undirected graphs.

We use the symbols $v_1, v_2, v_3, \ldots$ to represent the vertices and the symbols $e_1, e_2, e_3, \ldots$ to represent the edges of a graph. The vertices v_i and v_j associated with an edge e_k are called the *endpoints* of e_k. The edge e_k is then denoted as $e_k = v_i v_j$ or $e_k = (v_i, v_j)$.

An edge is said to be *incident* on its endpoints. Two vertices are *adjacent* if they are the endpoints of an edge. The *neighborhood* of v, $N(v)$, is the set of vertices adjacent to v.

For example, in the Figure 2.1 edge e_1 is incident on vertices v_1 and v_2; vertices

v_1 and v_3 are adjacent while v_7 and v_8 are not. An *independent set* of vertices in a graph is a set of mutually non-adjacent vertices. The number of edges incident

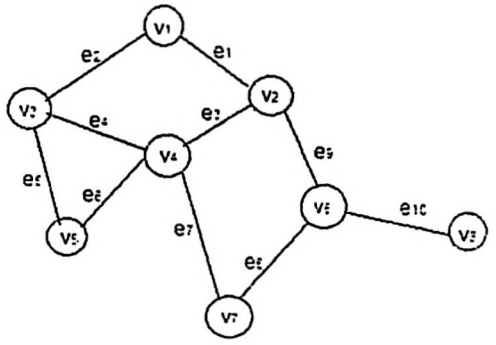


Figure 2.1: A Simple undirected graph

on a vertex v_i is called the *degree* of the vertex, and it is denoted by $deg(v_i)$. e.g. from figure 4.1 above, $deg(v_1) = 2, deg(v_2) = 3, deg(v_4) = 4$ and $deg(v_8) = 1$.

A graph G_1 is said to be a *subgraph* of G if $V(G_1) \subseteq V(G)$ and $E(G_1) \in E(G)$. A subgraph G_1 of a graph G is said to be a *proper subgraph* of G if either $V(G_1) \neq V(G)$ or $E(G_1) \neq E(G)$. A subgraph H of G is said to be an *induced subgraph* of G if each edge of G having its ends in $V(H)$ is also an edge of H. The induced subgraph of G with vertex set $S \subseteq V(G)$ is called the *subgraph of G induced* by S.

A *coloring* of G is a map $f : V(G) \rightarrow C$, where C is a set of distinct colors; it is *proper* if adjacent vertices of G receive distinct colors of C; that is, if $uv \in E(G)$ then $f(u) \neq f(v)$. Thus the *chromatic number*, $\chi(G)$ is the minimum number of colours for which there exists a proper coloring of G by colours of C. Clearly, in any proper colouring of G, the vertices that receive the same colour are independent. The vertices that receive a particular colour make up a *colour class*. Thus, in any chromatic partition of $V(G)$, the parts of the partition constitute the colour classes. This allows an equivalent way of defining the chromatic number (Butenko, 2001).

Definition 1 *The **chromatic number** of a graph G is the minimum number of colours needed for a proper colouring of G. If $\chi(G) = k$, G is said to be k-chromatic. A $k-$colouring of a graph G is a vertex colouring of G that uses k colors. A graph G is said to be $k-$colorable, if G admits a proper vertex colouring using k colours.*

For example, a graph in figure 2.2 is $3-$*colorable*, has three colour classes which are $V_1 = \{v_1, v_4, v_8\}, V_2 = \{v_2, v_5, v_7\}$ and $V_3 = \{v_3, v_6\}$

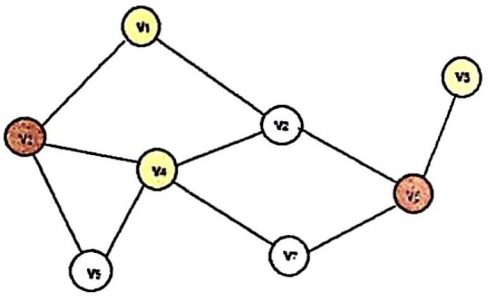


Figure 2.2: A 3-colouring of the graph in Figure 2.1

The graph k-colouring problem asks "given a graph G and an integer k, does G have a k-colouring?"

2.3 Complexity Theory

Complexity theory is a part of the theory of computation dealing with the resources required during computation to solve a given problem. It classifies decision problems based on how difficult they are to solve. A problem is considered computationally hard if it uses an unreasonable amount of time, space or another resource to the problem. A problem is considered to be easy if it can be solved using little resources (Sloper, 2001).

Definition 2 *The running time of an algorithm is the maximum number of computational steps used, usually expressed as $O(f)$, where f is a function of input size.*

Definition 3 *The complexity of decision problem is the minimum worst-case running time over all solution algorithms.*

In time complexity we try to classify the problem according to the number of steps they need for their solution. A decision problem is a problem having only two solutions, namely; "yes" or "no". The class P is a class of decision problems that can be decided in polynomial computational time. Every decision problem in P is considered to be tractable, that is, easily solvable. The class NP is another major class in the complexity theory. It is the class of decision problems that can be verified by a polynomial-time algorithm. That is, given a solution of a problem in NP, then we can verify the solution in polynomial time. A problem X is said to be NP-hard if every problem $A \in NP$ can be reduced to X in polynomial time. The class of NP-complete problems is a class of problems that are both NP and NP-hard (Großmann, 2005; Sloper, 2001).

NP-complete problems are the hardest problems in the NP class. Despite of more than forty years of research, no polynomial-time algorithm has yet been discovered for an NP-complete problem, and no one has proven that no polynomial-time algorithm can exist for any NP-complete problem (Sloper, 2001).

2.4 Graph Colouring Problem

The problem of finding the minimum number of colours such that a graph has proper colouring is called the *Graph Colouring Problem (GCP)*. GCP is known to be an NP-hard problem (Salari and Eshghi, 2008). It has many practical applications such as scheduling, frequency assignment in cellular networks, timetabling,

etc (Al-Omari and Sabri, 2006; Rothe, 2000). There are two classes of algorithms to solve this problem; namely *exact* and *approximate* algorithms. Since it has been proved that the GCP belongs to the class of NP-hard problems, exact algorithms (Lucet *et al*, 2005) are confined to solve small size instances of not more than 100 vertices (Brelaz, 1979) and as the problem size increases, the use of this class of algorithms quickly becomes infeasible. Therefore, the only possibility is to resort to approximate algorithms in order to obtain near-optimal solutions at relatively low computational time.

Due to the difficulty of the GCP, many researchers have tried to find efficient algorithms that provide approximate solutions. This includes heuristic algorithms. The term *heuristic* is used for algorithms which find solutions among all possible ones, but they do not guarantee that the best will be found, therefore they may be considered as approximately and not accurate algorithms. These algorithms, usually find a solution close to the best one and they find it fast and easily. Sometimes these algorithms can be accurate, that is they actually find the best solution, but the algorithm is still called heuristic until this best solution is proven to be the best.

2.5 Timetabling Problem

Timetabling problem (TTP) is an NP-hard optimization problem that involves the allocation of certain resources, subject to constraints, into a limited number of timeslots with the aim of satisfying a set of stated objectives to the highest possible extent (Wren, 1996 cited in Adewumi et al, 2009). One of key applications of timetabling is in educational timetabling. Schaerf (1999) classified educational timetabling into three classes; *school timetabling , course timetabling and examination timetabling.* These problems share the same basic characteristics of the general TTP but can still have significant differences between them. Each one of

them has its own constraints, requirements and rules (Abdullah. 2006). In the following subsections we briefly describe these problems

2.5.1 School Timetabling Problem (STP)

This problem tries to assign lessons to timeslots in such a way that no teacher or class is involved in more than one lecture at a time, and satisfying a set of other given constraints in order to produce a feasible timetable. Common examples of constraints in the STP are room capacities, locations, teacher loads, break time between two lessons and other personal preferences. de Werra (1985) presented a simple mathematical model for solving STP as follows: Given a set $C = \{c_1, c_2, \ldots, c_m\}$ of m courses, a set $T = \{t_1, t_2, \ldots, t_n\}$ of n teachers, a set $\{1, 2, \ldots, p\}$ of p periods and requirement matrix $R_{m \times n} = r_{ij}$ where r_{ij} is the number of lectures involving class c_i and teacher t_j.. The formulation of this is as follows:

$$\text{find } x_{ijk} \ (i = 1, \ldots, m; j = 1, \ldots, n; k = 1, \ldots, p)$$

$$\text{s.t.} \sum_{k=1}^{p} x_{ijk} = r_{ij} \ (i = 1, \ldots, m; j = 1, \ldots, n) \tag{2.1}$$

$$\sum_{j=1}^{n} x_{ijk} \leq 1 \ (i = 1, \ldots, m; k = 1, \ldots, p) \tag{2.2}$$

$$\sum_{i=1}^{n} x_{ijk} \leq 1 \ (j = 1, \ldots, n; k = 1, \ldots, p) \tag{2.3}$$

$$x_{ijk} \in \{0, 1\} \ (i = 1, \ldots, m; j = 1, \ldots, n; k = 1, \ldots, p) \tag{2.4}$$

$$\text{where } x_{ijk} = \begin{cases} 1, & \text{if class } c_i \text{ and teacher } t_j \text{ meet at period } k; \\ 0, & \text{otherwise} \end{cases}$$

Constraints 2.1 ensure that each teacher gives the right number of lectures to each class. Constraints 2.2 ensure that each teacher is involved in at most one

lecture for each period. Constraints 2.3 ensure that each class is involved in at most one lecture for each period.

2.5.2 Course Timetabling Problem

Course Timetabling Problem (CTP) involves scheduling a number of students taking given courses, lecturers and classrooms into a fixed set of timeslots within a week. The course timetable requires several slots and with different categories such as lectures, tutorials and practical sessions, which fits within a week and repeats for whole semester.

