RICE UNIVERSITY

A Study of University Timetabling that Blends Graph Coloring with the Satisfaction of Various Essential and Preferential Conditions

by

Timothy Anton Redl

A Thesis Submitted
in Partial Fulfillment of the Requirements for the Degree
Doctor of Philosophy

Approved, Thesis Committee:

[Signatures and names of committee members]

Houston, Texas
May 2004
INFORMATION TO USERS

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleed-through, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

UMI Microform 3122528
Copyright 2004 by ProQuest Information and Learning Company. All rights reserved. This microform edition is protected against unauthorized copying under Title 17, United States Code.

ProQuest Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346
Abstract

A Study of University Timetabling that Blends Graph Coloring with the Satisfaction of Various Essential and Preferential Conditions

by

Timothy Anton Redl

Constructing a satisfactory conflict-free semester-long timetable of courses and creating a similarly satisfactory conflict-free timetable for end-of-semester final examinations are two closely related and often difficult problems that colleges and universities face each semester.

We discuss the relevance of such timetabling problems as a natural and practical application of graph coloring, and develop a mathematical and computational model for solving university timetabling problems using techniques of graph coloring that incorporates the satisfaction of both “essential” timetabling conditions (i.e., conditions or constraints that must be satisfied in order to produce a legal or feasible timetable) as well as suggested “preferential” timetabling conditions (i.e., additional conditions or constraints that need not necessarily be satisfied to produce a legal or legitimate timetable, but if satisfied may very well produce a more “acceptable” timetable for students and/or faculty members). 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 course data, to creating a course conflict graph based on the assembled data, to coloring the conflict graph, to transforming this coloring to a
conflict-free timetable, to finally assigning courses to classrooms. Once a conflict-free timetable of courses has been constructed, we present ways in which such a course timetable for a particular semester can be used to construct a conflict-free timetable of final examinations. Our model also considers a number of sociological scheduling concerns and preferences addressed by university registrars, faculty, staff, and students. Computational results, obtained by the author using actual data provided by Rice University and the University of St. Thomas, are documented.
Acknowledgements

"Don't forget your second wind... sooner or later you'll feel that momentum kick in..."

Billy Joel, “You’re Only Human (Second Wind)"

This dissertation is not a work of one but of many. There are a number of people I want to acknowledge and thank for all of their help along the way.

First and foremost, I thank God, with whom truly all things are possible. This thesis is definitely proof of that!

I thank my thesis advisors, Dr. Nathaniel Dean and Dr. Richard Tapia, for their expertise, encouragement, and support, and for confidently allowing me to continue in the CAAM Ph.D. program at Rice through to completion. I thank Dr. Keith Cooper and Dr. Steve Cox for volunteering their time to serve on my thesis committee and for offering valuable ideas and further guidance in my research, and Dr. Robert Bixby for allowing me use of the OPLStudio constraint programming software for my data analysis. This research was funded by grants from the George R. Brown School of Engineering and the CAAM Department at Rice University, the Los Alamos Computer Science Institute (LACSI), a gift from the Loewenstein family, and a fellowship from the Alliances for Graduate Education and the Professoriate (AGEP) Program at Rice University.

I thank Registrar Jerry Montag, Associate Registrar Laura Branch, and graduate student Alan Mislove at Rice University, Associate Registrar Patricia Cavanaugh and Analyst/Programmer Chloe Tran at the University of Houston, and Assistant Registrar Yvonne Obregon at the University of St. Thomas, for providing me insight into the process of timetabling at their respective universities. All of your contributions to my work are very much appreciated.
I thank my family — my parents Bill and Jeanne, my sister Melissa, and my grandma Millie — for their constant love and encouragement in all that I do, for helping me strive to reach my goals, and for making me who I am today and who I will become. I dedicate this thesis to the memory of my grandfather, William G. Redl (a.k.a. "Pop"). I know that somewhere up in heaven Pop is smiling and proudly sharing with everyone stories about the accomplishments of his grandchildren. I truly have the most fantastic family in the entire world. I love you guys so much!

I thank my bride-to-be, Melissa, for all of the love and patience and understanding she has given me during this final “home stretch” of my thesis work. Thank you Melissa for coming into my life and making me so happy. I love you and look forward to spending the rest of my life together with you.

I thank all of my dear friends who have been there for me during these past seven years. Melisa, I will never forget all the times we’ve shared since we entered Rice together back in 1997, and I thank you for your friendship and for all of your strength and confidence in me. Your enthusiasm and love for life is unparalleled. I thank my fellow graduate students and colleagues in CAAM, past and present, most notably Genetha, Heidi, Don, Rachel, Diane, Nikki, Bill, and my officemates Illya, Sanjeeb, Chris, Ed, and Alicia. It has been a pleasure knowing and working with each and every one of you. Thanks also to all of the CAAM professors, especially Dr. Bill Symes, who graciously took the time to visit with my father and me during our first trip to Rice back in 1996, while I was still a junior at Dartmouth curious about venturing off to graduate school, possibly deep in the heart of Texas.

A large part of my time in Houston and at Rice has been spent with the wonderful people I have met through Campus and Young Adult Ministry at Christ the King Lutheran Church. I thank Pastor Beth Warpmaeker and all the fellow students and young adults for allowing me to be “part of the group”. Allison, I know that we have been there for each other in the past and will continue to be there for each other in the future. You are a great friend and I wish nothing but the very best for you. To all
of the gang at CTK, thank you so much for welcoming me as your brother in Christ, and for all that you have done for me in sincere friendship.

Last, but not least, I must thank the “four-legged” friends in my life: MacDougal, Murie, Mati, Pumpkin, Penny, and Scitso. Thank you for letting me feed you and pet you and hug you and play with you (yes, even you, Scitso!). You guys are the greatest!

“So, before we end, and then begin, we’ll drink a toast to how it’s been…”

Billy Joel, “I’ve Loved These Days”
# Contents

Abstract ii

Acknowledgements iv

List of Figures ix

1 Introduction 1

2 Background 6

2.1 An Overview of Graph Coloring 6

2.2 Sequential Graph Coloring Algorithms 10

2.3 Graph Coloring as an Integer Linear Program 14

2.4 Graph Coloring as a Constraint Program using OPL and OPLStudio 17

3 University Timetabling 20

3.1 University Timetabling and Graph Coloring 20

3.2 A History of Graph Coloring Approaches to Timetabling 24

3.2.1 Vertex-Coloring Approaches 24

3.2.2 Edge-Coloring Approaches 31

3.3 Alternative Approaches to Solving Timetabling Problems 34

3.3.1 Mathematical Programming 34

3.3.2 Some Other Approaches 36
4 University Timetabling Methods Currently in Practice

4.1 Rice University (Houston, TX) .................................. 39
4.2 The University of Houston (Houston, TX) ................. 41
4.3 The University of St. Thomas (Houston, TX) .......... 42

5 A New Graph Coloring Model for University Timetabling

5.1 An Overview of the Model ............................................. 47
5.2 Assembling University Course Data ................................. 52
5.3 Creating a Course Conflict Graph .................................. 56
  5.3.1 Method A: One Course → One Vertex ....................... 56
  5.3.2 Method B: One Course → Multiple Vertices ............ 63
5.4 Coloring the Conflict Graph ........................................ 65
  5.4.1 Coloring a "Method A" Conflict Graph ......................... 65
    5.4.1.1 Coloring "Group 10" Vertices: Simple Method .... 71
    5.4.1.2 Coloring "Group 10" Vertices: Time Block Method 72
    5.4.1.3 Coloring "Group 10" Vertices: Time Slot Method .... 75
    5.4.1.4 Coloring "Group 10" Vertices: 2-or-3-Color Method 78
  5.4.2 Coloring a "Method B" Conflict Graph ....................... 80
  5.4.3 A Restricted (List) Coloring Approach ..................... 85
5.5 Transforming the Coloring to a Timetable ....................... 86
  5.5.1 "Minimizing Pain" ............................................ 87
  5.5.2 Color Classes and Time Slots ................................ 90
5.6 Assigning Courses to Classrooms .................................. 96
  5.6.1 Room Assignment from a "Method B" Conflict Graph .... 97
  5.6.2 Room Assignment from a "Method A" Conflict Graph .... 99
5.7 Some Final Notes on Final Exam Timetabling ..................... 104
  5.7.1 From Course Timetable to Final Exam Timetable ........ 104
  5.7.2 Minimizing Consecutive Exams via Integer Programming 105
  5.7.3 Packing Multiple Exams in a Single Classroom .......... 107
6 Computational Results
  6.1 Rice University .................................................. 112
      6.1.1 Rice University, Fall 2000 ............................... 112
      6.1.2 Rice University, Spring 2001 ............................ 124
  6.2 The University of St. Thomas .................................. 136
      6.2.1 The University of St. Thomas, Spring 2001 .............. 136

7 Conclusions and Future Work ................................. 148
  7.1 Conclusions ...................................................... 148
  7.2 Future Work ..................................................... 151

Bibliography ...................................................... 153
List of Figures

2.1 A 3-coloring of a graph $G$ ($\chi(G) = 3$) .......................... 7
2.2 A 3-edge-coloring of a graph $G$ ($\chi(G) = 3$) ..................... 8
2.3 An instance of the party problem ($\chi(G) = 3$) ..................... 10

3.1 A simple 5-course timetabling example ............................. 21
3.2 A 4-edge-coloring (4-teacher, 5-class timetabling example) ...... 33
3.3 A 4-period timetable (4-teacher, 5-class timetabling example) ... 33

5.1 Partitioning the $n$ vertices of $G$ into ten “groups” ............... 57
5.2 Courses $c_i$ and $c_j$ must be assigned different time slots (Method B) . 64
5.3 Courses $c_i$ and $c_j$ may be assigned the same time slot (Method B) . 65
5.4 Placing colored vertices from Groups 1-9 into six “time block groups” 73
5.5 Courses $c_i$ and $c_j$ are assigned different rooms and different time slots 82
5.6 Courses $c_i$ and $c_j$ are assigned the same room and different time slots 82
5.7 Courses $c_i$ and $c_j$ are assigned different rooms and different time slots 83
5.8 Courses $c_i$ and $c_j$ are assigned different rooms and the same time slot 83
5.9 Courses $c_i$ and $c_j$ are assigned the same room and different time slots 84
5.10 “Pain Table”: one-day, one-hour time slots (from [70]) ............ 87
5.11 “Pain Table”: multi-day, variable-length time slots (adapted from [70]) 89
5.12 Course room assignment: FFDra and BFDra .......................... 102
5.13 Exam room reassignment: fewer exam rooms used .................. 111
5.14 Exam room reassignment: larger exam rooms used .................. 111
6.1 Rice University, Fall 2000 (SSG:SO, Coloring time: 0.04 sec) .......... 114
6.2 Rice University, Fall 2000 (SSG:DD, Coloring time: 0.06 sec) .......... 114
6.3 Rice University, Fall 2000 (SSG:ID, Coloring time: 0.07 sec) .......... 115
6.4 Rice University, Fall 2000 (SSG:RO, Coloring time: 0.09 sec) .......... 115
6.5 Rice University, Fall 2000 (LFSG:SO, Coloring time: 0.03 sec) .......... 116
6.6 Rice University, Fall 2000 (LFSG:DD, Coloring time: 0.07 sec) .......... 116
6.7 Rice University, Fall 2000 (LFSG:ID, Coloring time: 0.07 sec) .......... 117
6.8 Rice University, Fall 2000 (LFSG:RO, Coloring time: 0.04 sec) .......... 117
6.9 Rice University, Fall 2000 (SFSG:SO, Coloring time: 0.04 sec) .......... 118
6.10 Rice University, Fall 2000 (SFSG:DD, Coloring time: 0.07 sec) .......... 118
6.11 Rice University, Fall 2000 (SFSG:ID, Coloring time: 0.07 sec) .......... 119
6.12 Rice University, Fall 2000 (SFSG:RO, Coloring time: 0.04 sec) .......... 119
6.13 Rice University, Fall 2000 (RSG:SO, Coloring time: 0.03 sec) .......... 120
6.14 Rice University, Fall 2000 (RSG:DD, Coloring time: 0.07 sec) .......... 120
6.15 Rice University, Fall 2000 (RSG:ID, Coloring time: 0.06 sec) .......... 121
6.16 Rice University, Fall 2000 (RSG:RO, Coloring time: 0.04 sec) .......... 121
6.17 Rice University, Fall 2000 (OPLStudio, Coloring time: 40.71 sec) ...... 122
6.18 Rice University, Spring 2001 (SSG:SO, Coloring time: 0.04 sec) ...... 126
6.19 Rice University, Spring 2001 (SSG:DD, Coloring time: 0.07 sec) ...... 126
6.20 Rice University, Spring 2001 (SSG:ID, Coloring time: 0.07 sec) ...... 127
6.21 Rice University, Spring 2001 (SSG:RO, Coloring time: 0.04 sec) ...... 127
6.22 Rice University, Spring 2001 (LFSG:SO, Coloring time: 0.03 sec) ...... 128
6.23 Rice University, Spring 2001 (LFSG:DD, Coloring time: 0.07 sec) ...... 128
6.24 Rice University, Spring 2001 (LFSG:ID, Coloring time: 0.07 sec) ...... 128
6.25 Rice University, Spring 2001 (LFSG:RO, Coloring time: 0.05 sec) ...... 129
6.26 Rice University, Spring 2001 (SFSG:SO, Coloring time: 0.03 sec) ...... 130
6.27 Rice University, Spring 2001 (SFSG:DD, Coloring time: 0.07 sec) ...... 130
6.28 Rice University, Spring 2001 (SFSG:ID, Coloring time: 0.07 sec) ...... 131
6.29 Rice University, Spring 2001 (SFSG:RO, Coloring time: 0.05 sec) ... 131
6.30 Rice University, Spring 2001 (RSG:SO, Coloring time: 0.04 sec) ... 132
6.31 Rice University, Spring 2001 (RSG:DD, Coloring time: 0.07 sec) ... 132
6.32 Rice University, Spring 2001 (RSG:ID, Coloring time: 0.07 sec) ... 133
6.33 Rice University, Spring 2001 (RSG:RO, Coloring time: 0.04 sec) ... 133
6.34 Rice University, Spring 2001 (OPLStudio, Coloring time: 55.15 sec) ... 134
6.35 U. of St. Thomas, Spring 2001 (SSG:SO, Coloring time: 0.01 sec) ... 138
6.36 U. of St. Thomas, Spring 2001 (SSG:DD, Coloring time: 0.01 sec) ... 138
6.37 U. of St. Thomas, Spring 2001 (SSG:ID, Coloring time: 0.03 sec) ... 139
6.38 U. of St. Thomas, Spring 2001 (SSG:RO, Coloring time: 0.02 sec) ... 139
6.39 U. of St. Thomas, Spring 2001 (LFSG:SO, Coloring time: 0.01 sec) ... 140
6.40 U. of St. Thomas, Spring 2001 (LFSG:DD, Coloring time: 0.02 sec) ... 140
6.41 U. of St. Thomas, Spring 2001 (LFSG:ID, Coloring time: 0.02 sec) ... 141
6.42 U. of St. Thomas, Spring 2001 (LFSG:RO, Coloring time: 0.01 sec) ... 141
6.43 U. of St. Thomas, Spring 2001 (SFSG:SO, Coloring time: 0.00 sec) ... 142
6.44 U. of St. Thomas, Spring 2001 (SFSG:DD, Coloring time: 0.02 sec) ... 142
6.45 U. of St. Thomas, Spring 2001 (SFSG:ID, Coloring time: 0.03 sec) ... 142
6.46 U. of St. Thomas, Spring 2001 (SFSG:RO, Coloring time: 0.02 sec) ... 143
6.47 U. of St. Thomas, Spring 2001 (RSG:SO, Coloring time: 0.01 sec) ... 144
6.48 U. of St. Thomas, Spring 2001 (RSG:DD, Coloring time: 0.01 sec) ... 144
6.49 U. of St. Thomas, Spring 2001 (RSG:ID, Coloring time: 0.01 sec) ... 145
6.50 U. of St. Thomas, Spring 2001 (RSG:RO, Coloring time: 0.01 sec) ... 145
6.51 U. of St. Thomas, Spring 2001 (OPLStudio, Coloring time: 23.88 sec) ... 146
Chapter 1

Introduction

“Class schedules are designed so that every student will waste the maximum time between classes.”

First Law of Class Scheduling

“A pre-requisite for a desired course will only be offered during the semester following the desired course.”

Second Law of Class Scheduling

“When you are occasionally able to schedule two classes in a row, they will be held in classrooms at opposite ends of the campus.”

(Third) Law of Class Scheduling

“The one course you must take to graduate will not be offered during your last semester.”

Seit’s Law of Higher Education

(The above are corollaries of Murphy’s Law: “If anything can go wrong, it will.” [1])
Generally speaking, timetabling is the allocation, subject to constraints, of given resources to objects being placed in space-time, in such a way as to satisfy as nearly as possible a set of desirable objectives [71]. Specifically, the university timetabling problem for courses and/or final examinations can be viewed as fixing in time and space a sequence of meetings between instructors and students, while simultaneously satisfying a number of various essential or preferential conditions or constraints.

The chief focus and objective of this dissertation is to improve the understanding of and ability to both model and solve university course and final examination timetabling problems using techniques of graph coloring. This improvement is expressed from both mathematical and computational points of view, and to some extent a sociological point of view. We begin by highlighting some of the major published research regarding problems in timetabling, in particular two closely related problems that colleges and universities face each semester: 1) constructing a satisfactory conflict-free semester-long timetable of courses; and 2) creating a similarly satisfactory conflict-free timetable for end-of-semester final examinations. We discuss the relevance of such timetabling problems as a natural and practical application of graph coloring, and present various existing theoretical and computational results, including alternative problem formulations and algorithms, which can be used in attempting to solve these problems. We provide a detailed description and analysis of course and final examination scheduling methods currently in practice at three universities of varying sizes and types in Houston, Texas: 1) Rice University; 2) the University of Houston; and 3) the University of St. Thomas. We conclude with the development of a mathematical and computational model for solving university timetabling problems using techniques of graph coloring that incorporates the satisfaction of both "essential" timetabling conditions (conditions or constraints that must be satisfied in order to produce a legal or feasible timetable) as well as suggested "preferential" timetabling conditions (additional conditions or constraints that need not necessarily be satisfied to produce a legal or legitimate timetable, but if satisfied
may very well produce a more “acceptable” timetable for students and/or faculty members. This model involves creating a conflict graph from the input university course data, properly coloring the vertices of the conflict graph, and transforming this proper coloring into a conflict-free timetable of courses. From this conflict-free course timetable one can then assign the courses to classrooms based on room capacity and availability. The model takes advantage of the structural properties of conflict graph instances that naturally arise from university timetabling problems, and is based on the effectiveness of a variety of graph coloring approaches such as intelligently-ordered and intelligently-searched sequential coloring methods, as well as integer and constraint programming formulations of graph coloring, in solving such problems. It also considers a number of sociological scheduling concerns and preferences expressed by university registrars, faculty, staff, and students. Computational results, obtained by the author using actual data provided by Rice University and the University of St. Thomas, are documented.

As a means of a descriptive outline, this dissertation proceeds as follows:

In Chapter 2, we provide an overview of some basic graph theory definitions and of graph coloring, as well as a description of several variations of a greedy or sequential graph coloring algorithm, as background for this dissertation. We then present two mathematical programming formulations of graph coloring, the first being an integer linear programming formulation, and the second a constraint programming formulation.

In Chapter 3, we introduce the timetabling problem, and in particular two closely related timetabling problems regularly encountered at a college or university, namely the timetabling of courses for a specific semester, as well as the timetabling of final examinations to take place at the end of the semester. We explain the connection between timetabling and graph coloring, as timetabling is indeed both a natural and
practical application of graph coloring, and an equivalence relationship between the two has been studied. We then survey and highlight some of the major published research regarding graph coloring approaches to solving timetabling problems, as well as alternative approaches such as mathematical programming and other techniques, either classical or more non-traditional.

In Chapter 4, we describe and analyze course and final examination scheduling methods currently at practice at Rice University, the University of Houston, and the University of St. Thomas, based on interviews conducted with staff in either the Office of the Registrar or the Department of Registration and Academic Records at these universities.

In Chapter 5, we detail a new mathematical and computational model for solving university timetabling problems that blends graph coloring with the satisfaction of both “essential” and “preferential” timetabling conditions, in an effort to produce a “satisfactory” and “acceptable” timetable. We describe such essential and preferential timetabling conditions, obtained either from the existing literature or by way of interviews conducted by the author with university and college students, faculty members, and other staff, including registrars and scheduling assistants. We then discuss in detail the step-by-step process that is taken to implement our timetabling-by-graph-coloring procedure, from the assembling of university course data, to creating a course conflict graph based on the assembled data, to coloring the conflict graph, to transforming this coloring to a conflict-free timetable, to finally assigning courses to classrooms. Once a conflict-free timetable of courses has been constructed, we present ways in which such a course timetable for a particular semester can be used to construct a similarly conflict-free timetable of final examinations.

In Chapter 6, we present some computational timetabling results and other observations obtained by the author using aspects of the new timetabling model discussed in Chapter 5 and data provided by the Office of the Registrar at Rice University and at the University of St. Thomas, along with an analysis of said results and
observations.

Finally, in Chapter 7, we provide concluding remarks, as well as possible and proposed directions for future research in this area.
Chapter 2

Background

2.1 An Overview of Graph Coloring

A graph $G$ with $n$ vertices and $m$ edges consists of a vertex set $V(G) = \{v_1, v_2, \ldots, v_n\}$ and an edge set $E(G) = \{e_1, e_2, \ldots, e_m\}$. Each edge of $E(G)$ consists of two (possibly equal) vertices of $V(G)$. For an edge $e = \{x, y\}$ (which we often abbreviate by writing "$e = xy$"), the vertices $x$ and $y$ are called endpoints of $e$. If the edge $xy \in E(G)$, then we say that $x$ and $y$ are adjacent in $G$. A loop is an edge whose endpoints are equal. The degree of a vertex $v$ is the number of non-loop edges containing it plus twice the number of loops containing it. Multiple edges (also called parallel edges) are edges that share the same pair of endpoints. A graph that does not allow loops or multiple edges is called a simple graph. It is clear that every simple graph is a graph, but not every graph is a simple graph. In this dissertation, each graph $G$ is finite (i.e., $G$ has a finite vertex set and finite edge set) and undirected (i.e., for each edge $xy \in E(G)$, $xy = yx$), with no loops and no multiple edges allowed.

Given a graph $G$, a (vertex-) coloring of $G$ is a function $f$ from the vertices of $G$ to a set $C$ whose elements are called colors. It is often both conventional and convenient to use numbers $1, 2, \ldots$ for the colors. A proper $k$-coloring of $G$ is a coloring $f$ which uses exactly $k$ colors and satisfies the property that $f(x) \neq f(y)$ whenever $x$ and
y are adjacent in $G$. We say such a graph $G$ is $k$-colorable. Note that if $G$ has a loop, then $G$ has no proper coloring, since $f(x) = f(x)$. The chromatic number $\chi(G)$ of $G$ is the minimum number $k$ such that there exists a proper $k$-coloring of $G$. A clique $K_r$ of $G$ is an $r$-vertex subgraph of $G$ in which each pair of vertices in $K_r$ share an edge. For example, $K_2$ is a single edge, and $K_3$ is a triangle. It is easy to see that the size of the maximum clique $K_r$ of $G$ gives a lower bound on the chromatic number of $G$, since $\chi(K_r) = r$ and therefore $\chi(G) \geq r$. It is also easy to see that $\chi(G) \leq \Delta(G) + 1$, where $\Delta(G)$ denotes the maximum degree of $G$. A minimum coloring of $G$ is a proper coloring that uses as few colors as possible; that is, $\chi(G)$ colors are both necessary and sufficient for a proper coloring of $G$. Many times one is presented with a graph and must determine its chromatic number as well as produce a minimum coloring of the graph. This problem is known in graph coloring as the minimum coloring problem. The minimum coloring problem is formally NP-hard for general graphs. The decision problem of graph $k$-colorability is NP-complete in the general case for fixed $k \geq 3$. For $k = 2$ (bipartite graphs), the graph $k$-colorability problem can be solved in polynomial time [32]. Figure 2.1 below shows a minimum 3-coloring of a graph $G$ with five vertices and five edges.

![Figure 2.1: A 3-coloring of a graph $G$ ($\chi(G) = 3$)](image)

Edge-colorings of graphs are similar to vertex-colorings. Given a graph $G$, an edge-coloring of $G$ is a function $f'$ from the edges of $G$ to a set $C$ of elements called colors. Again, we often use numbers $1, 2, \ldots$ for the colors. A proper $k$-edge-coloring of $G$ is a coloring $f'$ which uses exactly $k$ colors and satisfies the property that $f'(x) \neq f'(y)$
whenever $x$ and $y$ share a vertex in $G$. We say such a graph $G$ is $k$-edge-colorable. Each color class of $G$ corresponds to a matching of $G$. A matching of a graph $G$ is a set $M$ of pairwise disjoint edges in $G$. The edge-chromatic number $\chi'(G)$ of $G$ is the minimum number $k$ such that there exists a proper $k$-edge-coloring of $G$. A minimum edge-coloring of $G$ is a proper edge-coloring that uses as few colors as possible; that is, $\chi'(G)$ colors are both necessary and sufficient for a minimum edge-coloring of $G$. Figure 2.2 below shows a minimum 3-edge-coloring of a graph $G$ with five vertices and five edges.

![Diagram](image)

Figure 2.2: A 3-edge-coloring of a graph $G$ ($\chi(G) = 3$)

When discussing a vertex-coloring $f$ of a graph $G$, it is common to drop the word “vertex” and just refer to $f$ as a “coloring” of $G$. When discussing an edge-coloring $f'$ of a graph $G$, however, it is common to explicitly refer to $f'$ as an “edge-coloring” of $G$. In this dissertation, we too will follow this naming convention.

Graph coloring deals with the fundamental problem of partitioning a set of objects into classes according to certain prescribed rules, and is of particular interest for its applications. One such application, namely scheduling and timetabling, has given rise to the subject matter of this dissertation. Two other applications of graph coloring are register allocation and frequency assignment problems. The register allocation problem [7], [8], [9], [18] involves an interference-free assignment of variables to a limited number of computer hardware registers during a program’s execution. Variables in registers can be accessed much more quickly than those not in registers. Typically, however, there are far more variables than available registers, so it becomes necessary to assign multiple variables to registers. Two variables are said to conflict with each
other if one is used both before and after the other within a short period of time (for example, within a subroutine). The goal is to assign variables that do not conflict so as to minimize the use of non-register memory. The frequency assignment problem [31] involves the assignment of a minimum number of frequencies to mobile radios and other users of the electro-magnetic spectrum in such a way that two customers that are sufficiently close must be assigned different radio frequencies, while those that are sufficiently distant can share the same frequency. Indeed, many practical situations can be modeled and solved as graph coloring problems. The general approach involves forming a graph (often called a conflict graph or an interference graph) with the vertices representing the items of interest, and an edge connecting each pair of “incompatible” or “conflicting” items, and then coloring it (i.e., solving an instance of the minimum coloring problem). The minimum coloring problem thus becomes the problem of assigning a color to each item so that items in each incompatible pair are assigned different colors, using the fewest number of colors possible. Consider, for example, the well-known party problem. The party problem can be described as follows: given a group of $n$ persons $p_1, p_2, \ldots, p_n$ at a party, along with a list of pairs of incompatible persons, assign the $n$ individual persons to rooms such that no two persons in the same room are incompatible, and to do this using as few rooms as possible. The conflict graph $G$ in Figure 2.3 below illustrates a solution to a small instance of the party problem with five persons (Elaine, George, Kramer, Jerry, and Newman) and five such incompatibilities. Each vertex of $G$ represents a particular person at the party and each edge represents an incompatibility between two persons. By coloring the vertices of $G$ and observing that $\chi(G) = 3$, we see that a partitioning of the group of five people into three subgroups (in this case, assigning both Elaine and Jerry to Room 1, assigning George to Room 2, and assigning both Kramer and Newman to Room 3) is both necessary and sufficient to achieve compatibility among all members of a particular subgroup, and thereby solve this instance of the party problem.
2.2 Sequential Graph Coloring Algorithms

Many algorithms and heuristics exist for coloring the vertices of a graph, the most common of which is arguably the sequential graph coloring approach. Leighton [46] describes several such sequential algorithms, including the randomly ordered sequential (RND) graph coloring algorithm. Given a graph \( G = (V, E) \), the RND algorithm randomly orders the vertices so that \( V = \{v_1, \ldots, v_n\} \) and then assigns colors to the vertices in the following manner. The first vertex, \( v_1 \), is assigned color number 1. Once the first \( i \) vertices have been colored (\( 1 \leq i \leq n - 1 \)), \( v_{i+1} \) is assigned the lowest possible color number such that no previously colored vertex adjacent to \( v_{i+1} \) has been assigned the same color number. For any graph \( G \), there exists an ordering of the vertices for which RND will produce an optimal (minimum) coloring of \( G \), while there may exist another particular vertex ordering that will lead RND to compute an extremely poor coloring of \( G \). Therefore, the problem of finding an optimal initial ordering of the vertices of a graph is equivalent to the problem of optimally coloring the graph.

This result led to the development of several new sequential coloring algorithms which differ from RND only in the method of initially ordering the vertices of the graph. Two such algorithms are largest-first (LF) and smallest-last (SL). The LF algorithm orders the vertices such that \( d(v_i) \geq d(v_{i+1}) \) for \( 1 \leq i < n \) where \( V = \{v_1, \ldots, v_n\} \). An SL ordering recursively orders the vertices of smallest degree last.
An SL ordering is one in which \( d(v_n) = \min_{w \in V} d(w) \) and for \( n - 1 \geq i \geq 1 \), \( d_U(v_i) = \min_{w \in U} d_U(w) \), where \( U = V - \{v_n, \ldots, v_{i+1}\} \). Each of these three sequential coloring algorithms requires \( O(n^2) \) time and \( O(n^2) \) space to color a graph with \( n \) vertices. Successful variations of LF and SL, namely LFI and SLI respectively, incorporate an *interchange* process which allows for the switching of the colors of a pair of vertices during execution of the algorithm. These variations each require \( O(n^3) \) time and \( O(n^2) \) space to color an \( n \)-vertex graph. The recursive-largest-first (RLF) algorithm is yet another sequential coloring algorithm. At each step in the RLF procedure, a vertex is selected for coloring which will, in some sense, leave the resulting uncolored vertices colorable in as few colors as possible. The procedure also completes the assignment of color \( i \) before beginning assignment of color \( i + 1 \). In general, the RLF algorithm requires \( O(n^3) \) time and \( O(n^2) \) space to color a graph with \( n \) vertices, but only \( O(n^2) \) time to color graphs for which \( ke \approx n^2 \), where \( k \) is the number of colors used to color the graph, and \( e \) is the number of edges in the graph. Such graphs, which are usually sparse, commonly arise in applications such as final examination scheduling.

For example, a graph associated with the 1977-78 Princeton University fall term final examinations schedule (273 nodes, 6727 edges, a density of approximately 1/6) requires 17 colors when properly colored by the RLF algorithm [46].

Sequential graph coloring algorithms are commonly referred to as *greedy* algorithms. A greedy graph coloring approach takes each vertex in turn in some particular order and tries to color the vertex with one of the colors used so far; that is, it tries to add the vertex to one of the existing color classes. If it is not possible to color the vertex with any existing color, then a new color class is created and the vertex is assigned the color of that new class. Below we present a description of four "color search" variations of a greedy approach to graph coloring. The resulting four greedy algorithms (SIMPLE-SEARCH GREEDY, LARGEST-FIRST-SEARCH GREEDY, SMALLEST-FIRST-SEARCH GREEDY, and RANDOM-SEARCH GREEDY) differ only in how to make the choice of which (existing) color class to use for a particular
vertex. When such a choice of available colors exists for a particular vertex, each algorithm selects the first color class encountered, according to the respective "color searching procedure" for that algorithm.

SIMPLE-SEARCH GREEDY (SSG) visits the color classes in the predefined order 1, \ldots, k. The SSG search method is the search method used in the algorithms of Leighton [46] described above, as well as in many of the graph coloring heuristics for solving timetabling problems described later in Chapter 3 of this dissertation. Indeed, it is the technique usually indicated when someone generally refers to the "sequential algorithm".

LARGEST-FIRST-SEARCH GREEDY (LFSG) ranks the color classes by the number of vertices currently in them, and searches this ranked list of color classes in order by descending size. Based on this searching heuristic, LFSG should tend to build larger-sized color classes.

SMALLEST-FIRST-SEARCH GREEDY (SFSG) ranks the color classes by the number of vertices currently in them, and searches this ranked list of color classes in order by ascending size. Based on this searching heuristic, SFSG should tend to balance the size of the color classes.

RANDOM-SEARCH GREEDY (RSG) searches the color classes in a purely random order.

Each of the four greedy graph coloring algorithms above was implemented with the ability to accommodate each of four different initial orderings of the vertices of the graph: 1) a specific predefined ordering; 2) an ordering by decreasing degree (i.e., largest-degreed vertices first); 3) an ordering by increasing degree (i.e., smallest-degreed vertices first); and 4) a purely random ordering of the vertices. (Recall that for any graph \( G \), there exists a particular vertex ordering for which a greedy algorithm will produce an optimal (minimum) coloring of \( G \), while there may exist another particular vertex ordering that will lead that same greedy algorithm to compute an extremely poor coloring of \( G \), and the problem of finding a desired coloring of a graph
is intimately connected to the problem of finding a desired initial vertex ordering.)

We thus have implementations of 16 different variations of the greedy approach to graph coloring:

1) SIMPLE-SEARCH GREEDY: SPECIFIED ORDER (SSG:SO);
2) SIMPLE-SEARCH GREEDY: DECREASING DEGREE (SSG:DD);
3) SIMPLE-SEARCH GREEDY: INCREASING DEGREE (SSG:ID);
4) SIMPLE-SEARCH GREEDY: RANDOM ORDER (SSG:RO);
5) LARGEST-FIRST-SEARCH GREEDY: SPECIFIED ORDER (LFSG:SO);
6) LARGEST-FIRST-SEARCH GREEDY: DECREASING DEGREE (LFSG:DD);
7) LARGEST-FIRST-SEARCH GREEDY: INCREASING DEGREE (LFSG:ID);
8) LARGEST-FIRST-SEARCH GREEDY: RANDOM ORDER (LFSG:RO);
9) SMALLEST-FIRST-SEARCH GREEDY: SPECIFIED ORDER (SFSG:SO);
10) SMALLEST-FIRST-SEARCH GREEDY: DECREASING DEGREE (SFSG:DD);
11) SMALLEST-FIRST-SEARCH GREEDY: INCREASING DEGREE (SFSG:ID);
12) SMALLEST-FIRST-SEARCH GREEDY: RANDOM ORDER (SFSG:RO);
13) RANDOM-SEARCH GREEDY: SPECIFIED ORDER (RSG:SO);
14) RANDOM-SEARCH GREEDY: DECREASING DEGREE (RSG:DD);
15) RANDOM-SEARCH GREEDY: INCREASING DEGREE (RSG:ID); and

To test the effects of a variety of different color searching procedures as well as the effects of a variety of different initial vertex orderings on graph coloring performance, we conducted computational experiments involving each of the above 16 greedy algorithm variations and the coloring of graphs arising from data from actual university timetabling problems. Results of such experiments, along with an analysis of said results, appear in Chapter 6 of this dissertation. Indeed, a greedy graph coloring approach involving one or more of the above algorithmic variations will be the back-
bone of our model for solving university timetabling problems that we develop and describe in Chapter 5 of this dissertation.

2.3 Graph Coloring as an Integer Linear Program

An integer program is a discrete optimization problem of the form

Minimize (or Maximize) \( F(x_1, x_2, \ldots, x_n) \)

subject to a set of \( m \) equality constraints

\[
\begin{align*}
g_1(x_1, x_2, \ldots, x_n) &= b_1 \\
g_2(x_1, x_2, \ldots, x_n) &= b_2 \\
&
\vdots \\
g_m(x_1, x_2, \ldots, x_n) &= b_m
\end{align*}
\]

and \( k \) inequality constraints

\[
\begin{align*}
h_1(x_1, x_2, \ldots, x_n) &\leq r_1 \\
h_2(x_1, x_2, \ldots, x_n) &\leq r_2 \\
&
\vdots \\
h_k(x_1, x_2, \ldots, x_n) &\leq r_k
\end{align*}
\]

In addition, the values of the decision variables \( x_1, x_2, \ldots, x_n \) must be integers; no fractional values of \( x_1, x_2, \ldots, x_n \) are permitted. In an integer program, the objective function \( F \) and the constraint functions \( g_1, g_2, \ldots, g_m \) and \( h_1, h_2, \ldots, h_k \) may be linear or nonlinear. If we restrict the objective function \( F \) and the constraint functions \( g_1, g_2, \ldots, g_m \) and \( h_1, h_2, \ldots, h_k \) to be linear, then we have an integer linear program. Integer linear programming formulations are much more easily handled in computation than integer (nonlinear) programming formulations.
Below is an example of an integer programming (IP) formulation of graph coloring which contains nonlinear constraints. In this formulation, $G$ is the graph we wish to properly color. The number of vertices in $G$ is denoted by $n$, and the number of edges in $G$ is denoted by $e$. We use $k$ to represent the number of colors we wish to use to properly color $G$. The values of $n$, $e$, and $k$ are known constants which serve as parameters in the model. The formulation addresses the following two questions: Given a graph $G$ and a number of colors $k$, does there exist a proper $k$-coloring of $G$? If such a $k$-coloring of $G$ exists, what is an example of such a coloring?

**IP Formulation of Graph Coloring:**

\[
\text{minimize } 0
\]

subject to

1. \(1 \leq x_i \leq k; \quad i = 1, 2, \ldots, n\)

2. \(1 \leq |x_i - x_j|\) for each edge $ij \in E(G)$

3. $x_1, \ldots, x_n$ are integer-valued

The above formulation contains $n$ integer variables $x_1, \ldots, x_n$, each representing one of the $n$ vertices of $G$. A feasible solution \(\{x_1, \ldots, x_n\}\) gives a proper $k$-coloring of $G$, if indeed one does exist. The three constraints define precisely what it means for a graph to exhibit a proper $k$-coloring. Constraint 1 maintains that each variable $x_i$ must receive a value between 1 and $k$; that is to say, each vertex of $G$ must be colored using a color from 1 to $k$. Constraint 2 further illustrates the definition of a proper coloring of $G$. For each edge of $G$, the vertices corresponding to that edge must be colored using different colors. Finally, constraint 3 requires that the values of the variables $x_1, \ldots, x_n$ be integers. Obviously this condition must hold since our $k$ colors are numbered as integers $1, 2, \ldots, k$.

Notice that in our formulation, we are only concerned with obtaining constraint feasibility, not a minimization or maximization of a specific objective function, and
can therefore employ the use of a “dummy” (i.e., constant) objective function. The objective “minimize 0” is used here. If we can obtain a solution \( \{x_1, \ldots, x_n\} \) that satisfies all of the above constraints for a particular graph \( G \) and given \( k \), then \( G \) has a proper \( k \)-coloring and \( \{x_1, \ldots, x_n\} \) is such a coloring.

We next present an example of an integer linear programming (ILP) formulation of graph coloring based on the above integer programming formulation. Recall that in an integer linear programming formulation all constraints must be linear. In the above IP formulation, constraint 2 is nonlinear. To establish a proper ILP formulation, we first introduce \( e \) additional integer variables \( z_1, \ldots, z_e \), each representing one of the edges in \( G \), where we define \( z_m = |x_i - x_j| \), for \( m = 1, 2, \ldots, e \). We then perform constraint linearization using a common “big \( M \)” technique illustrated by Hillier and Lieberman [35] and obtain the following ILP formulation. Here \( M \) represents a “very large positive number”. Notice the necessary addition of \( e \) binary variables to the formulation, one for each edge in \( G \). These binary variables are denoted by \( y_b \), where \( b = 1, 2, \ldots, e \).

**ILP Formulation of Graph Coloring:**

\[
\text{minimize } 0
\]

subject to

1. \( 1 \leq x_i \leq k; \quad i = 1, 2, \ldots, n \)

2. \( z_m - x_i + x_j - M y_b \leq 0, \)
   \( z_m + x_i - x_j + M y_b \leq M, \)
   \( z_m - x_i + x_j \geq 0 \), and
   \( z_m + x_i - x_j \geq 0 \)
   for each edge \( ij \in E(G); \quad m = 1, 2, \ldots, e, \quad b = 1, 2, \ldots, e \)

3. \( 1 \leq z_m; \quad m = 1, 2, \ldots, e \)

4. \( x_1, \ldots, x_n \) are integer-valued
5. \( y_1, \ldots, y_e \) are binary-valued

Like the IP formulation, the ILP formulation uses a constant objective function, as again we are only concerned with constraint feasibility. Satisfaction of the constraints in the ILP formulation guarantees the existence of a proper \( k \)-coloring of \( G \), as illustrated by the solution \( \{x_1, \ldots, x_n\} \).

The ILP formulation above contains \( n + e \) integer variables, \( e \) binary variables, \( 4e \) linear constraints, and \( n + e \) variable bounds. It can be used in conjunction with integer linear programming optimizing software packages such as CPLEX [36] to test the \( k \)-colorability of a graph \( G \), generate a proper \( k \)-coloring of \( G \) (if one exists), and attempt to determine, by gradually reducing the value of \( k \) and repeating the above test, the chromatic number of \( G \).