As in any other timetabling problem, CTP also involves hard and soft constraints. Abdullah (Abdullah, 2006) outlined some examples of these constraints for the CTP as follows:

Hard constraints

- A student and a teacher cannot be in two places at the same time.
- Only one course is allowed to be assigned to a timeslot in each classroom.
- The classroom capacity should be equal to or greater than the number of students attending the course at a particular timeslot.
- The classroom assigned to the course should satisfy the features required by the course.

Soft constraints

- Students should not have a single course on a day.

- Students should not have to attend more than two consecutive courses on a day.
- Students should not be scheduled to attend a course that is assigned to the last timeslot of the day.

Furthermore, CTP is divided into five sub-problems: teacher assignment, class-teacher timetabling, course scheduling, student scheduling, and classroom assignment (Carter and Laporte, 1995, cited in Alvarez-Valdés, 2001) . The teacher assignment focuses on allocating teachers to courses, without considering allocation of courses to timeslots. The course scheduling problem only focuses on allocation of courses to timeslots. It is often assumed that the teacher assignment of teachers to courses has been done before actual scheduling of courses to timeslots.

Several models for CTP have been developed. Given below is one of these models based on mathematical formulation as presented in (de Werra, 1985).

Given q courses $K_1, \ldots, K_q$, and for each i course K_i consists of k_i lectures. Students are divided into r groups $S_1, \ldots, S_r$, such that in each S_i all students take exactly same courses. This means that courses in S_i must be scheduled all at different times. The number of periods is p, and l_k is the maximum number of lectures that can be scheduled at period k (i.e. the number of rooms available at period k).The formulation is as follows;

$$\text{Maximize} \sum_{i=1}^{q}\sum_{k=1}^{p} C_{ik}y_{ik} \tag{2.5}$$

$$\text{s.t} \sum_{k=1}^{p} y_{ik} = k_i \ (i = 1, \ldots, q) \tag{2.6}$$

$$\sum_{i=1}^{q} y_{ik} \leq l_k \ (k = 1, \ldots, p) \tag{2.7}$$

$$\sum_{i \in S_l} y_{ik} \leq 1 \ (l = 1, \ldots, r; k = 1, \ldots, p) \tag{2.8}$$

$$y_{ik} \in \{0, 1\} \ (i = 1, \ldots, q; k = 1, \ldots, p) \tag{2.9}$$

where $y_{ik} = \begin{cases} 1, & \text{if lecture of course } K_i \text{ is scheduled at period } k; \\ 0, & \text{otherwise} \end{cases}$

Constraints (2.6) impose that each course is composed of the correct number of lectures. Constraints (2.7) enforce that at each time there aren't more lectures than rooms. Constraints (2.8) prevent conflicting lectures to be scheduled at the same period.

2.5.3 Examination Timetabling Problem

Whereas in course timetabling the period in which events are to be scheduled is normally fixed (usually a week) and this schedule is repeated cyclically, in examination timetabling a set of examinations must all be scheduled within a certain time period of usually a few weeks and the exact length of the timetable is in general not essential. The number of examinations to be scheduled varies greatly depending on the institution.

This problem involves the assignment of courses to be examined and their candidates to timeslots and examination rooms while satisfying a given set of constraints. As it has been stated earlier in chapter 1, that these constraints are classified into two categories; hard and soft. The hard constraints must be satisfied in order to produce a feasible timetable, whilst violation of the soft constraints should be minimised. For some problems, it is not possible to find a solution which satisfies all hard constraints so in such circumstances these must be relaxed to soft constraints with a high priority to minimise. The main hard constraints encountered in practical examination timetabling are:

- No student should be required to sit two examinations simultaneously.
- There must be enough seating capacity in the room for the number of students scheduled in it at any given time slot. (see, example in Mushi, 2004)

Some soft constraints for ETP are:

- Students should not be scheduled to sit more than one examination in a day.
- Students should not be scheduled to sit examinations in two consecutive timeslots.
- Each student's examinations should be spread as evenly as possible over the schedule.
- Examinations with large number of candidates should be scheduled earlier. (see example in Abdullah, 2006).

Schaerf (Schaerf ,1999) described a representation of ETP as follows. Given that

- q is the number of courses,
- r is the number of exams,
- k is a time slot
- p is the number of time slots,
- l_k is the maximum number of exams that can be scheduled at time slot k.
- i is a course, having a single exam,

- S_l is a group of exams such that in each group there are students that take all exams in the group.

Then

$$\text{find } y_{ik}\ (i = 1, \ldots, q; k = 1, \ldots, p)$$

such that

$$\sum_{k=1}^{p} y_{ik} = 1\ (i = 1, \ldots, q) \tag{2.10}$$

$$\sum_{i=1}^{q} y_{ik} \leq l_k\ (k = 1, \ldots, p) \tag{2.11}$$

$$\sum_{i \in S_l} y_{ik} \leq 1\ (l = 1, \ldots, 2^{r-1}; k = 1, \ldots, p) \tag{2.12}$$

$$y_{ik} \in \{0, 1\}\ (i = 1, \ldots, q; k = 1, \ldots, p) \tag{2.13}$$

where $y_{ik} = \begin{cases} 1, & \text{if exam of course } K_i \text{ is scheduled at timeslot } k; \\ 0, & \text{otherwise} \end{cases}$

Constraints (2.10) impose that each exam is scheduled only once. Constraints (2.11) enforce the maximum number of exams that can be scheduled for a given time slot. Constraints (2.12) prevent conflicting exams from being scheduled at the same time slot.

2.6 Complexity of Examination Timetabling Problem

In this section we discuss time complexity of the Examination Timetabling problem (ETP). In particular, we show that ETP is NP-hard.

According to the time complexity theory, in order to prove that a given problem $\mathcal{P}$ is NP-hard it suffices to show that there is a polynomial time reduction from

an NP-hard problem $\mathcal{Q}$ into $\mathcal{P}$ (see, eg., Garey and Jonhson, 1979). Since the graph k-colouring problem is NP-complete (Garey and Jonhson, 1979), we use the following results to prove that ETP is NP-hard.

Theorem 2.1 *The graph $k-$Colouring is polynomial time reducible to ETP.*

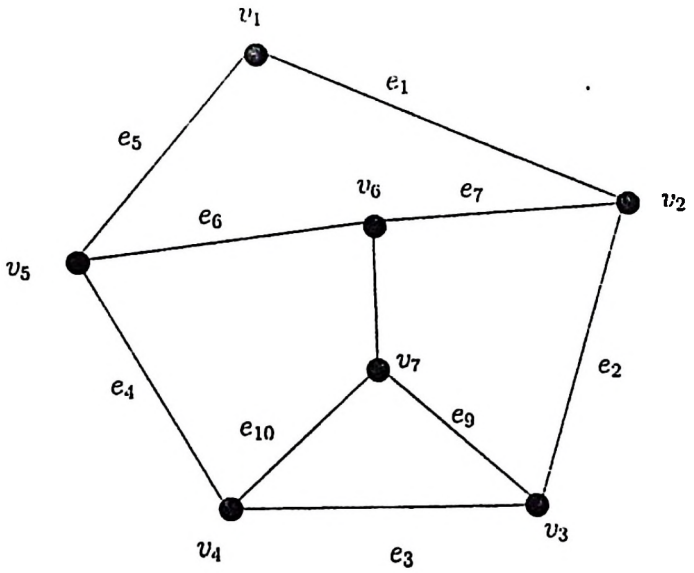
Proof: Recall that the NP-complete Graph k-Colouring problem asks that given a graph G and an integer k whether it is possible to assign a colour to each vertex of G in such a way that no two adjacent vertices have the same colour using at most k distinct colours. Our reduction from the graph k-coloring into ETP proceeds as follows. Let (G, k) be an instance of the graph k-colouring problem, with $V(G) = \{v_1, \ldots, v_n\}$ and $E(G) = \{e_1, \ldots, e_m\}$. We construct an instance $\mathcal{T}$ of ETP as follows.

- Let $C = \{c_1, \ldots, c_n\}$ be set of courses:
- Let $S = \{s_1, \ldots, s_m\}$ be set of students.
- For each edge $e_i = v_j v_k$ we add the student s_i to both courses c_j and c_k. $1 \leq i \leq m$, $1 \leq j, k \leq n$.

An example of a reduction of the examination tabling problem from the graph colouring problem is given in Figure 2.3.

Clearly, $\mathcal{T}$ can be constructed in polynomial time and $|C| = V(G)$. We now claim that G can be coloured with k colours if and only if $\mathcal{T}$ can be scheduled using k timeslots.

Suppose a $k-$colouring $f : V \to \{1, \ldots, k\}$ exists for G. We assign time-slots the courses as follows. Let t_i be time-slot for the course c_i. For each course c_i we set $t_i = f(v_i)$. The condition $f(v_i) \neq f(v_j)$ whenever $v_i, v_j \in E$ guarantees that two



$S(c_1) = \{s_1, s_5\}$

$S(c_2) = \{s_1, s_2, s_7\}$

$S(c_3) = \{s_2, s_3, s_9\}$

$S(c_4) = \{s_3, s_4, s_{10}\}$

$S(c_5) = \{s_4, s_5, s_6\}$

$S(c_6) = \{s_6, s_7, s_8\}$

$S(c_7) = \{s_8, s_9, s_{10}\}$

$\mathcal{T}$

Figure 2.3: Construction of an instance of the examination timetabling problem $\mathcal{T}$ from the graph G. $S(c_i)$ denotes the set of students in the course c_i.

courses with the same student receive different time-slots. Moreover, since f uses k colours, $\mathcal{T}$ is scheduled with k time-slots.

Conversely, from the construction of $\mathcal{T}$ from G, a successful time-slot allocation of $\mathcal{T}$ using k slots defines a proper k-colouring of G. This finishes proof of the theorem. $\diamond$

Theorem 2.1 implies that ETP is NP-hard. Therefore, it is unlikely to obtain a polynomial time algorithm to solve ETP.

It is due to the NP-hard nature of this problem that local search meta-heuristics such as tabu search, simulated annealing algorithms and many other have been applied to find good solutions. An analysis of the literature on these and other techniques applied to examination timetabling are given in section 2.7.

2.7 Methods of Solving ETP

Many methods have been developed to solve the examination timetabling problem (ETP). These methods can be put into two categories; *exact* and *heuristic (approximate)* methods. Exact methods include formulating and solving Integer Programming(IP) models for TTPs. Exacts methods have been applied successfully but in most cases to smaller instances of TTPs (Adewumi, et al, 2008).

For large TTPs, usually it is difficult to find and prove the existence of an optimal solution. Therefore, it is necessary to develop a heuristic approach in order to find a good solution within a reasonable amount of time (Adewumi, et al, 2008). Heuristic methods include Tabu Search, Simulated Annealing, Graph-Based heuristics among others. In the following subsections are some authors' works on this problem and their findings.

2.7.1 Integer Programming Model (IP)

An IP problem is a Linear Programming Problem in which some or all of the variables to be optimized must be positive integers. There are three common variations of IP models, namely; pure, mixed and 0/1 IP. In pure IP all variables are integers while in mixed IP some of variables are integers and in 0/1 IP the integer variables are binary which can take only values of 1 for YES and 0 for NO. Various IP for solving TTP have been developed. Examples of these models are in sections 2.5.1, 2.5.2 and 2.5.3.

IP has been tried in a few cases to solve timetabling problems but its use is certainly not widespread. This is because of the generally held belief that, "When the problem is expressed mathematically, the number of variables and constraints become unmanageably large for practical size problems".

Mehmet et al (Mehmet et al, 2008) developed a mathematical model for exam

timetabling to schedule the final exams of Fatih University Vocational School (FUVS). Computations are carried out using Xpress MP Optimization Software. The results are significantly better than the existing manual approach implemented at FUVS in terms of conflict elimination and classroom usage efficiency.

Al-Yakoob et al (Al-Yakoob et al, 2007) formulated a mixed-integer exam timetabling programming model (ETM) for Examination Timetabling Problem (ETP), which takes into account restrictions related to exam-period conflicts, facility and human resources, and commuting and traffic considerations. The results obtained significantly improve upon those derived via the existing manual approach implemented at Kuwait University, in terms of eliminating conflicts as well as from the overall efficiency and equity points of view.

Qu *et al* (Qu *et al* , 2009) formulated an IP model with an adaptive decomposition approach to solve TTP. In their work, they first presented an IP formulation for solving the difficult sub-problem. To have a tighter initial formulation, a well-known inequality called the Clique Inequality was utilized. They then examine the combinatorial properties of the problem to introduce a new family of cutting planes. These are shown to be helpful in obtaining a solution which is within a gap of less than 10% of optimal for the sub-problem, based on which the final solution is constructed. Promising results have been obtained on several benchmark exam timetabling problems in the literature

Mushi (Mushi, 2004) had formulated, tested and compare three Integer programming models to examination timetabling Problem at University of Dar Es Salaam. The findings showed that, although exact methods cannot give a solution to a real-size problem, these models give a good benchmark for testing the performance of other approaches. Furthermore it was observed that model 3, which is a mixed pure and 0/1 variables, performs better than the other two models.

2.7.2 Tabu Search (TS) Algorithm

Tabu Search (TS) is a metaheuristic that uses a local search procedure to iteratively move from a current solution s to a neighbour solution s' in the neighborhood of s, until some stopping criterion has been satisfied. The search process starts with an initial solution and moves from neighbour to neighbour as long as possible while improving the value of the objective function. Any solution which has been recently selected is put into a tabu list so that it becomes 'taboo' for a short period of time, depending on the length of the list. This minimizes the chance of cycling in the same solution, and therefore create more chances of improvement by moving into un-explored areas of the search space (Mushi. 2006). Hromkovic (Hromkovic, 1998) outlined a pseudo code for TS as shown in Figure 2.4;

Step 1: Initialize parameters;
Choose Initial Solution, S_0;
Set $TABU := S_0$;
Stop = FALSE;
$BEST := S_0$;
Step 2: Find a best feasible solution $S_i \in N(S_0) - TABU$:
Given an objective function F, find $\sigma = F(S_i) - F(S_0)$:
if $(\sigma < 0)$ **then** $BEST = S_i$;
Update TABU and Stop:
$S_0 = S_i$;
step 3: if Stop=TRUE **then output** (BEST) **else go to** step 2:

Figure 2.4: Pseudocode for TS algorithm

Some researchers have applied TS techniques to solve examination timetabling problems in recent years. Alvarez-Valdes *et al* (1997) used TS algorithms to

solve examination scheduling problem for the Faculty of Mathematics at the University of Valencia. First it finds a solution that no student has two exams simultaneously and then improves it by evenly spacing the examinations in the examination period. They concluded that the use of TS algorithms has allowed to obtain high quality solutions in very short computing times, in spite of the size of the problem and the complexity of data and objectives.

White and Xie (2001) implemented a TS algorithm which is called OTTABU. It used both recency-based short-term memory and frequency-based longer-term memory to improve the solution quality. The system was tested using real data obtained from the University of Ottawa registrar's office and real examination schedules were produced. It was found that the use of longer-term memory produced better schedules than those produced without such memory.

Di Gaspero and Schaerf (2001) presented an examination timetabling algorithm that is based on TS and graph colouring heuristics. In order to guide the search to explore different areas of the solution space, they modified the objective function by changing the weights. A dynamic size of the tabu list within a given interval was used to store the most recently accepted moves.

2.7.3 Simulated Annealing (SA) Algorithm

The term SA derives from the roughly analogous physical process of heating and then slowly cooling a substance to obtain a strong crystalline. SA is a probabilistic local search technique. The process starts by creating a random initial solution. The procedure searches by jumping from one candidate solution to another within a predefined neighbourhood. Each iteration of the SA algorithm replaces the current solution, s^* by a random "nearby" solution, s, chosen with a probability that depends on the difference between corresponding function values $f(s)$ and $f(s^*)$ and on a global parameter T, called *temperature* that gradually

decreases during the process according to some cooling schedule. If the move improves the quality of the solution then it is definitely accepted, otherwise, it is only accepted with a probability, $P = e^{-\Delta/T}$, where $\Delta = f(s) - f(s^*)$. Kirckpatrick *et al* (Kirkpatrick *etal*, 1983) proposed the use of SA for combinatorial problems. Hromkovic (Hromkovic, 1998) outlined the pseudo code for simulated annealing algorithm, and Mushi (Mushi, 2007) summarized the SA algorithm for examination scheduling problem as shown in Figure 2.5 below.

ETP SA Algorithm

Initialize parameters (Temperature T, freezing point F):

Get Initial Solution S_0;

While temperature $T >$ freezing point F

Get Solution, S in the neighbourhood of S_0;

Calculate $\sigma = S - S_0$:

If ($\sigma < 0$)

Accept new solution ($So = S$);

Else Generate a random value $r \in (0.1)$:

If ($r < e^{-\sigma/T}$)

Accept new solution ($So = S$);

Else

Reject new solution

Update temperature ($T = f(T)$);

End ETP SA;

Figure 2.5: Pseudocode for TS algorithm

SA has been extensively studied in the area of examination timetabling with some success. Johnson *et al* (1991) and Mushi (2007) applied SA in real world examination timetabling, and obtained a good solution compared to manual approaches.

El-mohamed *et al* (1998) applied an SA algorithm with different cooling schedules (geometric, adaptive and adaptive with reheating) to a course timetabling problem. The algorithms were tested on real data at Syracuse University. Experimental results show that the SA with adaptive cooling and reheating algorithm outperformed other methods.

Also, Thompson and Dowsland (1996) investigated SA techniques for examination timetabling. The problem is solved in a 2-phase approach. The first phase is concerned with finding a feasible solution and the second phase attempts to optimize the soft constraints to produce better quality timetables. They continued the work by implementing an adaptive cooling schedule rather than a geometric cooling schedule. The computational results show that an adaptive cooling schedule outperformed a simple geometric cooling approach.

2.7.4 Graph Colouring Heuristics

Graph based heuristics were among the earliest approaches to be used for the timetabling problem (Welsh and Powell, 1967). They are used as constructive heuristics, constructing solutions by ordering the exams that have not yet been scheduled, according to the perceived difficulty in scheduling that exam into a feasible timeslot. The difficulty of an exam can be represented in various ways such as the degree of conflict it has with other exams, the number of student enrollments etc. Some common graph based heuristics are:

1. Largest degree first: This heuristic first schedules the exam that has the largest number of conflicts with other exams (Abdullah, 2006; Malkawi *et al*, 2008).

2. Colour degree: Exams with a greater number of conflicts with the exams that have already been scheduled have a higher priority of being scheduled next. This is also known as Recursive Largest First (Abdullah, 2006).

3. Least Saturation degree: Exams with fewer feasible slots are scheduled as early as possible. The priority of exams (in the unscheduled list) to be scheduled are changed dynamically during the construction of solution (Abdullah, 2006).

4. Largest weighted degree: This is a modification of largest degree where priority is given to the examination that has the largest weighted conflict. Each conflict is weighted based on the number of students enrolled in two conflicting examinations (Abdullah, 2006; Malkawi *et al*, 2008).

5. Largest enrolment degree: This is a modification of largest degree where the examination with the largest student enrolment is scheduled first (Abdullah, 2006).

The GCP and its relationship to timetabling is widely discussed in the literature (Mehta, 1981; Abdullah, 2006; Burke *et al*, 1994; Malkawi *et al*, 2008; Redl, 2009). Burke et al. (Burke *et al*, 1994) presented graph colouring and room allocation algorithms for the university timetabling problem. A graph colouring algorithm is used to split the examinations into non-conflicting clusters and the room allocation algorithm is used to place the examinations into rooms.