### 2.4 Graph Coloring as a Constraint Program using OPL and OPLStudio

As illustrated above, we can model graph coloring both as an integer program and as an integer linear program. While both formulations are certainly valid models, they present the problem of proving the existence of a \( k \)-coloring of a graph \( G \) as an optimization problem (requiring in this case the presence of an useless objective function), rather than as purely a constraint satisfaction problem. For this reason, we find it advantageous to turn to a constraint programming formulation of graph coloring, one that does not require an objective function nor linearity among its constraints. Below is a simple example of such a constraint programming (CP) formulation, given a graph \( G \) with \( n \) vertices and \( e \) edges, and a set of \( k \) colors \( 1, 2, \ldots, k \).

**CP Formulation of Graph Coloring:**

Find a color label \( \text{color}[x_i] \) for each vertex \( x_i \in V(G), i = 1, \ldots, n \)

such that
1. \( \text{color}[x_i] \neq \text{color}[x_j] \) for each edge \( x_i x_j \in E(G) \)

2. \( 1 \leq \text{color}[x_i] \leq k; \ i = 1, \ldots, n \)

3. \( \text{color}[x_i] \) is an integer; \( i = 1, \ldots, n \)

The above CP formulation has \( n \) variables and \( e \) constraints, and can be used in conjunction with constraint programming-compatible languages such as OPL [37] and constraint programming computational solvers such as OPLStudio [38] to test the \( k \)-colorability of a graph \( G \), generate a proper \( k \)-coloring of \( G \) (if one exists), and attempt to determine, by gradually reducing the value of \( k \) and repeating the above test, the chromatic number of \( G \).

We conclude this section and chapter with a simple example of a constraint program written in OPL based on the above CP formulation for graph coloring, followed immediately by the resulting solution when solved by OPLStudio. The program tests the 4-colorability of a graph \( G \) with six vertices and nine edges, and provides a proper 4-coloring of \( G \).

```plaintext
enum Vertex {x1, x2, x3, x4, x5, x6};
enum Colors {1, 2, 3, 4};
var Colors color[Vertex];
solve {
    color[x3] <> color[x1];
    color[x3] <> color[x6];
    color[x3] <> color[x4];
    color[x6] <> color[x4];
    color[x6] <> color[x1];
    color[x1] <> color[x5];
    color[x1] <> color[x4];
    color[x4] <> color[x5];
```
color[x4] <> color[x2];
}

Solution [1]
color[x1] = 1;
color[x2] = 2;
color[x3] = 2;
color[x4] = 3;
color[x5] = 2;
color[x6] = 4;
Chapter 3

University Timetabling

3.1 University Timetabling and Graph Coloring

As mentioned in Section 2.1, one important application of graph coloring is timetabling. Many problems in timetabling involve various pairwise restrictions on the items being scheduled; that is, there exist restrictions on which items can be scheduled to take place simultaneously. If one is attempting to construct a schedule of courses at a college or university, it is obvious that two courses taught by the same professor cannot be scheduled for the same time slot. Also, two courses that require the same classroom must be scheduled at different times. Furthermore, suppose a particular student or group of students is required by the curriculum to take two different but related courses (e.g., physics and calculus) concurrently during a semester. In this case, these courses also need to be scheduled at different times in order to avoid conflict. The problem of determining the minimum number (or a reasonable number) of time slots needed to schedule all the courses subject to restrictions such as those above is a graph coloring problem. Figure 3.1 illustrates an example of a simple timetabling problem instance in which we have five courses to be scheduled: $a$, $b$, $c$, $d$, and $e$. In the chart on the left, an asterisk indicates those pairs of courses that would cause a timetabling conflict if both were scheduled at the same time. The cause for potential
conflict could be any of the restrictions discussed above. For example, courses $b$ and $c$ might be taught by the same professor, or courses $d$ and $e$ might require the same classroom. Given the list of courses $a$, $b$, $c$, $d$, and $e$ along with the set of potential conflicts, we can create a conflict-free timetable of courses by transforming the table on the left to the corresponding conflict graph $G$ on the right, and finding a minimum coloring. A vertex in $G$ represents a course, an edge represents a pair of courses that conflict, and a color represents the period in which that particular course is to be scheduled. We see that four periods are required to schedule all the courses without conflict: $\chi(G) = 4$. According to the coloring, we can schedule courses $a$ and $e$ for Period 1, and courses $b$, $c$, and $d$ for Periods 2, 3, and 4 respectively.

![Figure 3.1: A simple 5-course timetabling example](image)

A similar situation arises in attempting to create a timetable of final examinations at the end of a semester. Is there a professor who needs to proctor final examinations for two different courses? Are there final examinations for two courses that each require the same classroom, though not necessarily at the same time? Is there a particular student or group of students taking final examinations for two different courses? Each of these three cases presents a potential scheduling conflict. The problem of obtaining a conflict-free timetable for final examinations is yet another graph coloring problem.
Further similarities and differences exist between the above two timetabling problems (course scheduling and final examination scheduling). While the two problems appear to be essentially the same, certain particular nuances exist in each case. For example, the conflict graph representing a course scheduling problem for a particular semester may often be of larger order (i.e., contain more vertices) than the graph representing the same semester's final examination timetable, simply because certain courses may not have final exams. Also, in creating course schedules, one may want to allow for a course to have multiple sections or several occurrences of the same lecture, in an attempt to relax the problem and hopefully help to further eliminate conflicts. When scheduling final examinations, one may be faced with tighter restrictions, such as a limited number of days in the examination period, or extra constraints that require students to have at most one exam per day or to avoid having exams during consecutive periods or on consecutive days, if possible.

With course timetabling, it is often desirable that courses do not "student-conflict" (i.e., that two courses sharing a common student will not be scheduled at the same time). However, in most situations, a course timetabling solution without some "student conflict" does not exist, due to the fact that university courses are almost always scheduled prior to when students choose their courses. This simply means that a student, when planning his or her semester of courses in which to enroll, may very well have to choose between two or more initially desired courses that are scheduled to take place at the same time, to resolve this "student-conflict". [Note: The author, is aware of a rare but true situation in which a particular student (not the author himself, mind you) decided to enroll in and attempted to take two courses which actually met at the same time during a semester! He elected to attend the classes of one of those two courses that semester, while trying to "teach himself" the material from the other course by simply reading and studying the material in the textbook. His "experiment" lasted only two weeks into the semester, until he failed the first exam in the course in which he hadn’t been attending any of the lectures. He subsequently
dropped that course and was convinced that enrolling in two courses that met at
the same time during a semester was not a wise choice.] So, while initial "student-
conflict" may be inevitable when scheduling courses, it is absolutely imperative that
"student-conflict" not occur when scheduling final examinations, for a student should
not be scheduled to take exams for different courses at the same time.

The \textit{(class-teacher) timetabling problem} was first defined in 1963 by Gotlieb [33]
as follows:

Given a set of teachers

\[ T = \{t_i\}, \text{ where } i = 1, \ldots, \alpha, \]

a set of classes

\[ C = \{c_j\}, \text{ where } j = 1, \ldots, \beta, \]

a set of hours

\[ H = \{h_k\}, \text{ where } k = 1, \ldots, \sigma, \]

and an $\alpha \times \beta$ requirements matrix

\[ R = [r_{ij}] \]

where $r_{ij} \geq 0$ and $r_{ij}$ equals the number of hours teacher $t_i$ is to meet class $c_j$, create
a timetable that must satisfy these and other certain prescribed conditions. Such
conditions are unavailability constraints and preassigned meetings. \textit{Unavailability
constraints} are described by matrices $D$ and $E$. Here $D = [d_{ik}]$ is an $\alpha \times \sigma$ matrix with
$d_{ik} = 1$ if teacher $t_i$ is unavailable at hour $h_k$; otherwise $d_{ik} = 0$. Similarly, $E = [e_{jk}]$
is a $\beta \times \sigma$ matrix with $e_{jk} = 1$ if class $c_j$ is unavailable at hour $h_k$; otherwise $e_{jk} = 0$.
\textit{Preassigned meetings} are described by sets $P_{ij}$, where $i = 1, \ldots, \alpha$ and $j = 1, \ldots, \beta$. 
The sets $P_{ij}$ are defined by $P_{ij} = \{h_k \in H: \text{teacher } t_i \text{ is to meet class } c_j \text{ at hour } h_k\}$. $P_{ij} = \emptyset$ if there are no preassigned meetings involving both teacher $t_i$ and class $c_j$.

In 1967, Welsh and Powell [64] first showed the equivalence of timetabling problems with graph coloring problems, but were not able to solve such problems when preassigned meetings are considered. This equivalence result was strengthened in 1974 with the following theorem by Neufeld and Tartar [55]. It shows that Gotlieb’s timetabling problem with unavailability constraints and preassigned meetings is related to a graph with constraints on the colors to which specific nodes must or cannot be assigned, and that the existence of a proper coloring of this graph is both a necessary and sufficient condition for the existence of a solution to the timetabling problem.

**Theorem 1 (Neufeld, Tartar [55])** Corresponding to the timetable problem with unavailability constraints and preassigned meetings is a graph $H$. There exists a solution to the above timetable problem if, and only if, graph $H$ is $\sigma$-colorable.

The above theorem illustrates that a $\sigma$-coloring is equivalent to a timetable with $\sigma$ time slots.

### 3.2 A History of Graph Coloring Approaches to Timetabling

#### 3.2.1 Vertex-Coloring Approaches

Since the minimum coloring problem is formally NP-hard for general graphs [32], it therefore would seem unlikely, if not impossible, to find a fast (i.e., polynomial-time) algorithm to solve the minimum coloring problem. In fact, assuming that $P \neq NP$, there is an $\epsilon > 0$ such that no polynomial-time approximation algorithm for the minimum coloring problem can even find a solution that is guaranteed to be optimal
within a ratio of $|V|^c$, where $|V|$ denotes the number of vertices in the graph [62]. In 1981, Manvel suggested that the CPU time required to compute $\chi$ necessarily grows exponentially with the number of vertices, and that this property makes the minimum coloring problem intractable for graphs with over 100 vertices [47]. Even though the minimum coloring problem is intractable in the worst case, it is still necessary to be able to find solutions to graph coloring problems, for they may very well help to find good solutions to timetabling problems, as well as to a number of other interesting applications.

Graph coloring algorithms and heuristics continue to be developed with the goal of being able to properly color graphs more efficiently (i.e., using fewer colors), if not optimally (i.e., using the minimum number of colors necessary). The results of Welsh and Powell [64] in 1967 and Neufeld and Tartar [55] in 1972 alluded to in the previous section, illustrating the relationship between timetabling and graph coloring, have also led to the development of a number of new general graph coloring algorithms, with the hope that some of these algorithms would not only be able to solve (or approximately solve) the minimum coloring problem more efficiently, but also might be successful in coloring graphs that arise from timetabling problems, more specifically examination timetabling problems. The rest of this section will be devoted to highlighting some of the history of graph coloring approaches to solving timetabling problems.

A common approach used in approximation algorithms for graph coloring is called *successive augmentation*. In this technique, a partial coloring is first found on a small number of vertices and this coloring is iteratively extended vertex by vertex until the entire graph is colored. Examples relating to the successive augmentation approach to graph coloring include the work of Broder [10] in 1964, Peck and Williams [58] in 1966, Welsh and Powell [64] in 1967, Wood [69] in 1969, Matula, Marble and Isaacson [49] in 1972, Williams [67] in 1974, Carter [15] in 1978, Brelaz [6] in 1979, and Dutton and Brigham [26] in 1981. A brief synopsis of each is described below. As is evident from the next few paragraphs, many of these heuristic algorithms are
extremely similar (and in some cases, almost entirely identical) to each other. In fact, it appears that most authors were indeed unaware of the existence of other published material at the time of their work!

One of the earliest published examples of examination timetabling procedures involving graph coloring methods was presented by Broder in 1964, before the work of Welsh and Powell [64] and Neufeld and Tartar [55] was published. Broder's stated objective in examination timetabling was to minimize the number of student conflicts, rather than completely eliminate them. His algorithm is based on a relatively simplistic "largest degree first" coloring method. In the case of ties, each exam in the list to be scheduled is randomly assigned to one of the time periods that creates the fewest number of conflicts.

In 1966, Peck and Williams presented their examination scheduling algorithm, often described as "largest degree first: fill from top". As in Broder's method, vertices to be colored are sorted in decreasing order by degree. The algorithm scans the list of exams, placing as many exams as possible in the first time slot (lowest numbered color), then going back to the top of the list and filling the second time slot, and so on and so forth.

One year later, Welsh and Powell also gave a "largest degree first" coloring algorithm to accompany their graph coloring/timetabling equivalence result. The algorithm operates on the same basic rationale as that of previous authors employing this method, namely that the vertices with the most edges adjacent to them will be the hardest to color, if we wait until their neighbors have been colored.

Wood's graph algorithm operates on two $n \times n$ matrices, where $n$ denotes the number of vertices in the graph; a conflict matrix $C$ is used to illustrate which pairs of vertices must be colored differently due to constraint restrictions in the problem, and a similarity matrix $S$ is used to determine which pairs of vertices should be colored the same.

Matula, Marble and Isaacson's coloring algorithm is a "smallest degree, last recur-
sive” coloring algorithm. This rationale is similar to “largest first” methods in that vertices of lowest degree are easy to color. The algorithm includes an interchange feature which involves looking for a color swap in vertices adjacent to the one which is currently trying to be colored when the normal method would introduce a new color, thereby adding a limited search ability to the algorithm.

Williams, in defending a “largest modified degree first” graph coloring algorithm, proposed that the degree of a vertex was important but not entirely sufficient in determining the difficulty of scheduling its corresponding exam. He conjectured that a course was critical in the scheduling process if a large number of its neighbors were also critical and suggested a formula, based on the degree of a particular vertex and its neighbors, to quantify the critical nature of that vertex. Naturally, vertices that were determined to be more critical than others were colored first.

As its name suggests, Carter’s coloring algorithm “largest degree first recursive: fill from top” is very similar to Peck and Williams’ “largest degree first: fill from top”, with one addition. As each vertex is colored and removed from the list, the degrees of the vertices in the remaining subgraph are recalculated and the list resorted.

Two of the most popular heuristic graph coloring algorithms were introduced by Brelaz in 1979 and Dutton and Bingham in 1981. In Brelaz’ algorithm, titled “largest saturation degree”, the vertex adjacent to the most already assigned vertices is chosen to be the next to color at each step. Dutton and Bingham’s algorithm employs a slightly different heuristic. Taking each color in turn, the two vertices with the most common adjacent vertices are merged continually until a clique is formed. Once this is complete, all the vertices merged into the same one are colored identically.

An examination timetabling survey by Carter [16] in 1986 shows that the graph theoretic approach to timetabling is indeed the most popular, and references several graph coloring algorithms and heuristics, including those summarized above, and ways in which they have been applied to solving problems in examination timetabling problems at particular institutions, or are otherwise relevant to solving problems of
this nature. Notable among these algorithms is one described by Mehta [50] in 1981 that was implemented on a examination scheduling problem involving 750 students, 84 examinations, and 12 time periods at Cedar Crest College in Allentown, Pennsylvania. Mehta’s algorithm combines the “largest saturation degree first” algorithm of Brelaz with a complex “compression” routine that further attempts to reduce the number of colors used. According to Carter in his survey, Mehta’s work is significant in that it appears to be the most complex of the examination timetabling applications with respect to the objective of obtaining a “conflict-free” examination timetable, given a fixed number of time periods [16].

Hertz and de Werra [34] introduced a probabilistic search method involving a Tabu search coloring technique similar to simulated annealing in 1987 that could be applied to solving timetabling problems. It begins by consecutively finding color classes as in an Almost Maximal Independent Set (AMIS) algorithm. When there are only a certain number of unassigned vertices left, the algorithm colors the remaining vertices, successively swapping ones that are adjacent to vertices of the same color. Another example of a probabilistic search method for graph coloring applicable to timetabling is Ellis and Le polesa’s [28] Las Vegas coloring algorithm from 1989, which provides an estimate for the chromatic number of a graph based on the size of smallest clique amongst the leaves of a Zykov tree. A Zykov tree is a tree generated by identifying or adding an edge between two vertices, so that one of the vertices lies in in a focus clique and the other has the most neighbors in the clique.

In 1991, Johnson, Aragon, McGeoch and Schevon [42] implemented and tested three different approaches for graph coloring with a simulated annealing technique, observing that simulated annealing algorithms can achieve good results, but only if allowed a sufficiently large run time. They also believed that no particular graph coloring heuristic is necessarily “best”, but most if not any of them can perform well given the right type of graph; that is to say, certain graph coloring heuristics perform better on certain types of graphs.
A 1992 paper by Kiaer and Yellen [43] describes a heuristic algorithm using a graph coloring approach to find approximate solutions for a university course timetabling problem. A weighted graph is used to model the problem of scheduling university courses while minimizing conflict. The objective is to find a least-cost \( k \)-coloring of the graph, where \( k \) is the number of available time slots. Nonnegative weights on the edges of the graph are introduced to represent the severity of conflicts between pairs of courses. A large weight on an edge indicates a strong preference for not scheduling in the same time slot the courses associated with the endpoints of that edge. Also associated with each vertex in the graph is a cost vector having \( k \) nonnegative components. The \( i \)th component represents the cost incurred when that vertex (i.e., course) is assigned the \( i \)th time slot (i.e., colored with the \( i \)th color). The magnitude of each component of the cost vector is initially set to reflect the desirability of scheduling that course in the associated time slot. The larger the number, the less desirable that time slot. The cost vector of a vertex is updated as adjacent vertices are colored. At each stage in the coloring process, the cost vector components reflect the current cost of coloring the vertex, and provide a heuristic measure of the impact of scheduling that course in a particular time slot. An additional feature of the graph model, corresponding to the limitation on the number of classrooms in which the courses can be held, is a \( k \)-component classroom vector. The \( i \)th component of the classroom vector represents the number of available classrooms for each time period. When coloring the graph, vertices are colored one at a time, and in general the "most difficult" vertices are colored as early as possible, according to a complex vertex selection criteria based on the values of the cost vector components and the edge weights. Each time a vertex in the graph is colored, the cost vectors of the adjacent uncolored vertices, as well as the classroom vector, are updated. The heuristic algorithm was implemented and tested on a number of rather graphs of small order (i.e., 50 to 100 vertices), including one small problem involving the scheduling of 64 actual courses at Florida Institute of Technology (FIT) into six time slots. Results compared favorably to those obtained
by the FIT administration's usual manual method of course scheduling in terms of
the number and severity of conflicts resolved. The algorithm does not account for a
course's preference for a particular time slot or slots, nor does it allow for overlapping
time periods or varying classroom sizes and/or types.

In 1993, Burke, Elliman and Weare [13] introduced plans for a university timetabling
system based on graph coloring and constraint manipulation. Graph coloring and
room allocation heuristic algorithms were described along with an illustration of how
the two can be combined to provide the basis of a system for timetabling. The au-
thors also discussed the handling of several common timetabling features within the
system, primarily with regards to examination timetabling. Some particular features
are specifically exclusive to examination timetabling (e.g., the restriction that two
exams must or must not occur in consecutive time slots, or that larger exams should
come near the beginning of the exam period, as they take longer to grade), while
other features could be translated and applied to a system for course timetabling
(e.g., the restriction that an exam must or must not occur in a particular time slot).

Algorithms for finding optimal colorings of graphs (as opposed to approximation
methods involving successive augmentation or probabilistic searches) have for the
most part been based on implicit enumeration. In 1972, Brown [11] described a basic
enumeration technique known as backtrack programming, along with a flow diagram
of the algorithm. An improvement of this algorithm incorporated the use of a "look-
ahead" procedure to reduce the number of backtracks required. Both the basic and
the look-ahead algorithms attempt to compute an optimal coloring of a graph. As a
test instance, a 112-vertex subgraph with over 440 edges was obtained from a graph
modeling the problem of scheduling final examinations at the Massachusetts Institute
of Technology. Brown's basic algorithm correctly computed the chromatic number
of 15 for this particular subgraph, with 162 backtracks and a solution time of 0.4
seconds.

Before proceeding, it is again interesting to note and worth restating that a large
number of the graph coloring algorithms and heuristics for timetabling described here, and appearing in the literature as a whole for that matter, have heretofore been predominantly used in studying and solving examination timetabling problems over course timetabling problems. There are, of course, several exceptions to this fact, and these cases are primarily explicitly noted above and throughout this dissertation as specific research relevant to either both course and final examination timetabling, or just simply to course timetabling. Again, course timetabling is related but generally not identical to the problem of examination timetabling. For example, in timetabling courses, time periods in which courses are to be scheduled are often of varying lengths and may be allowed to overlap, issues generally not present in scheduling examinations. Scheduling conditions and preferences desired in creating course timetables may differ from those conditions and preferences desired in creating final examination timetables.

3.2.2 Edge-Coloring Approaches

So far, we have seen ways in which timetabling problems can be modeled as graph coloring problems, in particular vertex-coloring problems. However, given a simpler version of Gotlieb’s original timetabling problem without the complications of unavailability constraints and preassigned meetings, one can model and solve this problem as an edge-coloring problem.