Sabar *et al* (Sabar *et al*, 2009) presented a simple graph based heuristic that employs a roulette wheel selection mechanism for solving examination timetabling problems. They arrange exams in descending order of the number of conflicts

(degree) that they have with other exams. The difficulty of each exam to be scheduled is estimated based on the degree of exams in conflict. The degree determines the size of a segment in a roulette wheel, with a larger degree giving a larger segment. The roulette wheel selection mechanism selects an exam if the generated random number falls within the exam's segment. This overcomes the problem of repeatedly choosing and scheduling the same sequence of exams. They utilize the proposed Roulette Wheel Graph Colouring heuristic on the uncapacitated Carter's benchmark datasets. Results showed that this simple heuristic is capable of producing feasible solutions for all 13 instances.

Despite the fact that the literature on timetabling problem is always growing, the review of literature revealed that, there exist no study based on graph colouring approach for timetabling problem has been done in Tanzania.

CHAPTER THREE
METHODOLOGY

3.1 Graph Colouring Model

One of the most common models for examination timetabling problems is the graph colouring model which is used as the basic model for our problem. Modeling the examination timetabling problem in this way allows us to apply techniques and ideas successfully used for GCPs.

The basic structure of every examination timetabling problem contains a set of examinations, $E = \{e_1, ..., e_n\}$ together with constraints on which examinations clash and therefore cannot be scheduled together. Equating the set E of examinations to set of vertices, $V = \{v_1, \ldots, v_n\}$ of a graph we can add an edge, (v_i, v_j) for every pair (e_i, e_j) of examinations which clash.

The other important component of every examination timetabling problem is the time slots into which all the examinations must be scheduled. Using our graph representation, it is clear that any pair of examinations which share an edge cannot be assigned to the same time slot. Therefore, examinations must be assigned to timeslots in such a way that this constraint holds. This is analogous to the well known problem of graph-colouring (Welsh and Powell, 1967; Brelaz, 1979) in which the vertices of a graph must be assigned a colour such that no two vertices sharing a common edge have the same colour. de Werra (de Werra, 1985) gives a formulation of course timetabling as a graph colouring problem and also discusses the differences between course and examination timetabling with regard to this model. For examination timetabling, colours represent the different timeslots.

The related graph k-colouring problem limits the number of colours to k and

requires only that a solution to the problem is found using k or fewer colours. This is equivalent to an examination timetabling problem in which the number of timeslots is fixed. Otherwise, the number of colours (timeslots) is a variable which is to be minimised. Both cases represent common TTPs. This model only takes account of the boolean value of whether two examinations clash or not and treats all clashes equally. A very simple example of this graph colouring model is given in Figure 3.2

The problem in Figure 3.2 contains seven examinations, labelled e_1 to e_7 together with a set of thirteen edges showing the conflicting examinations. Clearly, real world examination timetabling problems are far more complex, sometimes containing up to 1000 examinations, but the graph-colouring model is still applicable (Malkawi, *et al*, 2008).

The model involves creating a conflict graph from the assembled input university exams data, properly colouring the conflict graph, and transforming this coloring into a conflict-free exams timetable. From this conflict-free exam timetable one can then assign the exams to classrooms based on room capacity and availability.

3.1.1 Recursive Largest First (RLF) Algorithm for GCP

RLF incrementally constructs an independent set $V_1 \subset V$ of vertices that are coloured with the first colour, then, considering only vertices in $V \setminus V_1$ constructs the next independent set V_2 as a second colour class and so on. The first vertex placed in V_i, $i = 1, 2, \ldots$, is a vertex of maximum degree in the subgraph induced by $V \setminus \cup_{j=1}^{i-1} V_j$. At each other step, the algorithm selects a vertex with the maximum number of adjacent vertices that are uncoloured but already inadmissible for the i^{th} colour. Formally, the algorithm can be stated as indicated in Figure 3.1 ;

Input:= $G = (V, E)$,the undirected graph

Output:= C, the list of coloured vertices, and k, number of colours used

1. $C \longleftarrow \emptyset$
2. $U \longleftarrow \emptyset$
3. $V^* \longleftarrow V$
4. $k \longleftarrow 0$
5. Choose a vertex x of maximum degree in the subgraph induced by V^*. Increment k by 1 and proceed to step 6.
6. Assign colour k to x. Move x from V^* to C and all $i \in V^*$ that are adjacent to x from V^* to U.
   If V^* remains non-empty, proceed to step 7. Otherwise check whether $C = V$. If so, then stop with G coloured with k colours. If not, then $V^* := U, U := \emptyset$ and return to step 5.
7. Choose a vertex $x \in V^*$ that has the maximum number of edges to vertices in U. Go to step 6.

Figure 3.1: Pseudocode for RLF algorithm

The complexity of RLF is $O(n^3)$. One factor $O(n)$ is due to the determination of a vertex x of maximal degree. Traversing the non-neighbors of x in search for a vertex y with a maximal number of common neighbors with x may cost another $O(n^2)$ elementary operations.

3.1.2 Creating Exam Conflict Graph

After gathering all necessary information from registrar's office, we construct an examination timetabling conflict graph G reflecting the given data. Suppose we

have a set of n examinations $\{e_1, e_2, \ldots, e_n\}$ to be scheduled for a particular semester. Each examination e_i will be represented by exactly one vertex v_i in G. Therefore G contains n vertices, and $V(G) = \{v_1, v_2 \ldots v_n\}$.

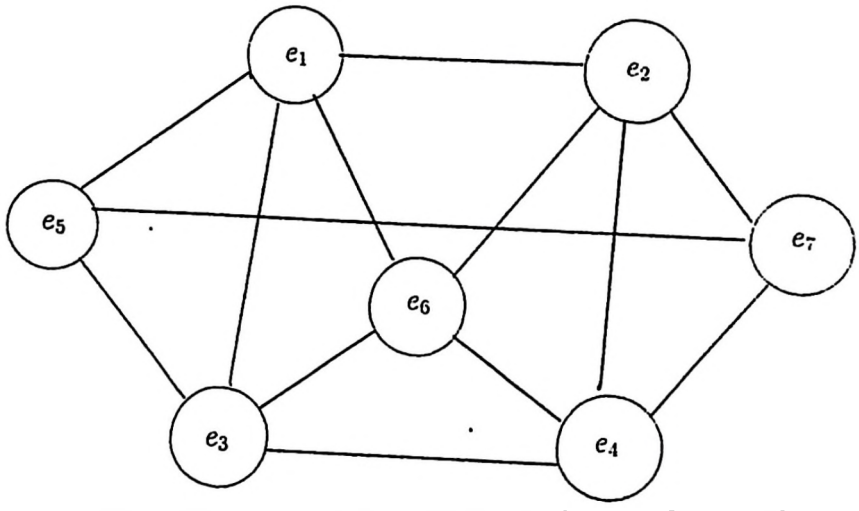


Figure 3.2: A simple 7-exams timetabling conflict graph

Figure 3.2 above illustrates an example of a simple timetabling instance in which we have seven exams to be scheduled : $e_1, e_2, \ldots, e_7$. We transform this graph into the matrix below (see figure 3.3), where a 1 indicates that a respective pair of exams, would cause a timetabling conflict if they are both scheduled at the same timeslot. The cause of these conflicts could be any of the hard constraints mentioned earlier. For example, exams e_1 and e_3 might have at least a common student or exams e_2 and e_7 might require the same classroom.

Then, we colour the resulted graph to obtain colour classes

3.1.3 Colouring the Conflict Graph by RLF Algorithm

After constructing the exam conflict graph, we then properly coloured its vertices, and ultimately we used that proper vertex colouring to construct a conflict-free examination timetable. Our goal is to properly colour the vertices of our conflict graph using RLF graph coloring algorithm.

	e_1	e_2	e_3	e_4	e_5	e_6	e_7
e_1	0	1	1	0	1	1	0
e_2	1	0	0	1	0	1	1
e_3	1	0	0	1	1	1	0
e_4	0	1	1	0	0	1	1
e_5	1	0	1	0	0	0	1
e_6	1	1	1	1	0	0	0
e_7	0	1	0	1	1	0	0

Figure 3.3: Adjacency Matrix shows conflict status between the exams

The algorithm works recursively, finding each colour set successively then removing all the coloured vertices from the graph and looking for a new set in the new reduced graph. For each colour, the remaining vertex with the largest degree is chosen and one by one vertices with a maximal number of common adjacent vertices are merged into it. When all vertices for the current component of the graph are coloured, the process is repeated in the other components again starting with a vertex with maximal degree. None of the merged vertices are adjacent to each other so they may be considered a colour set.

By considering the graph in figure 3.2, we illustrate how our exam conflict graph is coloured using RLF algorithm. We construct the first colour class V_1, we start by choosing the vertex of maximum degree. Here e_1, e_2, e_3, e_4 and e_6 both have degree 4. We consider the one with lowest vertex ID, i.e. e_1. Then, we look for non-neighbours of e_1 which have maximum number of common neighbours with e_1. Non-neighbours of e_1 are e_4 and e_7, but e_4 has three common neighbours, which are e_2, e_3 and e_6. While e_2 and e_5 are common neighbours of e_1 and e_7. Then, e_4 is contracted into e_1 and e_1 is removed from the graph. Therefore, $V_1 = \{e_1, e_4\}$, and so e_1 and e_4 are coloured with colour 1. The figure 3.4 below shows the subgraph induced by $V \setminus V_1$.