Consider the simpler version of the timetabling problem as it appears in *Graph Theory with Applications* by Bondy and Murty [5]: In a school, there are m teachers $X_1, X_2, \ldots, X_m$ and n classes $Y_1, Y_2, \ldots, Y_n$. Given that teacher $X_i$ is required to teach class $Y_j$ for $p_{ij}$ periods, schedule a complete timetable in the minimum number of possible periods.

To model the above timetabling problem as an edge coloring problem, the teaching requirements are represented first by an $m \times n$ matrix $P = [p_{ij}]$, where the entry $p_{ij}$ is the number of periods teacher $X_i$ is required to teach class $Y_j$. The ma-
trix $\mathcal{P}$ is then transformed into a bipartite graph $G$ with bipartition $(X, Y)$, where $X = \{X_1, X_2, \ldots, X_m\}, Y = \{Y_1, Y_2, \ldots, Y_n\}$, and vertices $X_i$ and $Y_j$ are joined by $p_{ij}$ edges. It is assumed that in any one period, each teacher can teach at most one class, and each class can be taught by at most one teacher. Therefore, a teaching schedule for one period corresponds to a matching in the graph and, conversely, each matching corresponds to a possible assignment of teachers to classes for one period. The problem is thus reduced to partitioning the edges of $G$ into as few matchings as possible or, equivalently, to properly color the edges of $G$ with as few colors as possible. Figures 3.2 and 3.3 below illustrate a solution to a 4-teacher, 5-class example where $X = \{X_1, X_2, X_3, X_4\}, Y = \{Y_1, Y_2, Y_3, Y_4, Y_5\}$, and

$$
\mathcal{P} = \begin{pmatrix}
2 & 0 & 1 & 1 & 0 \\
0 & 1 & 0 & 1 & 0 \\
0 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 1 & 1 \\
\end{pmatrix}
$$

The edges of the graph $G$ in Figure 3.2 are partitioned into four matchings: $M_1, M_2, M_3$, and $M_4$. Associated with these four matchings are four time periods: 1, 2, 3, and 4. Each matching $M_i$ corresponds to an assignment of teachers to classes for Period $i$, as shown in Figure 3.3. Note that it may often be the case that a particular teacher or teachers will not be assigned to teach a class during a particular period. For example, the timetable in Figure 3.3 does not assign a class to teacher $X_2$ in Period 2. From the edge-coloring of $G$ and its associated timetable we conclude that four periods are both necessary and sufficient to schedule a conflict-free timetable for the above example.

The following theorem is attributed to König, where $\Delta(G)$ denotes the maximum degree of $G$:

**Theorem 2 (König)** If $G$ is bipartite, then $\chi'(G) = \Delta(G)$. 
Figure 3.2: A 4-edge-coloring (4-teacher, 5-class timetabling example)

<table>
<thead>
<tr>
<th></th>
<th>Period 1</th>
<th>Period 2</th>
<th>Period 3</th>
<th>Period 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X_1$</td>
<td>$Y_1$</td>
<td>$Y_1$</td>
<td>$Y_3$</td>
<td>$Y_4$</td>
</tr>
<tr>
<td>$X_2$</td>
<td>$Y_2$</td>
<td>—</td>
<td>$Y_4$</td>
<td>—</td>
</tr>
<tr>
<td>$X_3$</td>
<td>$Y_3$</td>
<td>$Y_4$</td>
<td>—</td>
<td>$Y_2$</td>
</tr>
<tr>
<td>$X_4$</td>
<td>$Y_4$</td>
<td>$Y_5$</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>

Figure 3.3: A 4-period timetable (4-teacher, 5-class timetabling example)

Therefore, if no teacher teaches for more than $p$ periods, and if no class is taught for more than $p$ periods, the teaching requirements can be scheduled in a $p$-period timetable. Furthermore, we have a polynomial-time algorithm for constructing such a timetable. If $l$ total lectures are to be given, and they have been scheduled in a $p$-period timetable, then it is easy to see that at least $\lceil l/p \rceil$ rooms will be needed in some particular period, since the timetable requires an average of $l/p$ lectures to be given per period. It turns out that one can always arrange $l$ lectures in a $p$-period timetable so that at most $\lceil l/p \rceil$ rooms are occupied in any one period. We thus have a polynomial-time solution to this simplified timetabling problem [5].
3.3 Alternative Approaches to Solving Timetabling Problems

3.3.1 Mathematical Programming

While timetabling problems have a direct correlation to graph coloring, several direct integer programming formulations of timetabling have also been developed in an attempt to solve these problems.

In 1969, Lawrie [45] was one of the first persons to model the timetabling problem as an integer program (more specifically, an integer linear program) with linear constraints and no objective function. Finding a solution to an instance of the timetabling problem therefore amounts to finding a feasible solution to the corresponding integer linear programming problem (i.e., one satisfying all of the problem's constraint functions).

Approximately 20 years later, Aubin and Ferland [4] attempted to solve a large timetabling problem comprised of two subproblems: timetabling and grouping. In the timetabling subproblem, a master timetable is derived taking into account student registrations and lecturer and classroom availabilities. The grouping subproblem specifies groups of students for large courses that have to be repeated several times during the week. The entire problem can be modeled as an integer programming problem that contains two sets of decision variables (one set for each of the two subproblems) and one objective cost function which is dependent on both sets of variables. Obtaining an optimal solution to this integer programming problem amounts to minimizing the objective function subject to four sets of constraints, two sets of which essentially specify a starting time for each lecture (timetabling), and two sets of which are responsible for assigning students to course sections (grouping). A solution procedure for the entire large-scale timetabling problem is presented by Aubin and Ferland which iteratively improves the objective function value by working successively on the timetabling and the grouping subproblems until a stable solution
(i.e., a solution that cannot further be improved by the procedure) is reached. In this approach each subproblem is solved successively while keeping fixed the value of the decision variables of the other subproblem. The overall procedure is therefore viewed as iteratively reducing the number of conflicts by successively modifying the master timetable and the grouping of students. The procedure was implemented in PASCAL on the CYBER 835 of the Computing Center of the University of Montreal, and tested using data from a large high school in Montreal (3300 students, 250 courses, 825 course sections, 85 available classrooms, and over 200 lecturers). Solution quality was measured with respect to the number of resulting conflicts and distribution of students among course sections, as well as shortage (if any) of classrooms. The procedure's efficiency was based on CPU time of each algorithm (timetabling and grouping) and the number of reassignments necessary to reduce the objective function value. The paper by Aubin and Ferland provides a detailed version of both the timetabling and grouping algorithms as well as tables of computational results of further tests on which the entire procedure was performed.

Brunetta and DePoli [12], in a 1997 article referring to the problem of course timetabling, stated that "because of the recent advances in hardware and software, integer programming is perfectly capable of providing solutions for real applications". They developed a decision support system based on an integer linear programming approach and graphical user interface that allows a user to build a course timetable using an Internet browser. The integer linear programming model is implemented in C code, with the CPLEX 3.0 callable library acting as the integer linear program solver. A number of real world instances were solved with low computational effort using data from the University of Padova in Italy. Data sets ranged in size from 31 to 39 courses to be scheduled.

In 2000, Abdennadher and Marte [2] timetabled the courses offered at the Computer Science Department of the University of Munich, by using an approach involving Constraint Logic Programming (CLP). CLP is a mathematical programming

Also in 2000, Reis and Oliviera [59] identified eight classes of interrelated timetabling subproblems, and introduced UniLang, a new language for representing timetabling problems that can easily be read by computer specialists and school administrators. They illustrated how a timetabling problem represented in UniLang can be translated to a constraint logic program and solved using a constraint logic programming language.

3.3.2 Some Other Approaches

Desroches, Laporte, and Rousseau [25] developed HOREX in 1978, a computer program that follows a complex series of steps to construct a complete examination timetable. The name HOREX is derived from the French word horaire, meaning timetable. The developers of HOREX did not give explicit details regarding their methods, but rather described six general stages of the algorithm. Their approach follows a considerable departure from the more traditional methods of solving timetabling problems.

In step 1 of HOREX, the algorithm finds an initial solution, consisting of $p$ sets of non-conflicting exams using the graph coloring method of Matula, Marble, and Isaacson [49] described above. Here $p$ denotes the number of exam periods allowed. Step 2 then uses a minimum matching algorithm to combine these $p$ sets into pairs corresponding to a "day" (i.e., morning/afternoon) having the minimum number of "doubles" (i.e., students who would be required to take two exams in one day). The problem is solved using a branch and bound method for integer linear programming as implemented by Land and Powell [44] in 1973. Step 3 involves basic moves of one exam at a time in attempts to further reduce the number of "doubles". In step 4, exams are moved within each pair to maximize the number of morning exams. The
implementation of Land and Powell [44] is once again employed to solve this knapsack problem. In step 4, the days are ordered using a traveling salesman problem (TSP) heuristic developed by Miliotis [51] in 1976 to minimize the number of occurrences in which a student must take exams on consecutive days. To analogize the TSP heuristic, each "city" corresponds to one day, and "distance" is defined as the number of consecutive exam occurrences between each pair of days. Finally, in step 6, the minimum "tour", or cycled ordering of days, is rotated through each possible starting day to determine the minimum total number of such consecutive exam occurrences, ignoring those taking place over weekends and holidays.

HOREX was implemented at L'École Polytechnique de Montréal with typical problems consisting of 160 examinations to be scheduled over 12 days, with two exam time slots per day. While no explicit computational results were given, the algorithm developers offered some key observations. First, the initial graph coloring solution requires very little time, since the procedure virtually ignores all secondary (i.e., preferential) constraints. Consequently, the initial solution is very poor from the perspective of satisfying secondary constraints. Next, the steps taken to improve the initial solution are relatively expensive. Finally, the entire algorithm becomes impractical when more then a few exams are preassigned to fixed time periods.

An algorithm with an approach very similar to HOREX was published by White and Chan [66] in 1979. A typical run of this algorithm could schedule approximately 390 exams over 25 time periods for 16,000 students in roughly 40 minutes on an IBM 360/65 machine. The observations listed above that were offered by the developers of HOREX apply equally to White and Chan's algorithm. However, White and Chan claimed that in their approach preassigned exams could be accommodated.

In 1991, Mathiasiel and Comm [48] devised an interactive graphical approach to timetabling problems. Their program allowed human intuition to attempt to solve the problem, while relying on a database system to store the data resources, time period assignments and other information, as well as to perform a variety of operations on
the timetable.

A study by Elmohamed, Coddington, and Fox [29] in 1997 investigated a number of optimization methods for academic course scheduling based on simulated annealing. These techniques include mean-field annealing, simulated annealing with geometric cooling, simulated annealing with adaptive cooling, simulated annealing with adaptive cooling and reheating as a function of cost, and simulated annealing using each of the three different cooling schedules above, along with a rule-based preprocessor to provide a good initial solution to the problem. Of these methods, the best results were obtained using simulated annealing with adaptive cooling and reheating as a function of cost, and with a rule-based preprocessor to provide a good initial solution. Using this last approach, Elmohamed, Coddington, and Fox were able to generate valid timetables using data from Syracuse University.

A number of evolutionary and genetic algorithms have also been developed and used in attempting to solve university course and/or final examination timetabling problems. Notable work include that of Burke, Elliman and Weare [14] in 1995, Erben and Keppler [30] in 1995, Paechter, Rankin, Cumming, and Fogarty [57] in 1998, and Woods and Trenaman [70] in 1999. Some of these methods also explore the possible satisfaction of a variety of "hard" and "soft" constraints that can define problems in timetabling.

The new graph-coloring-based university timetabling model and variations we develop and present later in Chapter 5 of this dissertation will indeed incorporate the satisfaction of a number of these as well as other essential (i.e., "hard") and preferential (i.e., "soft") timetabling conditions.
Chapter 4

University Timetabling Methods Currently in Practice

4.1 Rice University (Houston, TX)

Rice University is a nationally recognized and well-renowned private research institution in Houston, Texas with a current enrollment of approximately 2,700 undergraduate and 1,850 graduate students, employing over 550 faculty members.

Each semester, roughly 1,600 to 1,750 courses (not including multiple sections of courses) are available to be offered by the university. First-year and sophomore course offerings remain rather constant from semester to semester, with most course variations occurring among the junior, senior, and graduate level courses. Each member of the Rice faculty is responsible for submitting a request form to the Registrar indicating what courses he or she is intending to teach that semester, along with optional time and room preferences for these courses. Often a particular academic department will ask its individual faculty members to first submit their course requests directly to the department chair for initial approval. The department chair will then compile the course requests and submit them to the Registrar. Most classes at Rice are taught during the day, though few classes do occur during evening hours. Classes generally
meet within predetermined time slots either once, twice or three times a week. There is usually no enrollment limit to a particular course, although multiple sections of individual courses may be allowed. After all of the requests are received, they are entered into the huge storage database EXETER, which also stores Rice University student, faculty and staff information (e.g., students’ grades, faculty positions, etc.). From EXETER the request data is then exported into the scheduling program Ad Astra, which organizes the data and outputs the conflicts that would arise if all requests were honored. Given these initial conflicts, the Registrar’s Office reschedules the courses (this is usually by hand) and then schedules any previously unscheduled courses (e.g., new courses or seminars) and makes any last-minute changes before the final timetable is imported back into EXETER and becomes available for posting on the Internet or in newspaper form to be distributed to the student body prior to registration. From the initial faculty request submissions to the final printing of the semester course schedule, this entire timetabling procedure takes approximately six to seven months. Even after this lengthy process has been completed, however, further scheduling conflicts will often arise and subsequent scheduling modifications will often necessarily take place after students have registered for classes for the upcoming semester, and also during the first few weeks of the semester once classes have begun. Normally, however, these modifications are said to be relatively minor and few in number, and the Registrar’s office is usually eventually able to honor a large number of faculty course requests on average in a given semester.

With regards to final examination scheduling, Rice University offers two categories of exams: scheduled exams and self-scheduled exams. A professor may alternatively choose not to give a final exam, or to give a take-home final exam to his or her class; such an exam is usually handed out by the professor before the beginning of the exam period and due on or before the last day of the exam period, approximately two weeks later. Such take-home final exams are not scheduled by the Registrar. All other courses with enrollment of 50 or more students will be assigned a three-hour
final exam time slot and room during the two-week final exam period. Such exams are relatively few in number at Rice and are usually scheduled by hand. For courses with enrollment of fewer than 50 students, self-scheduled exams are offered. Students in these classes preselect, usually during the last few weeks of classes, the days and times they wish to take their final exams, given a set of available three-hour time slots. For each day of the two-week final exam period, two three-hour time slots are available for taking final exams. In the Spring semester, students graduating that May must take all of their final exams during the first week of the two-week exam period. With self-scheduled exams, it is often the case that a particular exam room will contain a group of students each taking a different final exam at the same time!

4.2 The University of Houston (Houston, TX)

The University of Houston is a public, urban research university offering a wide array of academic and professional degree programs. It is the largest member of the University of Houston System, which includes campuses in nearby Clear Lake, Fort Bend, Victoria, The Woodlands and Cinco Ranch, as well as in downtown Houston, Texas. Approximately 32,000 students attend the University of Houston, which employs over 850 faculty members.

The term "class schedule development" at the University of Houston refers to the creation or building of a class schedule for a semester and incorporates all activities such as 1) the creation of an initial schedule, 2) the deletion or addition of courses or sections, 3) any changes or adjustments to specific section information such as instructor, time and days, room and building, and 4) any changes to class quota in response to demand or departmental concerns. The process is initiated twice a year, usually in July for the following Spring semester, and again in December for the next Summer/Fall semester. The Department of Registration and Academic Records first compiles an initial schedule based on the prior same semester (Spring or Sum-
mer/Fall) which includes all courses with enrollment in that semester that remain active in the course catalog inventory. Exceptions to this rule include courses of independent study, private lesson, thesis, dissertation, and student teaching practicum, which automatically roll over to the next semester's schedule regardless of enrollment. Departments and colleges may then make adjustments to this schedule based on personal requests. Approximately 120 faculty advisors and department chairs have access to the large VAS ADABAS: RARCAS interactive database which allows for these adjustments by the departments and colleges themselves, as long as conflicts are avoided. Each semester, over 16,000 courses remain active in the course catalog, approximately 12,000 of which will be offered during a particular semester.

The University of Houston requires an even distribution of courses across 24 specified and recommended “time bands”, during which classes meet for either one hour three times a week, one and a half hours twice a week, or three hours once a week. Most classes occur within these time bands, although it is not required by the university that they do so. Preferential treatment with regards to room assignments is given to classes that are scheduled within the recommended time bands, however. When scheduling final examinations, courses with classes that meet during the same particular time band are scheduled for the same examination time slot in order to eliminate conflicts. Professors of classes that do not meet during the time bands are responsible for scheduling their own final examinations, keeping in mind that conflicts in the final exam schedule must ultimately be avoided.

4.3 The University of St. Thomas (Houston, TX)

The University of St. Thomas is a private Catholic university in Houston, Texas offering undergraduate and graduate programs characterized by excellence in teaching, research and scholarship. Approximately 1,900 undergraduate students and 1,100 graduate students are enrolled as campus-based students. An additional 2,000 or
so students are enrolled in online or distance courses affiliated with the university. The University of St. Thomas is served by 117 full-time faculty, 74 of whom are tenured, and by 153 part-time members. The average class size is 18 students and the student/faculty ratio is approximately 14 to 1. The university includes a School of Education, a School of Business, a School of Masters in Liberal Arts, and a School of Arts and Sciences. Each school has one university-appointed Dean. The School of Education and School of Masters in Liberal Arts each comprise a single academic department at the university, while the School of Business and the School of Arts and Sciences are comprised of roughly five and twelve individual academic departments, respectively. Each academic department is headed by an appointed Department Chair.

Each semester, roughly 600 to 700 courses at the University of St. Thomas are scheduled by the Registrar's Office. Most of these courses are lecture and/or laboratory courses, and generally do not include independent study or research courses, which are usually scheduled separately by the individual instructor or advisor and student. The Assistant Registrar is primarily responsible for scheduling the lecture and laboratory courses for the Winter semester as well as for the Summer and the Fall semester. The entire scheduling process occurs twice each year, once for the Spring semester and once for the Summer/Fall semesters, and is identical in both occurrences. A detailed description of the general procedure appears below, accompanied by specific dates and deadlines for one such scheduling process, namely that of the scheduling of the Summer/Fall 2003 semesters.

The scheduling process for a given semester begins one week before the first day of classes of the previous semester during "Faculty Study Day". For example, Faculty Study Day for the Summer/Fall 2003 semester scheduling process occurred on January 6, 2003. Faculty Study Day marks the Assistant Registrar's first contact with the individual department chairs at the university with regards to scheduling courses. On that day, the Assistant Registrar sends a "template" (in most cases one, in some
cases more than one) to each department chair, which contains a sample of that department’s schedule based on historical information from a previous semester. For reference, historical data for the entire university is also included on each department template. Each department chair is responsible for scheduling times for the courses within his or her own department. This is done by modifying the existing department template, where necessary, to account for new time requests for existing courses, as well as the addition or deletion of courses from the next semester’s schedule. Any other new and pertinent information or special instructor requests (e.g., a change in instructor for a particular course, or a request for a particular classroom for a given course), are also noted by the department chair on the modified template(s).

Once the Assistant Registrar has sent the scheduling templates to the department chairs, the department chairs have approximately two and a half weeks to complete them and send them to the dean of their respective schools for approval. The deans then have approximately one and a half weeks to approve the completed department schedule templates, making additional changes where necessary, and return them to the Assistant Registrar. For scheduling the Fall/Summer 2003 semesters, the dates for these two deadlines were January 22, 2003 and February 5, 2003, respectively. After receiving all of the dean-approved department schedule templates, the Assistant Registrar types the information into a PeopleSoft database, a process which usually takes about two to three days to complete. Once this has been done, the data from the PeopleSoft database is then imported into the scheduling program Ad Astra, which reorganizes the data and outputs any outstanding scheduling conflicts. Such conflicts are then resolved by the Assistant Registrar manually, one at a time. Changes to a course’s time slot cannot be performed via Ad Astra, but rather only within the PeopleSoft database. In order to change the time for a particular course, the Assistant Registrar must first export the entire data set from Ad Astra to PeopleSoft, make the necessary change, and then reimport the entire data set back into Ad Astra.

After all conflicts have been resolved, and a timetable of courses has been created,
the Assistant Registrar manually assigns a classroom to each course based on room availability and capacity. For many courses, classrooms are fixed depending on the type of room needed. The Assistant Registrar first assigns classrooms to the science courses, since those courses often have the most specific requests for room type (i.e., laboratory). Classrooms are then assigned for other courses based on equipment needs and other requests. During this process, individual room assignments are entered in Ad Astra, which provides a list of available classrooms to which each course can be properly assigned without conflicts.

Once this process has been completed, the Assistant Registrar sends the course timetable and room assignment to the deans and department chairs for a final proofing, which usually takes approximately two to three days. Once the proofing is finished, the information will appear on the university's web pages as well as be sent to the Course Bulletin printer for printing. For scheduling the Fall/Summer 2003 semesters, the printer submission date was February 27, 2003. A hard copy of the Course Bulletin is then available in two to three weeks, just prior to the preregistration period for continuing students, which occurs roughly two and half months after the beginning of the entire semester course scheduling procedure on Faculty Study Day. Even after the schedule has been completed, however, further scheduling requests to change either course times and/or classrooms will arise and subsequent scheduling modifications will often necessarily take place after students have preregistered for classes for the upcoming semester, and also during the semester once classes have begun. All such requests are dealt with individually by the Assistant Registrar on a first-come, first-served basis.

With regards to final examination scheduling, the University of St. Thomas uses a semester’s course timetable to directly schedule its final exams. All courses scheduled for the same time slot during the semester (e.g., MWF 10:10-11:00AM) are scheduled for the same fixed-length final exam time period (e.g., Wed. May 7, 8:30-11:00AM). Furthermore, all final exams are held in the same classroom in which the course is
held during the semester. This is done in order to avoid time and room conflicts for students taking final exams. The final exam schedule usually lasts anywhere from seven to nine consecutive days with four or five final exam time periods per day.
Chapter 5

A New Graph Coloring Model for University Timetabling

5.1 An Overview of the Model

The important task of scheduling courses and final examinations at a college or university is an example of timetabling. Timetabling is the scheduling of a set of related events in a minimal block of time such that no resource is required simultaneously by more than one event. In university timetabling, the resources involved, which we assume may be required by no more than one course at any particular time, are instructors, classrooms, and students. As mentioned earlier in this dissertation, timetabling (in particular, university timetabling) is a practical application of graph coloring. The minimum coloring problem and the timetabling problem have been classified as NP-hard problems in the general case. Simply put, this means that it is unlikely that it will be possible to find fast (i.e., polynomial-time) algorithms to solve these problems. In order to find optimal solutions to such NP-hard problems, it is usually necessary to consider all possible solutions to choose the best one. However, solutions to large-scale practical problems, including the timetabling problem, are often desired and needed much more quickly than any exhaustive search algorithm.
could hope to provide. As a result, numerous heuristics have been developed for such problems which produce "near-optimal" satisfactory solutions in much less time.

In this chapter we develop and present a mathematical and computational model with variations for solving university timetabling problems using techniques and heuristics of graph coloring that incorporates the satisfaction of both essential and preferential timetabling conditions. The model involves creating a conflict graph from the assembled input university course data, properly coloring the conflict graph, and transforming this coloring into a conflict-free timetable of courses. From this conflict-free course timetable one can then assign the courses to classrooms based on room capacity and availability. Finally, once a course timetable is constructed, a conflict-free schedule of final examinations for the courses can quite easily be obtained.

At colleges and universities it is most often the case that courses are scheduled for a particular semester prior to when students register and select their courses. When attempting to model and solve university course timetabling problems, it is therefore logical to assume that a particular set of courses has been designated to be offered during a semester by particular instructors, and that these courses are to be assigned appropriate (i.e., non-conflicting) time slots (and often, appropriate classrooms as well) prior to students' registration. Only after this scheduling has taken place would students then be allowed to register for courses whose times do not conflict. An alternate approach to university course timetabling not addressed in this dissertation that might be of worthy consideration for future research would be to first decide which courses would be offered by which professors, then allow students to register for their courses, and then assign times and classrooms to each course so that the number of "student conflicts" (i.e., the number of students who have course conflicts) is minimized.

University course timetabling problems involve pairwise restrictions on the courses being scheduled; that is, there exist restrictions on which courses can be scheduled simultaneously. Restrictions involved in creating a timetable of courses may be divided
into two categories, which we call essential and preferential timetabling conditions.

Essential timetabling conditions (also commonly referred to as hard constraints) are conditions or constraints that must be satisfied in order to produce a legal or feasible timetable. Examples of essential timetabling conditions include the following:

- Two or more courses taught by the same instructor cannot be scheduled for the same time slot [62, among others].

- Two or more courses that require the same classroom cannot be scheduled for the same time slot [62, among others].

- Equivalently, two or more courses scheduled for the same time slot cannot be assigned the same classroom [52, among others].

- Instructors must be available at the times their courses are scheduled [33].

- Each course must be scheduled for exactly one time slot and one room, to remain constant throughout the scheduling period [30].

- Some courses are required to meet a certain fixed number of times per week (for example, 3 times per week, 2 times per week, or 1 time per week) [52].

- Each course must be scheduled in an available classroom that can accommodate its size [13, among others].

- Specific outlined room requirements/type for a particular course, if any (for example, a laboratory workspace, or a computer workstation with projection screen) should be taken into account [52, among others].

- Any courses marked as “prescheduled” should be scheduled to the specified time [30, among others].

- Courses which enroll the same set of students (for example, a chemistry lecture course and its accompanying laboratory course) cannot be scheduled for the same time slot [52].
Preferential timetabling conditions (also commonly referred to as soft constraints) are additional conditions or constraints that need not necessarily be satisfied to produce a legal or legitimate timetable, but if satisfied may very well produce a more “acceptable” timetable for students and/or faculty members. These conditions are requests that should be fulfilled, if possible. Preferential timetabling conditions are often reasonable, but it may or may not be possible to fulfill each of them in addition to all of the essential conditions. Examples of preferential timetabling conditions include the following:

- An instructor may have a time preference as to when a course is scheduled to meet, either general (for example, in the morning, in the afternoon, or in the evening) or specific (for example, at 8:00 a.m.).

- An instructor may have a specific room request for a course, beyond the scope of the outlined room requirements specified above (this is an example illustrating the sometimes “fine line” between an essential and preferential timetabling condition).

- An instructor teaching a course that meets 1 time per week may have a preference as to which day to meet during the week.

- It might be preferable to assign each course to a classroom that is located in or close to the building in which that course’s department or school is based, and/or close to the office of the course’s instructor [30, among others].

- Most courses should not be scheduled in the evening, but rather during the morning and afternoon hours (i.e., during “normal business hours”), unless an evening time slot is specifically requested for a particular course [13, among others].

- Classrooms should be just large enough to hold the courses in them, in order to eliminate the presence of unused empty space [30].
• Minimize the number of classrooms used or needed when scheduling the courses [13].

• Try to schedule sufficiently large courses at times that “minimize pain for students” [70].

• Attempt for an even distribution of courses among time slots; this will aid in room assignment and also allow for maximum utilization of resources [17, among others].

In addition to the above essential and preferential timetabling conditions, we also introduce a few timetabling assumptions in our model:

• Many university courses meeting 3 times per week are usually scheduled to meet on Mondays, Wednesdays and Fridays for periods of 1 hour each (or 50 minutes, allowing 10 minutes of travel time for students between classes), for a total of 3 hours per week. In our model, courses meeting 3 times per week will be scheduled in this manner. For example, a “time slot” for a 3-time-per-week course might be “Monday/Wednesday/Friday, 9:00–10:00 a.m.”.

• Many university courses meeting 2 times per week are usually scheduled to meet on Tuesdays and Thursdays for periods of 1½ hours each (or 80 minutes, allowing 10 minutes of travel time for students between classes), for a total of 3 hours per week. In our model, courses meeting 2 times per week will be scheduled in this manner. For example, a “time slot” for a 2-time-per-week course might be “Tuesday/Thursday, 9:30–11:00 a.m.”.

• Many university courses meeting 1 time per week are usually scheduled to meet for a period of 3 hours (or 170 minutes, allowing 10 minutes of travel time for students between classes) on either Mondays, Tuesdays, Wednesdays, Thursdays, or Fridays (i.e., there are no Saturday or Sunday courses). In our model,
courses meeting 1 time per week will be scheduled in this manner. For example, a “time slot” for a 1-time-per-week course might be “Monday, 1:00-4:00 p.m.”.

Other assumptions present in our timetabling-by-graph-coloring model will be introduced and described in subsequent sections of Chapter 5.

5.2 Assembling University Course Data

Before the timetabling of university courses for a particular semester can take place, we must first assemble a collection of university course data for that semester, to serve as input to our problem instance.

As mentioned in Section 5.1, we assume in our timetabling model that we have a set of courses that are designated to be offered during a particular semester by particular instructors, and that these courses are to be assigned appropriate (i.e., non-conflicting) time slots, subject to a number of essential and preferential timetabling conditions. This assignment of courses to time slots (and often, to appropriate classrooms as well) takes place before (sometimes weeks or months before) students choose their courses for that semester.

As such, each course to be scheduled constitutes a data entry, containing the required or optional information below. Such information is expected to be supplied by either the instructor of the course, the academic department or school to which the course belongs, or some other appropriate source.

Required Data Fields:

- COURSE_ID: The COURSE_ID field contains the “name” of the course, identified by academic department and course number (for example, MATH101). If a course has multiple sections, each section will be entered as a different course. For example, if MATH101 has three sections, they will be entered as three
courses: MATH101.1, MATH101.2, and MATH101.3. Each course will have exactly one COURSE.ID. A course that is cross-listed as belonging to more than one academic department will appear only once in the schedule. For example, if ENGL101 and HUMA101 are the same course, then either ENGL101 and HUMA101 will appear on the schedule, but not both. This will prevent the scheduling of a single course at multiple times.

- **INSTRUCTOR**: The INSTRUCTOR field contains the name of the instructor or instructors of the course, identified by “LastName,FirstName,MiddleInitial” for each instructor (for example, “Redl,TimothyA.”). In the event that a course to be offered does not have an instructor prior to scheduling, the INSTRUCTOR field will be marked as such with a “-”. It is assumed, however, that the majority of courses to be scheduled will have their instructor(s) known prior to scheduling.

*Optional Data Fields:*

- **NUM_DAYS**: The NUM_DAYS field allows for a preference for the number of days each week that the course is to meet at its scheduled time (i.e., “3”, “2”, or “1”).

- **DAYS**: The DAYS field allows for a preference for the days of the week that the course will meet. Monday = “M”, Tuesday = “T”, Wednesday = “W”, Thursday = “R”, and Friday = “F”. Again, we assume that courses meeting 3 times per week will meet on Mondays, Wednesdays, and Fridays, so if NUM_DAYS = “3”, then DAYS = “MWF”. We also assume that courses meeting 2 times per week will meet on Tuesdays and Thursdays, so if NUM_DAYS = “2”, then DAYS = “TR”. Courses meeting 1 time per week can meet on one of either Mondays, Tuesdays, Wednesdays, Thursdays, or Fridays, so if NUM_DAYS = “1”, then DAYS = “M”, “T”, “W”, “R” or “F”. There are no courses scheduled on Saturdays and Sundays.
• **TIME_OF_DAY:** The TIME_OF_DAY field allows for a preference for the general time of day (i.e., morning, afternoon, or evening) that the course will meet. A "1" denotes a morning preference, a "2" denotes an afternoon preference, and a "3" denotes an evening preference. A preference for the course to be held "not in the evening" can also be specified, and is denoted in the TIME_OF_DAY field by a "4".

• **START_TIME:** The START_TIME field allows for a preference for the specific time that course will begin on its scheduled day(s) to meet, given in “military time” (for example, “0800” for 8:00 a.m., or “1300” for 1:00 p.m.). Again, we assume that courses will meet for a total of 3 hours per week. Courses meeting 3 times per week will meet on Mondays, Wednesdays, and Fridays for periods of 1 hour each (or 50 minutes, allowing 10 minutes of travel time for students between classes). Courses meeting 2 times per week will meet on Tuesdays and Thursdays for periods of 1 hour 30 minutes each (or 80 minutes, allowing 10 minutes of travel time for students between classes). Courses meeting 1 time per week will meet for a period of 3 hours (or 170 minutes, allowing 10 minutes of travel time for students between classes) on either Mondays, Tuesdays, Wednesdays, Thursdays, or Fridays. If a START_TIME preference is listed, it must coincide with one of the start times of the “fixed” time slots, as predetermined by the scheduler (and usually in the case of university scheduling, by the university). For example, classes on Mondays, Wednesdays and Fridays may be fixed to start each hour on the hour, beginning at 8:00 a.m., with the last class beginning at 9:00 p.m., with a break for lunch from 12:00 to 1:00 p.m. In this case, therefore, a START_TIME preference for a 3-time-per-week, “MWF” class must be one of either “0800”, “0900”, “1000”, “1100”, “1300”, “1400”, “1500”, “1600”, “1700”, “1800”, “1900”, “2000”, or “2100”. To possibly aid in the timetabling procedure, the scheduler may wish to convert some or all course START_TIME preferences to appropriate TIME_OF_DAY preferences.
For example:

- START_TIME = “900” → TIME_OF_DAY = “1”,
- START_TIME = “1400” → TIME_OF_DAY = “2”, or
- START_TIME = “1900” → TIME_OF_DAY = “3”.

- ROOM_TYPE: The ROOM_TYPE field allows for a preference for a particular type of room in which the course will be taught, as dictated by the type of course (for example, “lecture”, “seminar”, or “laboratory”). Lecture rooms are generally of a larger capacity than seminar rooms, and laboratory rooms are generally reserved for laboratory science courses, such as chemistry, biology, or physics, for example. Each course will require one of these types of rooms.

- ROOM: The ROOM field allows for a preference for a specific room in which the course will be taught, identified by building name (usually abbreviated) and room number (for example, “DH1049” indicates a request for Duncan Hall, Room 1049). It may appear unlikely that most or even many course requests for a particular room will be honored, due to the fact that only one course can occupy a given room at any one time. A request for a particular room type would probably be more realistic, given that there are often several rooms of the same type at a college or university. Furthermore, some rooms may be reserved by particular departments for their courses only, such as a chemistry laboratory room, for example, while other rooms are general usage rooms, for any department to use.

- CLASS_SIZE: The CLASS_SIZE field allows one to specify the expected enrollment of the course, if known, in number of students (for example, “30”). Since scheduling takes place prior to when students register and select their courses, it may be necessary to rely on past offerings of the course in previous semesters to obtain this information. This information will likely be helpful in later assigning courses to available classrooms. Clearly, no course may be
assigned to a room which has a smaller capacity than the number of students enrolled in the course.

- **CLASS_MAX_SIZE:** The CLASS_MAX_SIZE field allows one to specify the maximum enrollment of the course, in number of students (for example, "30"), if such a maximum number exists. Like CLASS_SIZE, this information will likely also be helpful in later assigning courses to available classrooms. Clearly, no course may be assigned to a room which has a smaller capacity than the maximum enrollment of the course.

- **OTHER_SPECIAL_REQUESTS:** This field allows for any other special requests, preferences or information with regards to a particular course's scheduling and assignment to a time slot and classroom.

Any optional data fields that are left blank will be marked as such with a "—".

### 5.3 Creating a Course Conflict Graph

Once we have gathered all of the necessary university course data for a given semester as described above, we can construct a course timetabling conflict graph reflecting the given data. Recall that in a conflict graph, the vertices represent the items of interest, and an edge connects each pair of "incompatible" or "conflicting" items.

In this section we present two methods of constructing a course conflict graph that can be used in our timetabling-by-graph-coloring model. Both methods employ a modification to and extension of the simple conflict graph construction method that was described in Section 3.1 and was illustrated with an example in Figure 3.1.

#### 5.3.1 Method A: One Course → One Vertex

One way to construct our course conflict graph $G$ is as follows. Suppose we have a set of $n$ courses $\{c_1, c_2, \ldots, c_n\}$ to be scheduled. Each course $c_i$ will be represented by ex-
actly one vertex $v_i$ in $G$. Therefore $G$ contains $n$ vertices, and $V(G) = \{v_1, v_2, \ldots, v_n\}$.

Each of the $n$ vertices in $G$ is first classified as belonging to exactly one of ten "groups". See Figure 5.1 below.

![Figure 5.1: Partitioning the $n$ vertices of $G$ into ten "groups"](image)

*Group 1*: Vertices corresponding to courses with NUM_DAYS = “3”, DAYS = “MWF”, and TIME_OF_DAY = “1” (i.e., “morning”)
Group 2: Vertices corresponding to courses with NUM DAYS = "3", DAYS = "MWF", and TIME OF DAY = "2" (i.e., "afternoon")

Group 3: Vertices corresponding to courses with NUM DAYS = "3", DAYS = "MWF", and TIME OF DAY = "3" (i.e., "evening")

Group 4: Vertices corresponding to courses with NUM DAYS = "3", DAYS = "MWF", and TIME OF DAY = "4" or "-" (i.e., "not in the evening" or "no preference")

Group 5: Vertices corresponding to courses with NUM DAYS = "2", DAYS = "TR", and TIME OF DAY = "1" (i.e., "morning")

Group 6: Vertices corresponding to courses with NUM DAYS = "2", DAYS = "TR", and TIME OF DAY = "2" (i.e., "afternoon")

Group 7: Vertices corresponding to courses with NUM DAYS = "2", DAYS = "TR", and TIME OF DAY = "3" (i.e., "evening")

Group 8: Vertices corresponding to courses with NUM DAYS = "2", DAYS = "TR", and TIME OF DAY = "4" or "-" (i.e., "not in the evening" or "no preference")

Group 9: Vertices corresponding to courses with NUM DAYS = "-"

Group 10: All remaining vertices, namely those corresponding to courses with NUM DAYS = "1"

Note 1: If all courses either list no NUM DAYS preference, or similarly, if no additional information is provided for any course beyond the required data fields of COURSE_ID and INSTRUCTOR (i.e., we have the bare minimum amount of information required for our problem), then all of the vertices of $G$ will be classified as Group 9 vertices.

Note 2: If all courses list either a NUM DAYS preference of "2" or "3", then Group 9 will be empty (i.e., it will contain none of the vertices of $G$). If we eliminate all 3-hour (i.e., 1-day-a-week) courses from our model, then Group 10 will be empty (i.e., it will contain none of the vertices of $G$).
Once all of the \( n \) vertices have been categorized accordingly and placed into their appropriate groups, edges can be added to the graph. Edges in the conflict graph indicate pairs of courses that we either cannot or do not want to schedule at the same time. Some edges will reflect essential timetabling conditions, or hard constraints. These edges must be added to the graph to eliminate conflicts. Other edges may be added to reflect preferential timetabling conditions, or soft constraints.

We can add edges to the conflict graph \( G \) in the following manner:

If the INSTRUCTOR (or one of multiple instructors) of course \( c_i \) and the INSTRUCTOR (or one of multiple instructors) of course \( c_j \) are the same, then we must add an edge between vertex \( v_i \) and vertex \( v_j \), since courses \( c_i \) and \( c_j \) cannot be scheduled for the same time slot. An instructor cannot teach two courses at the same time! In fact, an instructor teaching \( k \) courses will induce a clique of order \( k \) in the course conflict graph, since each of the \( k \) courses he or she teaches must be scheduled for a different time slot.

If the ROOM requested for course \( c_i \) and the ROOM requested for course \( c_j \) are the same, then in order to allow both courses to be assigned that room, we can add an edge between vertex \( v_i \) and vertex \( v_j \), so that courses \( c_i \) and \( c_j \) will not be scheduled for the same time slot. Again, we assume that only one course will occupy a classroom at any given time. In fact, if a particular room is requested by \( r \) different courses, and we wish to allow all \( r \) of those courses to be assigned that room, then we can introduce a clique of order \( r \) to our graph, to insure that each of the \( r \) courses is scheduled for a different time slot. As alluded to in Section 5.2, we may not be able to honor all or even most requests for a particular room. We may not have to, however. There are often several rooms of the same type (e.g., lecture, seminar, or laboratory) at a college or university, and honoring a request for a particular room type would probably be easier than one for a particular room. In the event that the requested room is unavailable, there may be another available room of similar type and size that would suit just as well.
Edges can also be added to reflect NUM_DAYS, DAYS, and TIME_OF_DAY preferences. First, for example, using the classification of the vertices of the conflict graph into "groups" as described above, we can add an edge:

(1) between each vertex in Group 1 and each vertex in Group 5, and
(2) between each vertex in Group 1 and each vertex in Group 6, and
(3) between each vertex in Group 1 and each vertex in Group 7, and
(4) between each vertex in Group 1 and each vertex in Group 8, and
(5) between each vertex in Group 2 and each vertex in Group 5, and
(6) between each vertex in Group 2 and each vertex in Group 6, and
(7) between each vertex in Group 2 and each vertex in Group 7, and
(8) between each vertex in Group 2 and each vertex in Group 8, and
(9) between each vertex in Group 3 and each vertex in Group 5, and
(10) between each vertex in Group 3 and each vertex in Group 6, and
(11) between each vertex in Group 3 and each vertex in Group 7, and
(12) between each vertex in Group 3 and each vertex in Group 8, and
(13) between each vertex in Group 4 and each vertex in Group 5, and
(14) between each vertex in Group 4 and each vertex in Group 6, and
(15) between each vertex in Group 4 and each vertex in Group 7, and
(16) between each vertex in Group 4 and each vertex in Group 8.