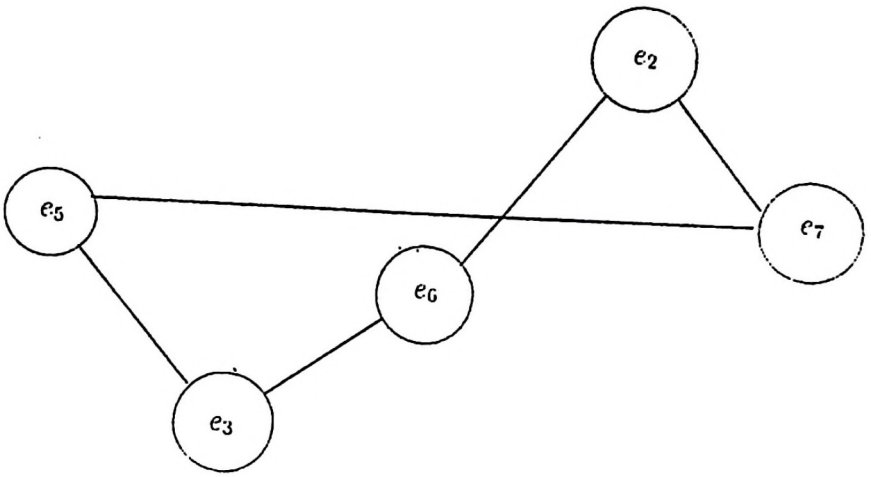


Figure 3.4: A subgraph induced by $V \setminus V_1$

We consider the induced subgraph $V \setminus V_1$ in order to construct the next colour class, V_2. e_3 is a vertex with maximum degree in $V \setminus V_1$. Its non-neighbours are e_2 and e_7, where e_2 has 1 common neighbour with e_3 (which is e_6) and e_7 has no common neighbour with e_3. So we contract e_2 into e_3, and remove e_3 from graph. Hence, e_2 and e_3 are coloured by colour 2. Below is subgraph induced by V_2.

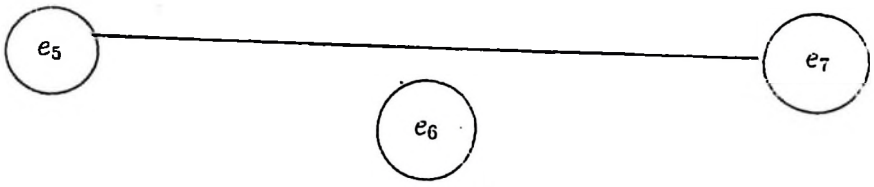


Figure 3.5: A subgraph induced by $V \setminus V_2$

In the induced subgraph in figure 3.5, e_5 and e_6 are coloured by next smallest legal available colour , i.e. colour 3. Vertex e_7 is coloured by colour 4. Therefore, $\chi(G)$ for this graph is 4. We use these colour classes to construct conflict free timetable. See Table 3.1 below

PERIODS	Period 1	Period 2	Period 3	Period 4
EXAMINATIONS	e_1, e_4	e_2, e_3	e_5, e_6	e_7

Table 3.1: A solution to examination conflict graph in Figure 3.2

3.2 Finding Rooms for Each Examination

We have already partitioned the exams conflict graph into independent subsets of exams which can scheduled into same timeslot without conflict. Next task is assigning room(s) for each exam at each timeslot.

Our aim is to assign each exam to a single room so as to reduce the number of rooms that are used during the exam period, by allowing for multiple exams to occupy a particular room at the same time, subject to room capacity constraints. of exam. However, when the exam is too large to be accomodated in a single room , it can be splited into some groups so as to fit into several rooms, with a maximum of five rooms per exam for each time slot. The algorithm is patterned after an algorithm by Burke, Elliman, and Weare (Burke, Elliman, and Weare ,1994) that focuses on accommodating a preference for larger and fewer rooms during the exam timetabling process.

CHAPTER FOUR
RESULTS AND ANALYSIS

4.1 Introduction

In this section, we present the results of this algorithm and compare it with that of manual scheduling. Our study case involved real scheduling data covering two semesters of 2010/11 at Sokoine University of Agriculture (SUA). The size of the two-semester data is illustrated in Table 4.1. This table shows the number of exams, number of available rooms, and the total number of student enrolment for all exams (a student usually enrolls in more than one exam). The algorithm was implemented using Microsoft Visual C++ 2010 Express. We ran the algorithm on a 2GHz machine with 1.87 GB RAM and Windows 7.

Table 4.1: Size of input data

Data	Number of Exams	Students' Enrolment	Number of Rooms
Semester I	337	43129	59
Semester II	375	43077	59

Source: Timetabling Section at SUA

4.2 Computational Results

4.2.1 Input data

This study was basically divided into two major parts. In the first part, an heuristic algorithm based on graph colouring is used to split the exams up into groups which may be scheduled together without conflict. While in the second part, an algorithm to fit the exams into rooms was implemented.

The data needed in this algorithm were stored in two text files namely: sem1 or

sem2 (.txt) and RoomSize (.txt). The sem1 (or sem2) file contains the Course Code, Degree Programme Registered for each course and Number of students. The RoomSize file contains Room Names and their corresponding capacities.

The user inputs the available courses for exams and classrooms with their corresponding enrollment size and capacities, respectively. To speed up and ease the data entry process, the user inputs the information in an excel sheet and then save them as text files.

4.2.2 Implementation of the Algorithm

In graph colouring algorithm, first we created conflict (adjacent) matrix for both data sets. Figure 4.1 below shows a sample of the resulting adjacent matrix for semester I.

Figure 4.1: A Sample of Exam Conflict Matrix for Semester I data

	$AE110$	$AE111$	$AE112$	$AE113$	$AEA101$	$AEA104$	$AEA111$
$AE110$	0	1	1	1	1	0	0
$AE111$	1	0	1	1	1	0	0
$AE112$	1	1	0	1	1	0	0
$AE113$	1	1	1	0	1	0	0
$AEA101$	1	1	1	1	0	1	1
$AEA104$	0	0	0	0	1	0	1
$AEA111$	0	0	0	0	1	1	0

A "1" in position ij indicates that there is a conflict between $exam_i$ and $exam_j$. This implies that at least one student has registered for $exam_i$ and $exam_j$. For example, from Figure 4.1 above we say that there is at least one student who has opted for both AE110 and AEA101. The matrix however does not tell who the affected students are.

After creating the matrix, we assign exams into timeslots using the graph colouring algorithm. Then, after the exams have been grouped according to specific colours for scheduling, the exam timeslots are now sorted according to colour assigned to it and its size. Timeslot with lowest colour number should be scheduled first. The sample of results are as presented in Table 4.2. For more details of results see Appendix A

Table 4.2: Sample of Results

Exam	Number of students	Room(s)	Day and Session
MTH106	2503		Day1-Morning
RD203	231	MPH	Day1-Morning
EDC202	214	SJ345	Day1-Morning
RD309	208	NLT	Day1-Morning
AS308	204	SI12	Day1-Morning
WLD207	92	MLT1	Day1-Morning
⋮	⋮	⋮	⋮
SC101	2293		Day2-Morning
HE310	331	DS	Day2-Morning
BTM104E	216	SJ345	Day2-Morning
EDM200	194	SH4	Day2-Morning
AEA210	125	MHPH	Day2-Morning
⋮	⋮	⋮	⋮

Source: Output of Algorithm

4.3 Analysis

4.3.1 Assigning Exam to Timeslot

The exam is assigned into exam timetable based on colour assigned. Based on this assignment, it shows that exam timetable at SUA for semester I and II, 2010/11 used 13 and 17 timeslots respectively compared to original exam timetable which has 30 timeslots. With this algorithm, the reseacher saved 17 and 13 timeslots respectively. This means that it save the resources such as rooms and exam periods of time.

However, compulsory courses for first year students have larger number of candidates, so it requires the timetable officers to split each of these into many rooms. Examples of these courses are MTH106 with 2503 candidates, and SC101 with 2293 candidates as indicated in 4.2. To assign the exam for each of these courses with others that have assigned the same colour could cause chaos in room allocation. Therefore, each of these exams will take their own timeslots and remaining exams will be allocated into other timeslots. That is, exam for MTH106 can be assigned first timeslot of Day 1, while the remaining exams for remaining course can be assigned second timeslot of the same day. Similary, the exam for SC101 can be assigned to first timeslot of Day 2, and the remaining exams for remaining courses on same colour number are assigned to second timeslot of the same day.

For second semester problem, 375 exams should be scheduled in 17 timeslots and for each timeslot an exam should be assigned into number of not more than five rooms. But however, only 373 exams have been assigned into rooms using this criterion. Therefore another two exams cannot be scheduled in the current timeslots. Due to this problem, two special timeslots has been created to accommodate each of these two exams. Therefore, our timetable now has 19 timeslots instead of 17. After exams have been assigned in special timeslot, then rooms

will be assigned to the exams. However in first semester problem, all 335 has been scheduled in 13 timeslots with no extra timeslot created.

4.3.2 Assigning Each Exam to Room(s)

As previously stated in Section 3.2, after assigning each exam into timeslot, next task was to assign each exam to room(s). We have a set of exams to be fitted into a set of rooms. The objective is to fit as many students as possible into each room to maximize the use of rooms. For this problem, a sorted list of exams for a specific timeslot to be fitted in the rooms based on:-

Largest First:- The largest space available will be fit to optimize the usage of largest space and minimize the number of rooms and invigilators.

Best First:- Exam will be fit in the smallest amount of remaining room capacity.

The room assignment works by filling up the largest rooms first, then continuing on with smallest amount of remaining capacity until all exams are assigned to the rooms. One of the major problems in room assignment is when number of students registered for a certain exam can not fit in one room or when more that one exams are scheduled in the same room. Table 4.3 shows the a sample of room allocation for exams that have been coloured with colour 1.

In this problem, we fitted each exam into a single room that is large enough to accommodate it. However, when the size of exam can not fit in a single room, we split it into maximum of four rooms per exam. The priority of choice for the room is given to larger rooms starting with largest available room.