Adding edges to the conflict graph $G$ in the manner described in lines (1) – (16) above reflects both NUM_DAYS and DAYS preferences. As a result of adding such edges, a course $c_i$ that requests to meet 3 days per week (i.e., on Mondays, Wednesdays, and Fridays) would ultimately be scheduled for a different time slot than a course $c_j$ that requests to meet 2 days per week (i.e., on Tuesdays and Thursdays).

We can also add an edge:

(17) between each vertex in Group 1 and each vertex in Group 2, and
(18) between each vertex in Group 1 and each vertex in Group 3, and
(19) between each vertex in Group 2 and each vertex in Group 3, and
(20) between each vertex in Group 5 and each vertex in Group 6, and
(21) between each vertex in Group 5 and each vertex in Group 7, and
(22) between each vertex in Group 6 and each vertex in Group 7.

Adding edges to the conflict graph $G$ in the manner described in lines (17) – (22) above strictly reflects TIME_OF_DAY preferences. As a result of adding such edges, two courses that request to meet on the same days but at different times of the day (for example, one course $c_i$ requests to meet on Mondays, Wednesdays, and Fridays in the morning, while another course $c_j$ requests to meet on Mondays, Wednesdays, and Fridays in the afternoon) would ultimately be scheduled for different time slots.

We can also add an edge:

(23) between each vertex in Group 3 and each vertex $v_i$ in Group 4, if the TIME_OF_DAY preference of the corresponding course $c_i = "4"$ (i.e., “not in the evening”), and
(24) between each vertex in Group 7 and each vertex $v_i$ in Group 8, if the TIME_OF_DAY preference of the corresponding course $c_i = "4"$ (i.e., “not in the evening”).

Adding edges to the conflict graph $G$ in the manner described in lines (23) – (24) above also reflects TIME_OF_DAY preferences, specifically “not in the evening” TIME_OF_DAY preferences. As a result of adding such edges, courses that request to meet “not in the evening” would ultimately not be scheduled for evening time slots. A course $c_i$ represented by a vertex in either Group 4 or Group 8 for which the TIME_OF_DAY preference of $c_i = "-"$ (i.e., “no preference”) may or may not ultimately be scheduled for an evening time slot.

Vertices in Group 9 correspond to courses with NUM_DAYS = "-". We assume that such courses without a NUM_DAYS preference will ultimately be scheduled to meet for either 3 days per week (i.e., on Mondays, Wednesdays, and Fridays) or 2 days per week (i.e., on Tuesdays and Thursdays).
To reflect any TIME_OF_DAY preferences for courses represented by vertices in Group 9, we can add edges to the conflict graph $G$ in the following manner. For each vertex $v_i$ in Group 9 corresponding to a course $c_i$, we can add an edge:

(25) between $v_i$ and each vertex in Groups 2, 3, 6, and 7, if the TIME_OF_DAY preference of $c_i$ = "1" (i.e., "morning"), or
(26) between $v_i$ and each vertex in Groups 1, 3, 5, and 7, if the TIME_OF_DAY preference of $c_i$ = "2" (i.e., "afternoon"), or
(27) between $v_i$ and each vertex in Groups 1, 2, 5, and 6, if the TIME_OF_DAY preference of $c_i$ = "3" (i.e., "evening"), or
(28) between $v_i$ and each vertex in Groups 3 and 7, if the TIME_OF_DAY preference of $c_i$ = "4" (i.e., "not in the evening").

Vertices in Group 10 correspond to courses with NUM_DAYS = "1". Such courses will be scheduled to meet for 1 day per week on either Monday, Tuesday, Wednesday, Thursday, or Friday. If Group 10 is nonempty, we can reflect any DAYS and/or TIME_OF_DAY preferences for courses represented by vertices in Group 10 by adding edges to the conflict graph $G$ in the following manner. For each vertex $v_i$ in Group 10 corresponding to a course $c_i$, we can add an edge:

(29) between $v_i$ and each vertex in Groups 5, 6, 7, and 8, if the DAY preference of $c_i$ = "M", "W", or "F", or
(30) between $v_i$ and each vertex in Groups 1, 2, 3, and 4, if the DAY preference of $c_i$ = "T" or "R", and
(31) between $v_i$ and each vertex in Groups 2, 3, 6, and 7, if the TIME_OF_DAY preference of $c_i$ = "1", or
(32) between $v_i$ and each vertex in Groups 1, 3, 5, and 7, if the TIME_OF_DAY preference of $c_i$ = "2", or
(33) between $v_i$ and each vertex in Groups 1, 2, 5, and 6, if the TIME_OF_DAY preference of $c_i$ = "3", or
(34) between $v_i$ and each vertex in Groups 3 and 7, if the TIME_OF_DAY preference of $c_i = "4"$.

5.3.2 Method B: One Course $\rightarrow$ Multiple Vertices

An alternative way to construct a course conflict graph $G$ is to ultimately represent each course by a collection of vertices, with each vertex in the collection corresponding to a possible room assignment for that course. Again, suppose we have a set of $n$ courses $\{c_1, c_2, \ldots, c_n\}$ to be scheduled, and as was the case with Method A, we have represented each course $c_i$ by exactly one vertex $v_i$, and classified each of the $n$ vertices of $G$ as belonging to exactly one of ten "groups", as described above. Now, also, suppose that we have a set of $m$ rooms $\{r_1, r_2, \ldots, r_m\}$ to which the $n$ courses can be assigned. Given these $m$ rooms, suppose a subset of them (say, $\{r_1, r_3, r_5, r_6, r_7\}$, for example) are possible room assignments for a particular course $c_i$, based on a variety of factors (e.g., CLASS_SIZE and ROOM_TYPE preferences). We now represent course $c_i$ in our course conflict graph $G$ by transforming vertex $v_i$ into a "bag" of 5 vertices, $[v_ir_1, v_ir_3, v_ir_5, v_ir_6, v_ir_7]$, with one vertex for each possible room assignment for course $c_i$. In the event that a course $c_i$ has only one possible room assignment $r_k$, course $c_i$ will be represented by exactly one vertex, namely $v_ir_k$.

We can add edges to our conflict graph $G$ in the following manner:

Suppose that two courses $c_i$ and $c_j$ must be scheduled for different time slots. They may have the same INSTRUCTOR, or they may have different NUM_DAYS preferences, or there may be some other reason for this potential timetabling conflict. The edge that would have been added between vertex $v_i$ and vertex $v_j$ in the course conflict graph using Method A now becomes a complete bipartite subgraph induced on the "bags" of vertices corresponding to the courses $c_i$ and $c_j$ using Method B. For example, if course $c_i$ and course $c_j$ correspond to "bags" of vertices $[v_ir_1, v_ir_3, v_ir_5, v_ir_6, v_ir_7]$ and $[v_jr_1, v_jr_4, v_jr_5, v_jr_6]$, respectively, then vertices $v_ir_1, v_ir_3, v_ir_5, v_ir_6$, and $v_ir_7$ form the first partite set, while vertices $v_jr_1, v_jr_4, v_jr_5$, and $v_jr_6$ form the second partite
set. See Figure 5.2 below.

![Diagram of course assignments](image)

Figure 5.2: Courses $c_i$ and $c_j$ must be assigned different time slots (Method B)

Alternatively, suppose that two courses $c_i$ and $c_j$ do not pose a potential scheduling conflict; that is, they may (but will not necessarily be) scheduled for the same time slot. While there would be no need to add an edge between vertex $v_i$ and vertex $v_j$ using Method A, we do introduce edges between identical pairs of room vertices $v_ir_k$ and $v_jr_k$ using Method B. If $c_i$ and $c_j$ have $t$ possible room assignments in common, then we introduce $t$ edges between the “bag” of vertices represented by $c_i$ and the “bag” of vertices represented by $c_j$. For example, if course $c_i$ and course $c_j$ correspond to “bags” of vertices $[v_ir_1, v_ir_3, v_ir_5, v_ir_6, v_ir_7]$ and $[v_ir_1, v_ir_4, v_ir_5, v_ir_6]$, respectively, then we would add an edge between vertex $v_ir_1$ and $v_jr_1$, an edge between $v_ir_5$ and $v_jr_5$, and an edge between $v_ir_6$ and $v_jr_6$. If courses $c_i$ and $c_j$ are ultimately scheduled for the same time slot, they cannot be assigned to the same room. See Figure 5.3 below.
Figure 5.3: Courses $c_i$ and $c_j$ may be assigned the same time slot (Method B)

5.4 Coloring the Conflict Graph

Once we have constructed a course timetabling conflict graph reflecting our data, we can then properly color the vertices of our graph, and ultimately use that proper vertex coloring to construct a conflict-free timetable of courses. Recall that in a proper vertex coloring of a graph $G$, a pair of vertices $v_i$ and $v_j$ are colored with different colors if there is an edge between them. Vertices that do not share an edge may be colored with different colors, or they may be colored the same color.

In this section we present several ways in which our course conflict graph can be properly colored as part of our timetabling-by-graph-coloring model. We address specific methods for coloring the type of conflict graph described in Section 5.3.1, in which each course to be scheduled is represented by exactly one vertex (Method A), as well as the type of conflict graph described in Section 5.3.2, in which each course is represented by a collection or “bag” of vertices (Method B).

5.4.1 Coloring a “Method A” Conflict Graph

Our general approach to graph coloring will follow that of the sequential graph coloring algorithm, as described in Section 2.2 of this dissertation. Sequential graph
coloring algorithms operate according to a greedy approach, and are commonly referred to as "greedy algorithms". Recall that a greedy graph coloring algorithm examines each vertex of the graph one at a time according to some particular order and tries to color the vertex with one of the colors used so far; that is, it tries to add the vertex to one of the existing color classes. If it is not possible to color the vertex with any existing color, then a new color class is created and the vertex is assigned the color of that new class. Greedy sequential graph coloring algorithms attempt to properly color a graph using the minimum number of colors possible.

In Section 2.2 we described four "color search" variations of a greedy or sequential approach to graph coloring, resulting in four greedy graph coloring algorithms that differ only in how to make the choice of which (existing) color class to use for a particular vertex. When such a choice of available colors exists for a particular vertex, each algorithm selects the first color class encountered, according to the respective "color searching procedure" for that algorithm. These four greedy graph coloring algorithms (SIMPLE-SEARCH GREEDY (SSG), LARGEST-FIRST-SEARCH GREEDY (LFSG), SMALLEST-FIRST-SEARCH GREEDY (SFSG), and RANDOM-SEARCH GREEDY (RSG)) were each implemented with the ability to accommodate each of four differential initial orderings of the vertices of the graph: 1) a specific predefined ordering; 2) an ordering by decreasing degree (i.e., largest-degree vertices first); 3) an ordering by smallest degree (i.e., smallest-degree vertices first); and 4) a purely random ordering of the vertices. The four greedy graph coloring algorithms, each with four potentially different initial vertex orderings, results in implementations of 16 different variations of the greedy or sequential approach to graph coloring.

We will ultimately employ one such variation of the sequential graph coloring algorithm mentioned above to properly color our conflict graph and partition its vertices into independent color classes. These color classes designate sets of courses which can safely be assigned to the same time slot without conflicts. Vertices corresponding to
courses which cannot be assigned to the same time slot due to potential conflicts will be colored with different colors in our model.

An interesting and important question to consider, therefore, is the following: Which of our 16 greedy graph coloring variations should we use in our timetabling-by-graph-coloring model? To test the effects of a variety of different color searching procedures as well as the effects of a variety of different initial vertex orderings on graph coloring performance, we conducted computational experiments involving each of the above 16 greedy algorithm variations and the coloring of different graphs arising from data from actual university course timetabling problems. Results of such experiments, along with an analysis of said results, appear at the beginning of Chapter 6 of this dissertation.

One of the preferential timetabling conditions described in Section 5.1 calls for an attempt to maintain an even distribution of courses among time slots, which should ultimately aid in room assignment and also allow for maximum utilization of university resources. Such a condition is often strongly preferred by both small-sized and large-sized universities, such as Rice University and the University of Houston, respectively. Therefore, for purposes of university course timetabling, it appears that one particular "color searching procedure", namely the one corresponding to the SMALLEST-FIRST-SEARCH GREEDY algorithm variation, may be most preferable to use in coloring our conflict graph. During the coloring procedure, SFSG ranks the color classes by the number of vertices currently in them, and searches this ranked list of color classes in order by ascending size. Based on this searching heuristic, SFSG attempts to balance the size of the color classes. If color classes correspond to time slots in our timetabling-by-graph-coloring model, then the SFSG color searching procedure should greatly aid in satisfying the above preferential timetabling condition of even distribution of courses among time slots as much as possible.

With regards to initial orderings of the vertices of the graph before coloring, our computational tests illustrate that coloring vertices in order of decreasing degree
together with the color searching heuristic found in SMALLEST-FIRST-SEARCH GREEDY (i.e., variation SMALLEST-FIRST-SEARCH GREEDY: DECREASING DEGREE (SFSG:DD)) produces the most balanced and evenly distributed color classes. Similar (and in some cases, more desirable) results were obtained by coloring the vertices according to a specific predefined ordering using coloring variation SMALLEST-FIRST-SEARCH GREEDY: SPECIFIED ORDER (SFSG:SO) and an initial vertex ordering which we describe below.

Recall from Section 5.3.1 that properly coloring a course conflict graph created using Method A involves assigning each vertex of the graph to a color class, and each vertex represents an individual course to be scheduled. Each of the vertices of $G$ has already been classified during the conflict graph construction process as belonging to exactly one of ten “groups”, which are labeled Group 1, Group 2, ..., Group 10. We propose coloring the vertices of our graph in “group order”, by first coloring all the vertices in Group 1 (i.e., the “MWF morning” requested courses), then all of the vertices in Group 2 (i.e., the “MWF afternoon” requested courses), then all of the vertices in Group 3 (i.e., the “MWF evening” requested courses), then all of the vertices in Group 4 (i.e., the “MWF not in the evening” or “MWF no time preference” requested courses), then all of the vertices in Group 5 (i.e., the “TR morning” requested courses), then all of the vertices in Group 6 (i.e., the “TR afternoon” requested courses), then all of the vertices in Group 7 (i.e., the “TR evening” requested courses), then all of the vertices in Group 8 (i.e., the “TR not in the evening” or “TR no time preference” requested courses), then all of the vertices in Group 9 (i.e., the “no days preference” requested courses, and finally all of the vertices in Group 10 (i.e., all other courses, namely those requested to meet for only 1 day per week).

Additionally, the vertices in each of the ten groups can first be ordered before any actual coloring begins. Vertices will then be colored in that order. Since each vertex corresponds to exactly one course that is to be scheduled, ordering the vertices in
each group corresponds to ordering the courses corresponding to the vertices in that group.

There are various ways in which we can order the courses/vertices in each group. Within each vertex group we could:

(a) sort courses by COURSE_ID, so that in each vertex group we color all vertices corresponding to courses from one academic department first, then another department, then another, and so on, until all vertices have been colored; if desired, we could choose to “color” particular departments before others, if it might ultimately produce a more acceptable schedule (for example, departments with a larger number of courses to be scheduled in a particular vertex group could be colored before departments with fewer courses to be scheduled), or

(b) sort courses by INSTRUCTOR, so that in each vertex group we color all vertices corresponding to courses to be taught by one instructor first, then another instructor, then another, and so on, until all vertices have been colored; if desired, we could choose to “color” particular instructors before others, if it might ultimately produce a more acceptable schedule (for example, instructors with a larger number of courses to be scheduled in a particular vertex group could be colored before instructors with fewer courses to be scheduled, and courses currently without an instructor could be colored last), or

(c) sort courses in the order in which their preference requests were received by the scheduler, so that in each vertex group, we “reward” courses for submitting their preference requests early (i.e., course vertices are colored on a “first come, first served” basis), or

(d) sort courses in random order, so as not to intentionally “favor” any particular course over another when coloring, or

(e) use some other criteria by which to sort courses within each vertex group (for example, we could sort courses by CLASS_SIZE or by (CLASS_MAX_SIZE), so that in each vertex group we color vertices corresponding to larger-sized courses first, followed
by those corresponding to smaller-sized courses).

In addition, we could choose to "precolor" particular vertices in our course conflict graph (i.e., "preschedule" particular courses), based on some priority criteria. Such vertices or courses would be preassigned a specific color and/or preassigned a specific time slot prior to the actual coloring and timetabling procedure, in such a way as to satisfy an essential timetabling condition that any courses designated as "prescheduled" should be scheduled to their respective specified time slots. Of course, we must be careful to properly precolor vertices corresponding to such courses, so as not to induce any immediate consequential scheduling conflicts!

Coloring according to the predefined "group order" described above provides us with the added benefit of a general grouping and ordering of the resulting color classes by what should ultimately categorize their corresponding time slots, reflecting the satisfaction of the NUM_DAYS, DAYS, and TIME_OF_DAY scheduling preferences of courses to be scheduled. If we number the color classes 1, 2, \ldots, k in the order they are created during coloring, then the following courses/vertices will be assigned the following colors:

- a course held during MWF mornings is assigned a color from \{1, 2, \ldots, a\},
- a course held during MWF afternoons is assigned a color from \{a + 1, a + 2, \ldots, b\},
- a course held during MWF evenings is assigned a color from \{b + 1, b + 2, \ldots, c\},
- a course held during TR mornings is assigned a color from \{c + 1, c + 2, \ldots, d\},
- a course held during TR afternoons is assigned a color from \{d + 1, d + 2, \ldots, e\},

and

- a course held during TR evenings is assigned a color from \{e + 1, e + 2, \ldots, k\},

where 0 < a < b < c < d < e < k.

At this point, we will have colored all of the vertices corresponding to courses which will ultimately be scheduled to meet for either 3 times per week (i.e., for periods of 1 hour on Mondays, Wednesdays, and Fridays) or 2 times per week (i.e.,
for periods of $1\frac{1}{2}$ hours on Tuesdays and Thursdays). We assume that all of the time slots assigned to the above courses will be non-overlapping (for example, the time slots preceding and following “Monday/Wednesday/Friday, 9:00–10:00 a.m.” would be “Monday/Wednesday/Friday, 8:00–9:00 a.m.” and “Monday/Wednesday/Friday, 10:00–11:00 a.m.”, respectively).

Courses meeting 1 time per week (i.e., on either Mondays, Tuesdays, Wednesdays, Thursdays, or Fridays) will ultimately be scheduled for 3-hour time slots. We assume that these 3-hour time slots will not overlap with each other, but will coincide with time slots designated for courses scheduled to meet for 3 times per week or 2 times per week. The 3-hour time slots on Monday, Wednesday, and Friday will coincide with 3 consecutive MWF 1-hour time slots, and the 3-hour time slots on Tuesday and Thursday will coincide with 2 consecutive TR $1\frac{1}{2}$-hour time slots. We propose four different ways of handling the coloring of these “Group 10” vertices corresponding to courses that will meet 1 time per week during 3-hour time slots. These four different methods, which we call the “Simple Method”, the “Time Block Method”, the “Time Slot Method”, and the “2-or-3-Color Method”, will influence the ways in which our proper coloring of the conflict graph can ultimately be transformed into a conflict-free course timetable (to be described later in Section 5.5).

5.4.1.1 Coloring “Group 10” Vertices: Simple Method

The simple method of coloring “Group 10” vertices involves simply coloring the vertices in Group 10 immediately after all of the vertices in Groups 1 through 9 have already been colored, taking care to properly color the vertices in Group 10 based on any scheduling preferences or potential scheduling conflicts among courses represented by such vertices, or between vertices in Group 10 and previously-colored vertices belonging to Groups 1 through 9. If we wish, we can sort the vertices in Group 10 prior to coloring, by employing any of the ordering procedures described above or by using some other ordering criteria. A vertex $v_i$ in Group 10 corresponding
to a course $c_i$ that is to be assigned to one of the following 3-hour time slots will be colored from among the following colors:

- Course $c_i$ to be assigned a Monday morning, Wednesday morning, or Friday morning 3-hour time slot $\leftrightarrow$ Vertex $v_i$ colored a color from $\{1, 2, \ldots, a\}$,
- Course $c_i$ to be assigned a Monday afternoon, Wednesday afternoon, or Friday afternoon 3-hour time slot $\leftrightarrow$ Vertex $v_i$ colored a color from $\{a + 1, a + 2, \ldots, b\}$,
- Course $c_i$ to be assigned a Monday evening, Wednesday evening, or Friday evening 3-hour time slot $\leftrightarrow$ Vertex $v_i$ colored a color from $\{b + 1, b + 2, \ldots, c\}$,
- Course $c_i$ to be assigned a Tuesday morning or Thursday morning 3-hour time slot $\leftrightarrow$ Vertex $v_i$ colored a color from $\{c + 1, c + 2, \ldots, d\}$,
- Course $c_i$ to be assigned a Tuesday afternoon or Thursday afternoon 3-hour time slot $\leftrightarrow$ Vertex $v_i$ colored a color from $\{d + 1, d + 2, \ldots, e\}$, and
- Course $c_i$ to be assigned a Tuesday evening or Thursday evening 3-hour time slot $\leftrightarrow$ Vertex $v_i$ colored a color from $\{e + 1, e + 2, \ldots, k\}$

5.4.1.2 Coloring “Group 10” Vertices: Time Block Method

The time block method of coloring “Group 10” vertices assumes that we have first already colored all of the vertices in Groups 1 through 9, and assigned time blocks to the courses corresponding to these vertices, before we can color any of the vertices in Group 10. Once all of the courses corresponding to vertices in Groups 1 through 9 have been colored and assigned time blocks, we classify each of these vertices as belonging to one of the following six “time block groups”. See Figure 5.4 below.

*Time Block Group 1* (i.e., time block “MWF, morning”): Vertices previously in Group 1 and all other vertices assigned a color from $\{1, 2, \ldots, a\}$

*Time Block Group 2* (i.e., time block “MWF, afternoon”): Vertices previously in Group 2 and all other vertices assigned a color from $\{a + 1, a + 2, \ldots, b\}$

*Time Block Group 3* (i.e., time block “MWF, evening”): Vertices previously in
Group 3 and all other vertices assigned a color from \( \{b + 1, b + 2, \ldots, c\} \)

*Time Block Group 4 (i.e., time block “TR, morning”)*: Vertices previously in Group 5 and all other vertices assigned a color from \( \{c + 1, c + 2, \ldots, d\} \)

*Time Block Group 5 (i.e., time block “TR, afternoon”)*: Vertices previously in Group 6 and all other vertices assigned a color from \( \{d + 1, d + 2, \ldots, e\} \)

*Time Block Group 6 (i.e., time block “TR, evening”)*: Vertices previously in Group
7 and all other vertices assigned a color from \( \{e + 1, e + 2, \ldots, k\} \)

At this point we can implement the time block method and "color" the remaining vertices (i.e., the vertices in Group 10) by assigning each of these vertices to one of the six time block groups above. Instead of adding edges to our conflict graph \( G \) from vertices in Group 10 to vertices in Groups 1 through 9 as described in lines (29) – (34) in Section 5.3.1, and to accommodate our new time block group arrangement of vertices previously arranged in Groups 1 through 9, we can now, for each vertex \( v_i \) in Group 10 corresponding to a course \( c_i \), add an edge:

(29a) between \( v_i \) and each vertex in Time Block Groups 4, 5, and 6, if the DAY preference of \( c_i = "M", "W", \text{ or } "F" \), or
(30a) between \( v_i \) and each vertex in Time Block Groups 1, 2, and 3, if the DAY preference of \( c_i = "T" \text{ or } "R" \), and
(31a) between \( v_i \) and each vertex in Time Block Groups 2, 3, 5, and 6, if the TIME_OF_DAY preference of \( c_i = "1" \), or
(32a) between \( v_i \) and each vertex in Time Block Groups 1, 3, 4, and 6, if the TIME_OF_DAY preference of \( c_i = "2" \), or
(33a) between \( v_i \) and each vertex in Time Block Groups 1, 2, 4, and 5, if the TIME_OF_DAY preference of \( c_i = "3" \), or
(34a) between \( v_i \) and each vertex in Time Block Groups 3 and 6, if the TIME_OF_DAY preference of \( c_i = "4" \).

Vertices in Group 10 which have neither DAY nor TIME_OF_DAY preferences will not be incident to any of the edges described in lines (29a) – (42a) above.

Once all of the appropriate edges have been added, either to reflect scheduling preferences or to guard against potential scheduling conflicts, we can "color" the vertices in Group 10. Again, this amounts to assigning each vertex \( v_i \) in Group 10 to one of the six time block groups described above, or coloring each vertex in Group 10
with one of the following colors, corresponding to the following respective time block groups:

- Vertex $v_i$ is colored a color from $\{1, 2, \ldots, a\}$ $\iff$ Vertex $v_i$ is assigned to Time Block Group 1,
- Vertex $v_i$ is colored a color from $\{a + 1, a + 2, \ldots, b\}$ $\iff$ Vertex $v_i$ is assigned to Time Block Group 2,
- Vertex $v_i$ is colored a color from $\{b + 1, b + 2, \ldots, c\}$ $\iff$ Vertex $v_i$ is assigned to Time Block Group 3,
- Vertex $v_i$ is colored a color from $\{c + 1, c + 2, \ldots, d\}$ $\iff$ Vertex $v_i$ is assigned to Time Block Group 4,
- Vertex $v_i$ is colored a color from $\{d + 1, d + 2, \ldots, e\}$ $\iff$ Vertex $v_i$ is assigned to Time Block Group 5, and
- Vertex $v_i$ is colored a color from $\{e + 1, e + 2, \ldots, k\}$ $\iff$ Vertex $v_i$ is assigned to Time Block Group 6.

5.4.1.3 Coloring “Group 10” Vertices: Time Slot Method

When using the simple method or the time block method of coloring “Group 10” vertices as illustrated above, we do not put any “restriction” on the number of distinct 3-hour time slots in any of the six time blocks. We discuss this issue in further detail in Section 5.5, where we describe the process of transforming a proper coloring of our conflict graph $G$ to a conflict-free timetable of courses.

Alternatively, we may wish to limit the number of distinct 3-hour time slots in each time block. For example, we could assume that there will only be one 3-hour time slot per time block per day. Under this assumption, we ultimately have a total of 15 distinct 3-hour time slots (three from each of Time Block Groups 1, 2, and 3, and two from each of Time Block Groups 4, 5, and 6) in which to schedule the courses represented by vertices in Group 10. We thus have:
• one 3-hour time slot on Monday morning (from Time Block Group 1), and
• one 3-hour time slot on Monday afternoon (from Time Block Group 2), and
• one 3-hour time slot on Monday evening (from Time Block Group 3), and
• one 3-hour time slot on Tuesday morning (from Time Block Group 4), and
• one 3-hour time slot on Tuesday afternoon (from Time Block Group 5), and
• one 3-hour time slot on Tuesday evening (from Time Block Group 6), and
• one 3-hour time slot on Wednesday morning (from Time Block Group 1), and
• one 3-hour time slot on Wednesday afternoon (from Time Block Group 2), and
• one 3-hour time slot on Wednesday evening (from Time Block Group 3), and
• one 3-hour time slot on Thursday morning (from Time Block Group 4), and
• one 3-hour time slot on Thursday afternoon (from Time Block Group 5), and
• one 3-hour time slot on Thursday evening (from Time Block Group 6), and
• one 3-hour time slot on Friday morning (from Time Block Group 1), and
• one 3-hour time slot on Friday afternoon (from Time Block Group 2), and
• one 3-hour time slot on Friday evening (from Time Block Group 3).

The time slot method of coloring “Group 10” vertices assumes that we have first already assigned not only time blocks, but actual time slots to courses meeting 3 times per week or 2 times per week, before we can color any of the vertices corresponding to courses meeting 1 time per week. Once each of the courses meeting 3 times per week or 2 times per week has been assigned an actual time slot (for example, “Monday/Wednesday/Friday, 9:00–10:00 a.m.” or “Tuesday/Thursday, 9:30–11:00 a.m.”), and the start times for each of the 15 3-hour time slots listed above have been established, we can, in lieu of adding edges to our conflict graph $G$ as described in lines (29a) – (34a) above, add edges in the following manner. For each vertex $v_i$ in Group 10 corresponding to a course $c_i$, and each vertex $v_k$ from Time Block Groups 1 through 6 corresponding to a course $c_k$ that has been assigned a time slot with a start time that is different from the start time of a 3-hour time slot, we can add an
edge:

(29b) between \( v_i \) and each vertex in Time Block Groups 4, 5, and 6, and between \( v_i \) and each such vertex \( v_k \) (as described in the previous paragraph) from Time Block Groups 1, 2, and 3, if the DAY preference of \( c_i = \text{"M"}, \text{"W"}, \) or \text{"F"}, or

(30b) between \( v_i \) and each vertex in Time Block Groups 1, 2, and 3, and between \( v_i \) and each such vertex \( v_k \) (as described in the previous paragraph) from Time Block Groups 4, 5, and 6, if the DAY preference of \( c_i = \text{"T"} \) or \text{"R"}, and

(31b) between \( v_i \) and each vertex in Time Block Groups 2, 3, 5, and 6, and between \( v_i \) and each such vertex \( v_k \) (as described in the previous paragraph) from Time Block Groups 1 and 4, if the TIME_OF_DAY preference of \( c_i = \text{"1"} \), or

(32b) between \( v_i \) and each vertex in Time Block Groups 1, 3, 4, and 6, and between \( v_i \) and each such vertex \( v_k \) (as described in the previous paragraph) from Time Block Groups 2 and 5, if the TIME_OF_DAY preference of \( c_i = \text{"2"} \), or

(33b) between \( v_i \) and each vertex in Time Block Groups 1, 2, 4, and 5, and between \( v_i \) and each such vertex \( v_k \) (as described in the previous paragraph) from Time Block Groups 3 and 6, if the TIME_OF_DAY preference of \( c_i = \text{"3"} \), or

(34b) between \( v_i \) and each vertex in Time Block Groups 3 and 6, and between \( v_i \) and each such vertex \( v_k \) (as described in the previous paragraph) from Time Block Groups 1, 2, 4, and 5, if the TIME_OF_DAY preference of \( c_i = \text{"4"} \).

Furthermore, an edge will be drawn between each vertex \( v_i \) in Group 10 and each such vertex \( v_k \) in each of Time Block Groups 1 through 6, if \( c_i \) has neither DAY nor TIME_OF_DAY preferences.

Once all of the appropriate edges have been added, either to reflect scheduling preferences or to guard against potential scheduling conflicts, we can "color" the vertices in Group 10. Again, this amounts to ultimately assigning each vertex \( v_i \) in Group 10 to one of the 15 3-hour time slots within the six time block groups, as described above, or coloring each vertex in Group 10 with one of the following colors,
corresponding to the following respective time block groups:

- Vertex $v_i$ is colored a color from $\{1, 2, \ldots, a\}$ $\leftrightarrow$ Vertex $v_i$ is assigned to one of the three 3-hour time slots from Time Block Group 1,
- Vertex $v_i$ is colored a color from $\{a + 1, a + 2, \ldots, b\}$ $\leftrightarrow$ Vertex $v_i$ is assigned to one of the three 3-hour time slots from Time Block Group 2,
- Vertex $v_i$ is colored a color from $\{b + 1, b + 2, \ldots, c\}$ $\leftrightarrow$ Vertex $v_i$ is assigned to one of the three 3-hour time slots from Time Block Group 3,
- Vertex $v_i$ is colored a color from $\{c + 1, c + 2, \ldots, d\}$ $\leftrightarrow$ Vertex $v_i$ is assigned to one of the two 3-hour time slots from Time Block Group 4,
- Vertex $v_i$ is colored a color from $\{d + 1, d + 2, \ldots, e\}$ $\leftrightarrow$ Vertex $v_i$ is assigned to one of the two 3-hour time slots from Time Block Group 5, and
- Vertex $v_i$ is colored a color from $\{e + 1, e + 2, \ldots, k\}$ $\leftrightarrow$ Vertex $v_i$ is assigned to one of the two 3-hour time slots from Time Block Group 6.

**Note:** As a result of coloring vertices from Group 10 by the time slot method, only one color from $\{1, 2, \ldots, a\}$, one color from $\{a + 1, a + 2, \ldots, b\}$, one color from $\{b+1, b+2, \ldots, c\}$, one color from $\{c+1, c+2, \ldots, d\}$, one color from $\{d+1, d+2, \ldots, e\}$, and one color from $\{e+1, e+2, \ldots, k\}$ will be used as colors for “Group 10” vertices.

5.4.1.4 Coloring “Group 10” Vertices: 2-or-3-Color Method

Finally, we propose yet another alternative option for handling the coloring of “Group 10” vertices, which we call the 2-or-3-color method. This method is modeled after a graph coloring algorithm modification presented in an unpublished paper by Miner, Elmohammed, and Yau [52].

Recall that in our timetabling-by-graph-coloring model, we assume that 3-hour time slots designated for courses scheduled to meet 1 time per week will coincide with 1-hour time slots and $1\frac{1}{2}$-hour time slots designated for courses scheduled to meet 3 times per week and 2 times per week, respectively. The 3-hour time slots on Monday,
Wednesday, and Friday will coincide with three consecutive MWF 1-hour time slots, and the 3-hour time slots on Tuesday and Thursday will coincide with two consecutive TR 1.5-hour time slots. The 2-or-3-color method colors a vertex \( v_i \) corresponding to a 3-hour, 1-time-per-week course \( c_i \) with 2 colors, if \( c_i \) will be scheduled to meet on a Tuesday or Thursday, and with 3 colors, if \( c_i \) will be scheduled to meet on a Monday, Wednesday, or Friday.

For example, suppose we have three vertices \( v_i, v_j, \) and \( v_k \), representing three 1-hour, 3-time-per-week courses \( c_i, c_j, \) and \( c_k \), respectively. Furthermore, suppose that \( v_i \) has been colored with color \( a \), \( v_j \) has been colored with color \( b \), and \( v_k \) has been colored with color \( c \), and that colors \( a, b, \) and \( c \) represent the 1-hour time slots “Monday/Wednesday/Friday, 8:00–9:00 a.m.”, “Monday/Wednesday/Friday, 9:00–10:00 a.m.”, and “Monday/Wednesday/Friday, 10:00–11:00 a.m.”, respectively. Then consider a 3-hour, 1-time-per-week course \( c_p \) that has the same INSTRUCTOR as one of the three 1-hour courses, say \( c_i \). Course \( c_p \) cannot be assigned the 3-hour time slot “Monday, 8:00–11:00 a.m.”, “Wednesday, 8:00–11:00 a.m.”, or “Friday, 8:00–11:00 a.m.”, since it would conflict with \( c_i \). So those three 3-hour time slots must be represented by colors which clearly identify the fact that no vertex adjacent to a 1-hour, 3-time-per-week course vertex colored \( a, b, \) or \( c \) may be given these colors. Therefore, we can label those three 3-hour time slots with all three conflicting 1-hour course vertex colors, as “color \( a-b-c \)”. However, two different 3-hour courses having the same INSTRUCTOR could both be scheduled for the time period represented by color \( a-b-c \), as long as they were on different days. Therefore, we can label the three 3-hour time slots on Monday, Wednesday, and Friday represented by color \( a-b-c \) as “\( a-b-c-\text{Monday} \)”, “\( a-b-c-\text{Wednesday} \)”, and “\( a-b-c-\text{Friday} \)”, respectively, to distinguish them from one another.

Similarly, suppose we have two vertices \( v_i \) and \( v_j \), representing two 1.5-hour, 2-time-per-week courses \( c_i \) and \( c_j \), respectively. Furthermore, suppose that \( v_i \) has been colored with color \( a \) and \( v_j \) has been colored with color \( b \), and that colors \( a \) and \( b \)
represent the $1\frac{1}{2}$-hour time slots "Tuesday/Thursday, 8:00–9:30 a.m." and "Tuesday/Thursday, 9:30–11:00 a.m.", respectively. Then consider a 3-hour, 1-time-per-week course $c_p$ that has the same INSTRUCTOR as one of the two $1\frac{1}{2}$-hour courses, say $c_i$. Course $c_p$ cannot be assigned the 3-hour time slot "Tuesday, 8:00–11:00 a.m." or "Thursday, 8:00–11:00 a.m.", since it would conflict with $c_i$. So those two 3-hour time slots must be represented by colors which clearly identify the fact that no vertex adjacent to a $1\frac{1}{2}$-hour, 2-time-per-week course vertex colored $a$ or $b$ may be given these colors. Therefore, we can label those two 3-hour time slots with both conflicting $1\frac{1}{2}$-hour course vertex colors, as "color $a$-$b$". However, two different 3-hour courses having the same INSTRUCTOR could both be scheduled for the time period represented by color $a$-$b$, as long as they were on different days. Therefore, we can label the two 3-hour time slots on Tuesday and Thursday represented by color $a$-$b$ as "$a$-$b$-Tuesday" and "$a$-$b$-Thursday", respectively, to distinguish them from one another.

Consequently, due to the presence of coinciding time slots in our model, the 2-or-3-color method assumes that we have first already colored and scheduled the 1-hour, 3-time-per-week course vertices and the $1\frac{1}{2}$-hour, 2-time-per-week course vertices first, before coloring the vertices corresponding to the 3-hour, 1-time-per-week courses (i.e., the vertices in Group 10). This strategy bears resemblance to that of the time block method and time slot method discussed above.

### 5.4.2 Coloring a “Method B” Conflict Graph

Recall from Section 5.3.2 that the construction of a course conflict graph $G$ using Method B involves the representation of each course by a collection or "bag" of vertices, with each vertex in the "bag" corresponding to a possible room assignment for that course. If we have two courses $c_i$ and $c_j$ that ultimately are to be scheduled for different time slots, then the edge added between vertex $v_i$ and vertex $v_j$ in a "Method A" conflict graph is replaced by a complete bipartite subgraph induced on
the “bags” of vertices corresponding to the conflicting courses $c_i$ and $c_j$ in a “Method B” conflict graph. If courses $c_i$ and $c_j$ do not pose a potential scheduling conflict (i.e., they may ultimately be scheduled for the same time slot), then a “Method A” conflict graph does not have an edge between vertex $v_i$ and vertex $v_j$, while a “Method B” conflict graph does contain an edge between each pair of identical room vertices $v_i r_k$ and $v_j r_k$, to ensure that if courses $c_i$ and $c_j$ are, in fact, scheduled for the same time slot, they will not be assigned the same room. In general, any edge between two vertices $v_i$ and $v_j$ in a “Method A” conflict graph corresponds to a complete bipartite subgraph induced on the “bags” of vertices corresponding to the conflicting courses $c_i$ and $c_j$ in a “Method B” conflict graph, and any “non-edge” between two vertices $v_i$ and $v_j$ in a “Method A” conflict graph corresponds to edge between each pair of identical room vertices $v_i r_k$ and $v_j r_k$.

When coloring vertices in a “Method B” conflict graph we can follow primarily the same procedures as when coloring a “Method A” conflict graph, as described in Section 5.4.1, with one important exception. In a “Method A” conflict graph, each vertex in the graph is ultimately assigned a color, whereas in a “Method B” conflict graph, only one vertex in each “bag” of vertices in the graph is ultimately assigned a color. That is to say, each “course” represented in the graph is assigned a color. As when properly coloring a “Method A” conflict graph, each color in a properly-colored “Method B” conflict graph corresponds to a different time slot. In a “Method B” conflict graph, a colored vertex $v_i r_k$ implies that course $c_i$ has been assigned to room $r_k$. See Figures 5.5, 5.6, 5.7, 5.8, and 5.9 below.

In Figure 5.5, courses $c_i$ and $c_j$ must be scheduled for different time slots. Vertex $v_i r_3$ is colored with color 1, and vertex $v_j r_6$ is colored with color 2, implying that courses $c_i$ and $c_j$ have been assigned different rooms and different time slots.

In Figure 5.6, courses $c_i$ and $c_j$ must be scheduled for different time slots. Vertex $v_i r_5$ is colored with color 1, and vertex $v_j r_5$ is colored with color 2, implying that courses $c_i$ and $c_j$ have been assigned the same room and different time slots.
In Figure 5.7, courses $c_i$ and $c_j$ do not pose a potential scheduling conflict, and may be scheduled for the same time slot. Vertex $v_i r_6$ is colored with color 1, and vertex $v_j r_4$ is colored with color 2, implying that courses $c_i$ and $c_j$ have been assigned different rooms and different time slots.

In Figure 5.8, courses $c_i$ and $c_j$ do not pose a potential scheduling conflict, and may be scheduled for the same time slot. Vertex $v_i r_7$ is colored with color 1, and vertex $v_j r_1$ is colored with color 1, implying that courses $c_i$ and $c_j$ have been assigned
Figure 5.7: Courses $c_i$ and $c_j$ are assigned different rooms and different time slots.

Figure 5.8: Courses $c_i$ and $c_j$ are assigned different rooms and the same time slot.

different rooms and the same time slot.

In Figure 5.9, courses $c_i$ and $c_j$ do not pose a potential scheduling conflict, and *may* be scheduled for the same time slot. Vertex $v_i r_6$ is colored with color 1, and vertex $v_j r_6$ is colored with color 2, implying that courses $c_i$ and $c_j$ have been assigned the same room and different time slots.

As with coloring a "Method A" conflict graph, we can color a "Method B" conflict graph using a variation of the sequential or greedy coloring algorithm (for example,
Figure 5.9: Courses $c_i$ and $c_j$ are assigned the same room and different time slots

SMALLEST-FIRST-SEARCH-GREEDY) that only colors one vertex in each bag of vertices corresponding to a particular course. We can likewise choose the order in which we color the "bags" of a "Method B" conflict graph. For example, we can color them according to some predefined ordering, such as "group order", by using the SMALLEST-FIRST-SEARCH-GREEDY: SPECIFIED ORDER coloring variation, where each "bag" is classified as belonging to one of ten "groups", and groups of "bags" are colored in the following order: Group 1, Group 2, Group 3, Group 4, Group 5, Group 6, Group 7, Group 8, Group 9, and finally Group 10 — and "bags" within each group are colored in order, according to some preferred ordering criteria (for example, it may be advantageous to "color" courses with the fewest available rooms first, since when coloring a "Method B" conflict graph we are also assigning courses to classrooms). "Bags" in Group 10 (i.e., those that correspond to courses meeting 1 time per week) can be colored in a "Method B" conflict graph using any of the four coloring methods proposed in Section 5.4.1 for coloring vertices in a "Method A" conflict graph: 1) the simple method; 2) the time block method; 3) the time slot method; or 4) the 2-or-3-color method.