Table 4.3: Sample of Room Allocation

Room	Capacity	Exam and its Candidates	Unused Seats	% of Unused Seats
MPH	250	RD203 (231)	19	7.60
SJ345	216	EDC202 (214)	2	0.93
NLT	220	RD309 (208), HT306 (11)	1	0.45
SI12	204	AS308 (204)	0	0.00
SH4	200	AS206 (180)	20	10.00
SLT1	224	FEC301 (180), INF311(44)	0	0.00
SK123	160	AEA311 (160)	0	0.00
MLT1	93	WLD207 (92)	1	1.08
MC7	90	AE202 (84)	6	6.67
MC8	90	EDP303 (84)	6	6.67
MC1	80	BLS207 (80)	0	0.00
MC2	80	HE103 (78)	2	2.50
MLR5	90	AE215 (73)	17	18.89
MLT2	93	ENV213 (73)	20	21.51
SL3	96	FMM205 (72)	24	25.00
MLT6	65	ENV309 (62)	3	4.62
MLT7	65	BLS308 (61)	4	6.15
MLR3	60	EE207 (60)	0	0.00
MLT5	57	HE315 (55)	2	3.51
MLR4	60	FT205 (52)	8	13.33
MLT3	47	AQ307 (45)	2	4.26
SR	30	AQ204 (27)	3	10.00
RM025	35	BTM215S (26)	9	25.71
MC6	15	AE325 (14)	1	6.67

Source: Output of Algorithm

CHAPTER FIVE
CONCLUSION AND FUTURE WORKS

5.1 Conclusion

In this study, we developed a graph colouring based algorithm for solving ETP at SUA. We used RLF graph colouring heuristic to decompose a set of exams into independent sets, each of them consists of exams that can be scheduled at one timeslot without conflict between them.

The graph colouring heuristic was used to group exams based on colours and each colour represents timeslot(s). Each exam is represented as vertex and conflicts are represented by the edges between vertices. Vertices that share some edges can not be scheduled in the same timeslot.

The obtained results show that the exam period at SUA can be reduced from three to two weeks. Moreover, the results show that double sessions (i.e., morning and afternoon session) appeared in only two days, out of the ten days of two weeks of the exam period. This implies that most students will have to do one exam a day.

The current examination timetable setting process takes several weeks to prepare it, and still we are sure that the resulting timetable is the best possible. The developed algorithm will shorten this period by automatically generating a timetable which can then be imported into the timetable software for other changes in case there is an additional soft constraints.

5.2 Future Works

We have designed and implement an algorithm to tackle the examinations timetabling problem at SUA. A practical problem was used as a test case and solved with very high performance. The algorithm performs significantly better than manual system.

Based on the experiment, graph colouring heuristic is suitable for the problem that focus on hard constraints but it is difficult to solve for soft constraints. Therefore, in order to obtain optimal solution, metaheuristics approach such as Genetic Algorithm, Tabu Search and Simulated Annealing, should be applied together with graph colouring.

Another direction of future work is to include some constraints that

- limit number of departmental exams per timeslot subject to number of available resources such invigilators and rooms. This is important in order to avoid overloading the departments' workforce.
- promote uniform distribution of exams over the colors (timeslots). This is important so as to allow at least one gap between examinations and maximize the students' preparation time between exams.
- balance exams load for each timeslot over the available classrooms.

REFERENCES

Abdullah, S (2006). *Heuristic approaches for university timetabling problems.* Ph.D. Thesis of the University of Nottingham. pp 1- 226.

Adewumi, A.O., Sawyerr, B. A., Ali, M. M. (2009). "A heuristic solution to the university timetabling problem". *Engineering Computations: International Journal for Computer-Aided Engineering and Software.* Vol 26 (8). pp. 972-984

Al-Omari, H. and Sabri, K. E. (2006). "New Graph Coloring Algorithms". *American Journal of Mathematics and Statistics.* Volume 2 (4). pp 739-741.

Alvarez-Valdes, R., Crespo, E., Tamarit, J. M. (1997). "A Tabu Search Algorithm To Schedule University Examinations". *QÜ ESTIIO.*Volume 21 (1). pp 201-215.

Al-Yakoob, S., Sherali, H., Al-Jazzaf, M. (2007). "A mixed-integer mathematical modeling approach to exam timetabling". *Journal of Computational Management Science,* Volume 7 (1). 19-46.

Burke, E. K., Elliman, D.G., Weare (1994). "A University Timetabling System Based On Graph Coloring and Constraint Manipulation". *Journal of Research On Computing In Education.* Volume 27 (1). 1-18.

Butenko, S., Festa, P.,Pardalos, M (2001). "On the Chromatic Number of Graphs". *Journal of Optimization Theory and Applications,* Volume 109 (1). pp 51-67.

Burke, E., Bykov, Y., Newall., Petrovic, S. (2004). *A time-Predefined Local Search Approach to Exam Timetabling Problems.* pp 1-46.

Brelaz, D. (1979). "New Methods to Color the Vertices of a Graph". *ACM*. Volume 22 (4): pp 251-256.

Cooper, T. B and Kingston, J. H. (1995). "The Complexity of Timetable Construction Problems". *Technical Report Number* 495. pp 1-13.

Cowling P., Kendall G., Hussin N. (2002). "A Survey And Case Study Of Practical Examination Timetabling Problems." *Proceedings of the 4th International Conference on Practice and Theory of Automated Timetabling* (PATAT 2002), Gent, pp. 258-261.

Di Gaspero, L. and Schaerf, A. (2001). "Tabu Search Techniques for Examination Timetabling". *PATAT III: Lecture Notes in Computer Science*. Volume 2079. pp 104-117.

De Werra, D. (1985). "An Introduction to timetabling". *European Journal of Operational Research*, Volume 19. pp 151-162.

El-mohamed, M. A. S and Coddington P. (1998). *A Comparison Of Annealing Techniques For Academic Course Scheduling.* Lecture Notes in Computer Science, Volume 1408. pp 92-114.

Hromkovič, J (1998), *Algorithmics for Hard Problems: Introduction to Combinatorial Optimization, Randomization, Approximation, and Heuristics*. 2nd Edition. Berlin, Springer.

Garey, M and Johnson, D (1979), *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman and Company.

Großmann, A (2005). "Introduction to Complexity Theory". *Lecture Notes: Knowledge Representation and Reasoning Group Artificial Intelligence Institute.*

Johnson, D., Aragon, C., McGeoch, L., Schevon, C. (1991). "Optimization by Simulated Annealing: An Experimental Evaluation; Part II, Graph Coloring And Number Partitioning". *Journal of Research*, Volume 39 (3). pp 378-406.

Kirkpatrick, S., Gelatt, C. D., Vecchi, M. P. (1983). "Optimization by Simulated Annealing". *Journal of Science, New Series*, Volume 220 (4598). pp 671-680.

Lucet, C., Mendes, F., Moukrim, A. (2005). *An Exact Method for Graph Coloring*. Computers and Operations Research. pp 1-19.

Malkawi, M., Hassan, Al-Haj., Hassan, O. A. (2008). "A New Exam Scheduling Algorithm Using Graph Coloring". *The International Arab Journal of Information Technology*, Volume 5 (1). pp 80-86.

Mehmet ,S., Uysal, O., Sari, M. (2008). *A Mixed-Integer Mathematical Model for Exam Timetabling: A Case Study at Fatih University Vocational School*. www.asap.cs.nott.ac.uk/patat/patat08/Papers/Mehmet-WC2c.pdf, retrieved on Friday, 16th September, 2011.

Mehta, N. K (1981). "The Application of a Graph Coloring Method to an Examination Scheduling Problem". *Interfaces*, Volume 11 (5). pp 57-65.

Mkandawile, M. J (2004). *Development of Algorithms for Timetabling Problem: Case Study of University of Dar es salaam*. Unpublished Masters Dissertation of University of Dar Es Salaam.

Mushi, A. R (2004). "Mathematical Programming Formulations for the Examinations Timetable Problem: The Case of University of Dar es salaam". *African Journal Of Science and Technology (Science and Enginnering Series)*, Volume 5 (2). pp 34-40.

Mushi, A. R (2006). "Tabu Search Heuristic for University Course Timetabling Problem". *African Journal Of Science and Technology (Science and Enginnering Series)*, Volume 7(1). pp 34-40.

Mushi A. R (2007). "Simulated Annealing Algorithm for the Examinations Timetabling Problem". *African Journal Of Science and Technology (Science and Enginnering Series)*. Volume 8 (2). pp 24-32.

Qu, R (2009). "Hybridizing Integer Programming Models with an Adaptive Decomposition Approach for Exam Timetabling Problems". *The 4th Multidisciplinary International Scheduling: Theory and Applications 2009 (MISTA 2009)*, pp. 435-446, Dublin, Ireland

Redl, T. A. (2009). "On using Graph Coloring to Create University Timetables With Essential and Prefferential Conditions". *Advances in Marketing, Management and Finances, Proceedings of the 3rd International Conference on Management, Marketing and Finances (International Conference on Computational and Information Sciences, University of Houston-Downtown)*. pp. 162-167.

Rothe, J (2000). "Heuristics Versus Completeness for Graph Coloring". *Chicago Journal of Theoretical Computer Science: The MIT Press*, Volume 2000 (1). pp 1-16, 2000.

Sabar. N.R (2009). "Roulette wheel Graph Colouring for Solving Examination Timetabling Problems". *Combinatorial Optimization and Applications Lecture Notes in Computer Science*, Volume 5573. pp. 463-470

Salari, E. and Eshghi. K (2008). "An Aco Algorithm for the Graph Coloring Problem". *International Journal in Contemporary Mathematical Sciences*, Volume 3 (6). pp 293-304.

Sloper,C (2001). *Parameterized Complexity and the Method of Test Sets.* Department of Informatics, University of Bergen, Norway.

Thompson, J. M and Dowsland, K. A (1996). "Variants of Simulated Annealing for the Examination Timetabling Problem". *Annals of Operations Research*, Volume 63 (1996). pp 105-128.

Welsh,D.J.A and Powell, M.B (1967). "An Upper Bound for the Chromatic Number of a Graph and Its Applications to Timetabling Problems". *Comp. Journal.* Vol 10. 85-86.

Whalen. S (2002). "Graph Coloring with Genetic Algorithms". *International Journal in Contemporary Mathematical Sciences.* pp 1-9

White, G. M. and. Xie, B. S (2001). "Examination Timetables and Tabu Search With Longer-term Memory". *Lecture Notes in Computer Science* vol. 2079. The Practice and Theory of Automated Timetabling III: Selected Papers (PATAT 2000). pp 85-103.

APPENDIX A

Table 6.1: Examination Timetable from Semester II dataset

Exam	Number of Students	Room(s)	Day and Session
MTH106	2503		Day1 Morning
RD203	231	MPH	Day1 Morning
EDC202	214	SJ345	Day1 Morning
INF311	44	SLT1	Day1 Morning
SC101	2293		Day2 Morning
HE310	331	DS	Day2 Morning
BTM104E	216	SJ345	Day2 Morning
CIT100	1347	SLT3, SLT4, SLT6, MPH, DS	Day3 Morning
EE205	272	NLT SLT1,	Day3 Morning
EE104	251	SI12, SJ345,	Day3 Morning
AEA102	921	SLT1, MPH, DS	Day4 Morning
RD110	247	SLT4, SLT6,	Day4 Morning
BTM104F	216	SJ345	Day4 Morning
EE105	933	SLT6, MPH, DS	Day5 Morning
FBL102	452	SLT1, SLT4, SLT3,	Day5 Morning
AEA204	319	NLT SLT3,	Day5 Morning
EE102	469	DS	Day6 Morning
BTM108	307	SLT3, MPH,	Day6 Morning
AEA103	279	SLT1, SLT4,	Day6 Morning

Exam	Number of Students	Room(s)	Day and Session
EE103	444	DS	Day7 Morning
ENV102	363	SLT4, MPH,	Day7 Morning
CS204	287	SLT1, SLT6,	Day7 Morning
AS104	572	MPH, DS	Day8 Morning
AEA105	279	SLT3, SLT6,	Day8 Morning
CS205	260	SLT4, SLT1,	Day8 Morning
ENV103	398	DS	Day9 Morning
AEA106	279	SLT1, MPH,	Day9 Morning
RD103	274	SLT4, SLT6,	Day9 Morning
RD106	274	DS	Day10 Morning
AEA110	271	DS, MPH,	Day10 Morning
SS100	216	SJ345	Day10 Morning
CS104	232	MPH	Day11 Morning
BTM214	209	SJ345	Day11 Morning
CS203	177	SH4	Day11 Morning
CS105	232	MPH	Day12 Morning
ENV220	73	MC1	Day12 Morning
INF108	40	CR	Day12 Morning
ZOO106	142	MHPH	Day13 Morning
INF111	71	MC1	Day13 Morning
MTH111	23	CR	Day14 Morning
INF204	42	RM026	Day15 Morning
INF212	42	RM026	Day1 Afternoon
INF213	42	RM026	Day2 Afternoon

Source: Output of Algorithm

APPENDIX B

```
// Timetabling.cpp : Defines the entry point for the console application.


#include "stdafx.h"

//int Graph[500][500];
int Total_Cs_Dg;
int Total_Courses;
Degree_Course Data[1000];
const int MAX = 1000;
int NumOfExams;
int Graph[MAX][MAX];
int timeslot[MAX];
int degree[MAX];
int NN[MAX];
int NNCount;
int unscheduled;
int ColorNumber=0;
int const MaxRoomAllocate=4;
int const NoSlots=30;
RoomSize RoomInfo[100];
Allocation CourseRoom[500];
int NoColorUsed=17;
int NoRooms;
int NoCourses;


class Set
```

```
{
public:
char Element[15];
Set * Next;
Set(char Crs[15])
{
strcpy_s(Element.Crs);
Next = NULL;
}
};


int emptyset(Set * C)
{
return (C == NULL);
}


void PutElement(Set  &P, char Crs[15])
{
if (P == NULL)
{
P = new Set (Crs);
}
else
{
PutElement(P->Next, Crs);
}
}
```

```
int Inset (Set * C, char Crs[15])
{
if (C == NULL)
{
return 0;
}
else
{
Set * Temp;
for(Temp = C;((Temp!=NULL)&& strcmp(Temp->Element, Crs)!= 0);Temp =Temp-
>Next);
if(Temp ==NULL)
return 0;
else
return 1;
}
}


int SetSize (Set * C)
{
int count=0;
Set * Temp;
for(Temp = C;Temp!=NULL;Temp =Temp->Next)
count++;
return count;
}
```

```
void SetIntersection(Set *& C, Set *C1, Set *C2)
{
Set * T;
C=NULL;
if (!emptyset(C2))
{
for (T = C2; T!=NULL; T =T->Next)
{
if(Inset(C1,T->Element))
PutElement(C,T->Element);
}
}
}


void ReadGraph()
{
char ReadDegree[15];
char ReadCourse[15];
int ReadNoStudents;
int i=0;
ifstream DataFile;
DataFile.open("c:/siza/data2.txt");
if (DataFile.fail())
{
cout<<"Sorry, the file couldn't be opened! \ n";
exit(1);
```

```
}
DataFile >> ReadDegree;
while (!DataFile.eof())
{
DataFile>>ReadCourse;
DataFile>>ReadNoStudents;
strcpy_s(Data[i].Degree,ReadDegree);
strcpy_s(Data[i].Course,ReadCourse);
Data[i].NoStudents=ReadNoStudents;
i++;
DataFile >> ReadDegree;
} DataFile.close():
Total_Cs_Dg=i:
}


class ListofCourses
{
public:
int ID;
char Course[15];
ListofCourses * Next;
ListofCourses(int n. char Crs[15])
{
ID=n;
strcpy_s(Course,Crs);
Next = NULL;
}
};
```

```
void AddCourse(ListofCourses * &P, int n, char Crs[15])
{
if (P == NULL)
{
P = new ListofCourses (n, Crs);
}
else
{
AddCourse(P->Next,n, Crs);
}
}


int InList (ListofCourses * C, char Crs[15])
{
if (C == NULL)
{
return 0;
}
else
{
ListofCourses * Temp;
for(Temp = C;((Temp!=NULL)&& strcmp(Temp->Course, Crs)!= 0);Temp =Temp-
>Next);
if(Temp ==NULL)
return 0;
else
```

```
return 1;
}
}

void DisplayCourses(ListofCourses *C)
{
ofstream ResultsFile:
ListofCourses * T;
ResultsFile.open("c:/siza/Results.txt");
for(T= C;T!=NULL;T=T->Next)
{
ResultsFile<<(T->ID)<<" \t";
ResultsFile<<(T->Course)<<" \n";
}
ResultsFile.close();
}

void CreateCourseList(ListofCourses *& C)
{
int i;
int index=0;
for(i=0;i<Total_Cs_Dg;i++)
{
if (!InList(C,Data[i].Course))
{
index++;
AddCourse(C,index,Data[i].Course);
```

```
}
}
Total_Courses=index;
cin>>index:
}

int Confict(char Course1[15], char Course2[15])
{
Set *S1=NULL;
Set *S2=NULL;
Set *S=NULL;
int i;
for(i=0;i<Total_Cs_Dg; i++)
{
if (strcmp(Data[i].Course, Course1)== 0)
{
PutElement(S1,Data[i].Degree);
}
}
for(i=0;i<Total_Cs_Dg; i++)
{
if (strcmp(Data[i].Course, Course2)== 0)
{
PutElement(S2,Data[i].Degree);
}
}
```

```
SetIntersection(S,S1.S2);
if(emptyset(S))
return 0;
else
return 1;
}


void CreateAdjMatrix(ListofCourses * C)
{
int i,j;
ofstream ResultsFile;
ListofCourses * Ptr1, *Ptr2;
for(i=0;i<Total_Courses;i++)
for(j=0;j<Total_Courses;j++)
{
Graph[i][j]=0;
}


ResultsFile.open("c:/Conflict.txt");
int kk=0;
for(Ptr1= C;Ptr1!=NULL;Ptr1=Ptr1->Next)
{ kk++;
ResultsFile<<kk<<". "<<Ptr1->Course<<"->";
for(Ptr2= C;Ptr2!=NULL;Ptr2=Ptr2->Next)
{
if(Ptr1!=Ptr2)
if (Confict(Ptr1->Course,Ptr2->Course))
```

```
{
ResultsFile¡¡Ptr2->Course<<", ";
Graph[(Ptr1->ID)-1][(Ptr2->ID)-1]=1;
}
}
ResultsFile<<endl;
}


ResultsFile.close();
ResultsFile.open("c:/Results.txt");
ResultsFile<<Total_Courses<< "\n";
// ResultsFile<<Total_Courses<<endl;
for(i=0;i<Total_Courses;i++)
{
for(j=0;j<Total_Courses;j++)
{ // cout<<i<<" vs " <<j<<endl;
ResultsFile<<Graph[i][j]<<" ";
}
ResultsFile<<endl;
}
ResultsFile.close();
}//End of Creating Conflict Matrix


void GetInput()// Graph Colouring Starts Here
{
ifstream inputfile;
inputfile.open("c:/Results.txt");
```

```
inputfile >>NumOfExams : // read the number of exams first
for (int i=0; i < NumOfExams; i++) // then read the adjacency matrix
for (int j=0; j < NumOfExams; j++)
inputfile >> Graph[i][j]; // read the element one by one
inputfile.close();
}


// initializing function
void Init()
{
for (int i=0; i < NumOfExams; i++)
{
int sum=0;
timeslot[i] = 0: // be sure that at first, no exam is scheduled
degree[i] = 0; // count the number of conflicting exams for each exam
for (int j = 0; j < NumOfExams; j++)
if (Graph[i][j] == 1) // if i-th exam is in conflict with another
degree[i] ++; // increase its degree by 1
sum=sum+degree[i];
}
NNCount = 0; // number of element in NN set
unscheduled = NumOfExams:
}
// this function finds the unprocessed vertex of which degree is maximum
int MaxDegreeVertex()
{
int max = -1;
int max_i;
```

```
for (int i =0; i < NumOfExams; i++)
if (timeslot[i] == 0)
if (degree[i]>max)
{
max = degree[i];
max_i = i;
}
return max_i;
}
// this function is for UpdateNN function it reset the value of scanned array
void scannedInit(int scanned[MAX])
{
for (int i=0; i < NumOfExams; i++)
scanned[i] = 0;
}
// this function updates NN array
void UpdateNN(int ColorNumber)
{
NNCount = 0;
// firstly, add all the uncolored vertices into NN set
for (int i=0; i < NumOfExams; i++)
if (timeslot[i] == 0)
{
NN[NNCount] = i;
NNCount ++; // when we add a vertex, increase the NNCount
}
// then, remove all the vertices in NN that
// is adjacent to the vertices colored with ColorNumber
```

```
for (int j=0; j< NumOfExams; j++)
if (timeslot[j] == ColorNumber) // find one vertex colored ColorNumber
for (int k=0; k < NNCount; k++)
while (Graph[j][NN[k]] == 1)
// remove vertex that adjacent to the found vertex
{
NN[k] = NN[NNCount - 1];
NNCount --; // decrease the NNCount
}
}

// this function will find suitable y from NN
int FindSuitableY(int ColorNumber, int& VerticesInCommon)
{
int temp,tmp_y,y;
// array scanned stores uncolored vertices except the vertex we are processing
int scanned[MAX];
VerticesInCommon = 0;
for (int i=0; i < NNCount; i++) // check the i-th vertex in NN
{
// tmp_y is the vertex we are processing
tmp_y = NN[i];
// temp is the neighbors in common of tmp_y and the vertices coloured Color-
Number
temp = 0;
scannedInit(scanned);
//reset scanned array in order to check all
//the vertices if they are adjacent to i-th vertex in NN
```

```
for (int x=0; x < NumOfExams; x++)
if (timeslot[x] == ColorNumber) // find one vertex colored ColorNumber
for (int k=0; k < NumOfExams; k++)
if (timeslot[k] == 0 && scanned[k] == 0)
if (Graph[x][k] == 1 && Graph[tmp_y][k] == 1)
// if k is a neighbor in common of x and tmp_y
{
temp ++;
scanned[k] = 1; // k is scanned
}
if (temp > VerticesInCommon)
{
VerticesInCommon = temp;
y = tmp_y;
}
}
return y;
}
// find the exam in NN of which degree is maximum
int MaxDegreeinNN()
{
int tmp_y; // the exam has the current maximum degree
int temp, max = 0;
for (int i=0; i < NNCount; i++)
{
temp = 0;
for (int j=0; j < NumOfExams; j++)
if (timeslot[j] == 0 && Graph[NN[i]][j] == 1)
```

```
temp ++;
if (temp>max) //if the degree of vertex NN[i] is higher than tmp_y's one
{
max = temp; // assignment NN[i] to tmp_y
tmp_y = NN[i];
}
}
if (max == 0) // so all the exams have degree 0
return NN[0];
else // exist a maximum, return it
return tmp_y;
}


// processing function
void Scheduling()
{
int x,y;
//int ColorNumber = 0;
int VerticesInCommon = 0;
while (unscheduled>0) // while there is an unscheduled exam
{
x = MaxDegreeVertex(); // find the one with maximum degree
ColorNumber ++;
timeslot[x] = ColorNumber; // give it a new color
unscheduled -
UpdateNN(ColorNumber); // find the set of non-neighbors of x
while (NNCount>0)
{
```

```
// find y, the vertex has the maximum neighbors in common with x
// VerticesInCommon is this maximum number
y = FindSuitableY(ColorNumber, VerticesInCommon);
// in case VerticesInCommon = 0
// y is determined that the vertex with max degree in NN
if (VerticesInCommon == 0)
y = MaxDegreeInNN();
// color y the same to x
timeslot[y] = ColorNumber;
unscheduled --;
UpdateNN(ColorNumber);
// find the new set of non-neighbors of x
}
}
}


void PrintOutput() // print out the output
{
ofstream periods;
periods.open("c:/siza/slots.txt");
for (int i=0; i < NumOfExams; i++) //element i-th of array timeslot stores the
timeslot of (i+1)-th exam
periods<<(i+1) << " "<< timeslot[i]<<endl;
periods.close();// END OF GRAPH COLOURING
}

void ReadRoomInfo()//Room Assignment Starts Here
{
```

```
int i=0;
ifstream DataFile;
DataFile.open("c:/siza/RoomSize.txt");
DataFile>> RoomInfo[i].Room;
while (!DataFile.eof())
{
DataFile>> RoomInfo[i].Capacity;
RoomInfo[i].FreeSpace=RoomInfo[i].Capacity;
i++;
DataFile >> RoomInfo[i].Room;
}
DataFile.close();
NoRooms=i;
}

void ReadCourseInfo()
{
int i=0;
ifstream DataFile;
DataFile.open ("c:/CourseSize.txt");
DataFile >> CourseRoom[i].Course;
while (!DataFile.eof())
{
DataFile >> CourseRoom[i].Size;
CourseRoom[i].Color=0;
CourseRoom[i].Slot=0;
i++;
DataFile >> CourseRoom[i].Course;
```

```
}
DataFile.close();
NoCourses=i;
}

void InitCourseAllocation()
{
int i=0;
ifstream DataFile;
int index;
DataFile.open("c:/slots.txt");
DataFile >> index;
while (!DataFile.eof())
{
DataFile >> CourseRoom[i].Color;
i++;
DataFile >> index;
}
DataFile.close();
}

void SortCoursesColors()
{
int i,j;
Allocation Temp;
for(i=0;i<NoCourses;i++)
for(j=0;j<(NoCourses-1);j++)
```

```
if(CourseRoom[j].Color>CourseRoom[j+1].Color)
{
Temp=CourseRoom[j];
CourseRoom[j]=CourseRoom[j+1];
CourseRoom[j+1]=Temp;
}
for(i=0;i<NoCourses;i++)
for(j=0;j<(NoCourses-1);j++)
if((CourseRoom[j].Color==CourseRoom[j+1].Color) &&(CourseRoom[j].Size<CourseRoom[j+
{
Temp=CourseRoom[j];
CourseRoom[j]=CourseRoom[j+1];
CourseRoom[j+1]=Temp;
}
}


void DisplayRoomAllocation()
{
int i;
ofstream Rooms;
Rooms.open("c:/RoomAllocation.txt");
for(i=0;i<NoCourses;i++)
{
Rooms<<CourseRoom[i].Course<<" \t"<<CourseRoom[i].Size<<
" \t"<<CourseRoom[i].Color<<" \t":
Rooms<<CourseRoom[i].Room<<"\t <<SlotName(CourseRoom[i].Slot)<<" \n":
}
Rooms.close();
```

```
}

void EmptyRooms()
{
int i;
for (i=0;i<NoRooms;i++)
RoomInfo[i].FreeSpace=RoomInfo[i].Capacity;
}

void SortRooms()
{
int i,j;
RoomSize Temp;
for(i=0;i<NoRooms;i++)
for(j=0;j<NoRooms-1;j++)
{
if(RoomInfo[j].FreeSpace > RoomInfo[j+1].FreeSpace)
{
Temp = RoomInfo[j];
RoomInfo[j]=RoomInfo[j+1];
RoomInfo[j+1]=Temp;
}
}
}

int FreeMaxRoom()
{
```

```
SortRooms();
return RoomInfo[NoRooms-1].FreeSpace;
}


void AssignSlot(int index, int AllocatedSlot)
{
int i,j;
int sum;
int Required=CourseRoom[index].Size;
if (CourseRoom[index].Size > FreeMaxRoom())
{
i=NoRooms;
sum=0;
do
{
i--;
sum=sum+RoomInfo[i].FreeSpace;
} while ((CourseRoom[index].Size>sum)&&(i>=NoRooms-(MaxRoomAllocate)));
j=i;
if (CourseRoom[index].Size <=sum)
{
for( ;i<=NoRooms-1;i++)
{
if(Required > RoomInfo[i].FreeSpace)
{
Required=Required-RoomInfo[i].FreeSpace;
RoomInfo[i].FreeSpace=0;
}
```

```
else
{
RoomInfo[i].FreeSpace=RoomInfo[i].FreeSpace-Required;
}
strcat_s(CourseRoom[index].Room, RoomInfo[i].Room);
strcat_s(CourseRoom[index].Room, ", ");
}
CourseRoom[index].Slot=AllocatedSlot;
}
}
else
{
SortRooms();
i=0;
while (CourseRoom[index].Size > RoomInfo[i].FreeSpace){
i++;}
CourseRoom[index].Slot=AllocatedSlot;
strcpy_s(CourseRoom[index].Room, RoomInfo[i].Room);
RoomInfo[i].FreeSpace=RoomInfo[i].FreeSpace-CourseRoom[index].Size;
}
}


void AssignCourseSlot()
{
int slot=1;
int i;
AssignSlot(0,1);
for(i=1; i<NoCourses;i++)
```

```
{
if (CourseRoom[i].Color != CourseRoom[i-1].Color)
{
slot++;
EmptyRooms();
}
AssignSlot(i,slot) ;


}


//Now assign slots those course to assigned slot as the assigned colour
int crs;
slot=NoColorUsed+1;
crs=0;
do
{
crs++;


if (CourseRoom[crs].Slot==0) //Check if the current course is not assigned slot
{
if (CourseRoom[crs].Color != CourseRoom[crs-1].Color ) // Make sure that courses
with diffent colours are signed to differet slots.
{
slot++;
EmptyRooms();
}
```

```
AssignSlot(crs.slot);
}

if (CourseRoom[crs].Slot==0) //Check if the current course is not assigned slot
{
slot++;
EmptyRooms();
crs--;
}

} while ((crs<NoCourses) && (slot<NoSlots));

}//END OF ASSIGNING ROOMS

int _tmain(int argc, _TCHAR* argv[])
{ ListofCourses *C=NULL;
ReadGraph();
CreateCourseList(C);
DisplayCourses(C);
CreateAdjMatrix(C);
GetInput();
Init();
Scheduling();
PrintOutput();
ReadRoomInfo();
ReadCourseInfo();
```

```
InitCourseAllocation();
SortCoursesColors();
AssignCourseSlot();
DisplayRoomAllocation();
return 0;

}
```