Once again, coloring a "Method B" conflict graph not only partitions the set of
courses to be scheduled into independent color classes. It also assigns each course an appropriate classroom in which to meet, while simultaneously ensuring that two courses in the same color class are assigned to different rooms, so as to avoid conflict.

When properly coloring either a "Method A" or a "Method B" course conflict graph we can place additional restrictions on the cardinalities of the color classes that result from such a coloring, so as not to exceed certain limits. Doing so will ultimately limit the number of courses that can scheduled to take place during a particular time slot, which can ultimately be an aid, and even at times a necessity, when assigning courses to classrooms.

### 5.4.3 A Restricted (List) Coloring Approach

Recall from Section 2.1 that given a graph $G$, a coloring of $G$ (also commonly referred to as a vertex-coloring) is a function $f$ from the vertices of $G$ to a set $C$ whose elements are called colors. A proper $k$-coloring of $G$ is a coloring $f$ which uses exactly $k$ colors and satisfies the property that $f(x) \neq f(y)$ whenever vertices $x$ and $y$ are adjacent in $G$. Restricted (vertex-)colorings (also referred to as list (vertex-)colorings) are extensions of (vertex-)colorings. For each vertex $v \in V(G)$, let $\phi(v) \subseteq C$ be a given set of feasible colors for $v$. The restricted coloring problem $(G, \phi)$ is to find a proper coloring of $G$ such that each vertex $v$ is assigned a color $f(v) \in \phi(v)$. The restricted $k$-colorability problem is NP-complete in the general case, since it contains as a special case the $k$-colorability problem, where for each vertex $v$ we have $\phi(v) = \{1, 2, \ldots, k\} = C$.

Studies of restricted graph coloring were published by de Werra in 1997 [21] and 1999 [22], in particular some mathematical programming models and formulations involving the perfectness, balance, or total unimodularity of constraint matrices in specific types of instances of timetabling problems.

Restricted graph coloring models have direct relevance to course timetabling problems. If vertices in a graph $G$ represent the courses to be scheduled, edges in $G$ rep-
resent conflicting pairs of courses (i.e., pairs of courses which cannot be scheduled for
the same time), colors in a proper coloring of $G$ represent time slots, and a fea-
sible timetable in $k$ time slots is associated with a $k$-coloring of $G$, then restricted graph
coloring models allow for each course $c_i$ to have its own prespecified set of available
time slots to which it can be assigned (i.e., each vertex $v_i$ in $G$ will have its own pre-
specified set of available colors with which it can be colored). With restricted graph
coloring models we can also place additional restrictions on the cardinalities of the
color classes, so as not to exceed certain limits, which again can ultimately be an aid,
and is even at times a necessity, when assigning courses to classrooms.

5.5 Transforming the Coloring to a Timetable

At this point, our course conflict graph $G$ is properly colored. If $G$ is a "Method A"
conflict graph, then each vertex of $G$ has been colored. If $G$ is a "Method B" conflict
graph, then each "bag" of vertices representing a single course will have exactly one of
its vertices colored. After coloring, the next step in our timetabling-by-graph-coloring
process is to transform our proper coloring of $G$ to a conflict-free course timetable,
which we will call $T(G)$, or simply $T$.

Recall from our discussion of simple university timetabling-by-graph-coloring mod-
elns in Chapter 3 that if we have a conflict graph $G$, where each vertex in the graph
represents a course to be scheduled, each edge represents a pair of courses that con-

flict (i.e., cannot be scheduled for the same time slot, for whatever reason), and after
$G$ has been properly colored, each color class represents a non-overlapping time slot,
then we have an extremely straightforward way of transforming a proper coloring of
$G$ to a conflict-free timetable of courses. Each course will be assigned to the time slot
designated by the color of its corresponding vertex in $G$. A $\sigma$-colored conflict graph
is equivalent to a timetable with $\sigma$ time slots. If $\chi(G) = k$, then we can transform $G$
to a conflict-free course timetable with $k$ time slots.
5.5.1 “Minimizing Pain”

In the simplest model of course timetabling, where there are no preassigned meetings or unavailability constraints, no time or time of day preferences of courses, and all courses represented by vertices that have been colored the same color are to be assigned to the same time slot, we need only be concerned with the one-to-one assignment of color classes to time slots when transforming the coloring to a timetable. Each time slot can be represented by any color class, so the question simply becomes, “Which color classes become which time slots?” In this case, we can incorporate a preferential timetabling condition presented in Section 5.1, namely to try to schedule sufficiently large courses at times that “minimize pain for students”. As part of research for a 1999 paper describing the designing of university course timetables by using an evolutionary approach, Woods and Trenaman [70] surveyed a number of university students and lecturers to determine the most and least “painful” time slots during the course of a week. Participants in the survey weighted a week’s worth of one-hour time slots on a scale of 1 to 50 according to how painful, or undesirable, these time slots were believed to be, either for taking or for teaching a class during that time. The information was collated and summed up in the following “pain table”, which lists pain values ranging from 0 to 1 (a pain value of 0 being least painful, a pain value of 1 being most painful). See Figure 5.10 below.

<table>
<thead>
<tr>
<th></th>
<th>9a</th>
<th>10a</th>
<th>11a</th>
<th>12p</th>
<th>1p</th>
<th>2p</th>
<th>3p</th>
<th>4p</th>
<th>5p</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mon</td>
<td>0.9</td>
<td>0.8</td>
<td>0.7</td>
<td>0.3</td>
<td>0.5</td>
<td>0.3</td>
<td>0.2</td>
<td>0.4</td>
<td>0.6</td>
</tr>
<tr>
<td>Tue</td>
<td>0.7</td>
<td>0.6</td>
<td>0.5</td>
<td>0.0</td>
<td>0.3</td>
<td>0.0</td>
<td>0.1</td>
<td>0.2</td>
<td>0.5</td>
</tr>
<tr>
<td>Wed</td>
<td>0.7</td>
<td>0.5</td>
<td>0.2</td>
<td>0.0</td>
<td>0.3</td>
<td>0.0</td>
<td>0.0</td>
<td>0.2</td>
<td>0.5</td>
</tr>
<tr>
<td>Thu</td>
<td>0.7</td>
<td>0.5</td>
<td>0.0</td>
<td>0.0</td>
<td>0.3</td>
<td>0.0</td>
<td>0.0</td>
<td>0.3</td>
<td>0.5</td>
</tr>
<tr>
<td>Fri</td>
<td>0.8</td>
<td>0.7</td>
<td>0.5</td>
<td>0.3</td>
<td>0.4</td>
<td>0.5</td>
<td>0.7</td>
<td>0.8</td>
<td>0.9</td>
</tr>
</tbody>
</table>

Figure 5.10: “Pain Table”: one-day, one-hour time slots (from [70])
From the data in the pain table in Figure 5.10, it appears that having class on Mondays and Fridays is generally more painful than having class on days in the middle of the week, and that early Monday morning classes and late Friday afternoon classes are the most painful of all! From the author’s past experiences as both a student and a lecturer, these results are not necessarily all that surprising!

Based on the above pain table (or some other similar pain table obtained from some other survey of “time slot painfulness”), we can match color classes to time slots so as to try to “minimize pain” for larger-sized courses. We can do this by designating those color classes containing more courses (i.e., more vertices) with a significantly large CLASS_SIZE to take place during “less-painful” time slots. Alternatively, we could designate those color classes simply containing a greater number of courses (i.e., vertices) to take place during “less-painful” time slots, in an effort to also “minimize pain” for students (and lecturers). For example, to “minimize pain”, we could schedule more large-sized courses (or simply schedule more courses) during the “Wednesday 2:00–3:00 p.m.” time slot (with a pain value of 0.0), and schedule fewer large-sized courses (or simply schedule fewer courses) during the “Monday 9:00–10:00 a.m.” time slot (with a pain value of 0.9).

As we have discussed earlier, we have three different “types” of time slots in our timetabling-by-graph-coloring model, and a course may have a DAYS preference as to whether it should be scheduled as a 3-time-per-week, a 2-time-per-week, or a 1-time-per-week course. Courses meeting 3 times per week meet on Mondays, Wednesdays, and Fridays for periods of 1 hour each. Courses meeting 2 times per week meet on Tuesdays and Thursdays for periods of 1½ hours each. Courses meeting 1 time per week for 3 hours meet on either Mondays, Tuesdays, Wednesdays, Thursdays, or Fridays. By averaging the appropriate pain values in the pain table above, we can create a new pain table reflecting the three different types of time slots in our model (see Figure 5.11 below), and then assign color classes to appropriate time slots, similar to the manner described above, in an effort to “minimize pain”.
Figure 5.11: “Pain Table”: multi-day, variable-length time slots (adapted from [70])

To “minimize pain”, we can designate color classes containing more courses (i.e., more vertices) with a large CLASS\_SIZE (or alternatively, color classes simply containing a greater number of courses) to take place during “less painful” time slots within each type of time slot. For example, suppose we have two color classes designated for Tuesday/Thursday 1 1/2-hour time slots. To “minimize pain”, we could schedule more large-sized courses (or simply schedule more courses) during the “Tuesday/Thursday 1:30–3:00 p.m.” afternoon time slot (with a pain value of 0.2), and schedule fewer large-sized courses (or simply schedule fewer courses) during the Tuesday/Thursday 9:00–10:30 a.m.” morning time slot (with a pain value of 0.6).

Now, consider the above simplified timetabling model with multi-day, variable length time slots, but adding the possibility of allowing for not only DAYS preferences, but also TIME\_OF\_DAY preferences for courses.

We can designate color classes containing more courses (i.e., more vertices) with a large CLASS\_SIZE (or alternatively, color classes simply containing a greater number of courses) to take place during “less painful” time slots within each type of time slot. For example, suppose we have two color classes designated for Tuesday/Thursday 1 1/2-hour time slots in the morning. To “minimize pain”, we could schedule more large-sized courses (or simply schedule more courses) during the “Tuesday/Thursday 10:30
a.m.-12:00 p.m.” morning time slot (with a pain value of 0.4), and schedule fewer large-sized courses (or simply schedule fewer courses) during the Tuesday/Thursday 9:00–10:30 a.m.” morning time slot (with a pain value of 0.6).

We could alternatively conduct another survey similar to that of Woods and Trenaman, in an attempt to obtain possibly even more accurate pain values for the different types of time slots described above. We could also look to obtain additional pain values corresponding to evening time slots, and/or earlier morning time slots (for example, 8:00 a.m.).

5.5.2 Color Classes and Time Slots

Again, in the case of non-overlapping time slots, each different color in our properly colored course conflict graph (using either a “Method A” or a “Method B” conflict graph) corresponds to a different time slot in our conflict-free course timetable. A $k$-colored conflict graph is equivalent to a timetable with $k$ time slots.

Recall that if we color the vertices of our graph in “group order”, as described in Section 5.4.1, then the vertices from Groups 1 through 9 will receive the following colors:

- Group 1 vertices will receive colors from $\{1, 2, \ldots, a\}$,
- Group 2 vertices will receive colors from $\{a + 1, a + 2, \ldots, b\}$,
- Group 3 vertices will receive colors from $\{b + 1, b + 2, \ldots, c\}$,
- Group 4 vertices will receive colors from $\{1, 2, \ldots, c\}$,
- Group 5 vertices will receive colors from $\{c + 1, c + 2, \ldots, d\}$,
- Group 6 vertices will receive colors from $\{d + 1, d + 2, \ldots, e\}$,
- Group 7 vertices will receive colors from $\{e + 1, e + 2, \ldots, k\}$,
- Group 8 vertices will receive colors from $\{c + 1, c + 2, \ldots, k\}$, and
- Group 9 vertices will receive colors from $\{1, 2, \ldots, k\}$,

where $1 < a < b < c < d < e < k$. 

Colors will then correspond to non-overlapping time slots within time blocks in our timetable as follows:

- Colors $1, 2, \ldots, a$ correspond to $a$ non-overlapping MWF morning time slots (in any order),
- Colors $a + 1, a + 2, \ldots, b$ correspond to $b - a$ non-overlapping MWF afternoon time slots (in any order),
- Colors $b + 1, b + 2, \ldots, c$ correspond to $c - b$ non-overlapping MWF evening time slots (in any order),
- Colors $c + 1, c + 2, \ldots, d$ correspond to $d - c$ non-overlapping TR morning time slots (in any order),
- Colors $d + 1, d + 2, \ldots, e$ correspond to $e - d$ non-overlapping TR afternoon time slots (in any order), and
- Colors $e + 1, e + 2, \ldots, k$ correspond to $k - e$ non-overlapping TR evening time slots (in any order).

The colors $1, 2, \ldots, c$ will correspond to 1-hour time slots on Mondays, Wednesdays, and Fridays, and the colors $c + 1, c + 2, \ldots, k$ will correspond to $1\frac{1}{2}$-hour time slots on Tuesdays and Thursdays. Any of these 1-hour or $1\frac{1}{2}$-hour time slots may possibly be expanded to 3-hour time slots, however, to accommodate any 3-hour vertices/courses in Group 10 that have yet to be colored and/or scheduled.

If Group 10 is empty (i.e., there are no 1-time-per-week courses to schedule), then we are finished with the timetabling process; we will have assigned all courses to time slots. If Group 10 is nonempty, then we must now assign the 1-time-per-week courses represented by vertices from Group 10 to 3-hour time slots. Our method of assigning time slots to these courses will be determined by the choice of method used to color the "Group 10" vertices — the simple method, the time block method, the time slot method, or the 2-or-3-color method.

Suppose that we colored the vertices from Group 10 by the simple method or by
the *time block method*, as described in Sections 5.4.1.1 and 5.4.1.2, respectively. A vertex \( v_i \) from Group 10 corresponding to a course \( c_i \) will be assigned a 3-hour time slot according to the following color criteria:

- If \( v_i \) was colored a color from \( \{1, 2, \ldots, a\} \), then the MWF morning 1-hour time slot corresponding to that color will be expanded to 3 hours, and \( c_i \) will be assigned to that time slot.
- If \( v_i \) was colored a color from \( \{a + 1, a + 2, \ldots, b\} \), then the MWF afternoon 1-hour time slot corresponding to that color will be expanded to 3 hours, and \( c_i \) will be assigned to that time slot.
- If \( v_i \) was colored a color from \( \{b + 1, b + 2, \ldots, c\} \), then the MWF evening 1-hour time slot corresponding to that color will be expanded to 3 hours, and \( c_i \) will be assigned to that time slot.
- If \( v_i \) was colored a color from \( \{c + 1, c + 2, \ldots, d\} \), then the TR morning 1\(\frac{1}{2}\)-hour time slot corresponding to that color will be expanded to 3 hours, and \( c_i \) will be assigned to that time slot.
- If \( v_i \) was colored a color from \( \{d + 1, d + 2, \ldots, e\} \), then the TR afternoon 1\(\frac{1}{2}\)-hour time slot corresponding to that color will be expanded to 3 hours, and \( c_i \) will be assigned to that time slot.
- If \( v_i \) was colored a color from \( \{e + 1, e + 2, \ldots, k\} \), then the TR evening 1\(\frac{1}{2}\)-hour time slot corresponding to that color will be expanded to 3 hours, and \( c_i \) will be assigned to that time slot.

Once a time slot corresponding to a color from \( \{1, 2, \ldots, c\} \) has been expanded, any 1-hour courses represented by vertices that were colored with that color may be reassigned to any of the three 1-hour portions of that particular 3-hour time slot. For example, suppose that two courses — a MWF course \( c_i \) and a M course \( c_j \) — are represented by vertices \( v_i \) and \( v_j \), respectively, and \( v_i \) and \( v_j \) were both colored with color \( a \) in the conflict graph. Further suppose that color \( a \) initially cor-
responds to the 1-hour time slot “Monday/Wednesday/Friday, 8:00–9:00 a.m.” After expansion of this time slot to a 3-hour time slot (i.e., “Monday/Wednesday/Friday, 8:00–11:00 a.m.”), course $c_j$ will be scheduled for the 3-hour time slot “Monday, 8:00–11:00 a.m.”, and course $c_i$ can be rescheduled for any of the three 1-hour time slots “Monday/Wednesday/Friday, 8:00–9:00 a.m.”, “Monday/Wednesday/Friday, 9:00–10:00 a.m.”, or “Monday/Wednesday/Friday, 10:00–11:00 a.m.”. Along a similar line, once a time slot corresponding to a color from $\{c + 1, c + 2, \ldots, k\}$ has been expanded, any $1\frac{1}{2}$-hour courses represented by vertices that were colored that color may be reassigned to either of the two $1\frac{1}{2}$-hour portions of that 3-hour time slot.

This leads us to the following theoretical result:

**Theorem 3** Suppose $G$ is a course conflict graph, and $\chi(G) = k$. Suppose also that we have a proper $k$-coloring of $G$. Let $k = x + y + z$, where $x$ denotes the number of colors corresponding to MWF time slots that are to be expanded from 1-hour periods to 3-hour periods, $y$ denotes the number of colors corresponding to TR time slots that are to be expanded from $1\frac{1}{2}$-hour periods to 3-hour periods, and $z$ denotes the number of colors corresponding to time slots of either 1 hour or $1\frac{1}{2}$ hours that are not to be expanded. After expansion of necessary time slots (that is, after the timetabling of 1-time-per-week courses), the total number of hours each week used to timetable all of the courses represented by vertices in $G$ will have increased by $6x + 3y$.

**Proof of Theorem 3** Let $H$ denote the total number of hours each week needed to schedule all of the courses represented by vertices in $G$. Since $\chi(G) = x + y + z$ and each color class corresponds to either a “MWF time slot” (i.e., three 1-hour periods per week) or a “TR time slot” (i.e., two $1\frac{1}{2}$-hour periods per week), we have $H = 3\chi(G) = 3(x + y + z) = 3x + 3y + 3z$ hours before “time slot expansion”. During “time slot expansion”, $x$ of the MWF 1-hour time slots will each increase by 2 hours per day, or $3 \times 2 = 6$ hours per week, and $y$ of the TR $1\frac{1}{2}$-hour time slots will each increase by $1\frac{1}{2}$ hours per day, or $2 \times 1\frac{1}{2} = 3$ hours per week. The remaining $z$ 1-hour
and 1\frac{1}{2}-hour time slots will not increase in length. Hence, after “time slot expansion", $H$ will have increased by $6x + 3y$ hours (i.e., from $3x + 3y + 3z$ hours to $9x + 6y + 3z$ hours).

Now suppose that, instead of the simple method or time block method, we colored the vertices from Group 10 by the time slot method, as described in Section 5.4.1.3. In this case, we have assumed that there will only be one 3-hour time slot per time block per day, for a total of 15 different 3-hour time slots in which to schedule the courses represented by vertices in Group 10. Recall that with the time slot method, we assign actual time slots to courses meeting 3 times per week or 2 times per week before coloring any of the vertices corresponding to courses meeting 1 time per week. The start times for each of the 15 3-hour time slots for these courses have also been established before we color any of the “Group 10" vertices. In fact, we can choose, for example, to assign to each of the 15 3-hour time slots the color class from among those in its time block that contains the fewest number of vertices from Groups 1 through 9, so as to perhaps increase the likelihood that a particular “Group 10" vertex $v_i$ can be colored that color, and thus have the corresponding course $c_i$ able to be scheduled to that 3-hour slot. When we color the “Group 10" vertices, each vertex $v_i$ corresponding to a 1-time-per-week course will then only be colored with one of the colors corresponding to one of the 15 3-hour time slots. These 3-hour time slots will not overlap with each other, nor will they overlap with any of the 1-hour or 1\frac{1}{2}-hour time slots in the timetable that are represented by a different color than the color corresponding to a 3-hour time slot. Also, any 1-hour courses that were colored with the same color as that of one of the 15 designated 3-hour time slots may be reassigned to any of the three 1-hour portions of that particular 3-hour time slot, and any 1\frac{1}{2}-hour courses that were colored with the same color as that of a 3-hour time slot may be reassigned to either of the two 1\frac{1}{2}-hour portions of that 3-hour time slot, as was the case with the simple method or the time block method.

Now suppose that, instead of the simple method, time block method, or time
slot method, we colored the vertices from Group 10 by the 2-or-3-color method, as described in Section 5.4.1.4. In this case, a vertex $v_i$ corresponding to a course $c_i$ meeting 1 time per week will either be colored with three colors, if is to meet on either Monday, Wednesday, or Friday, or colored with two colors if it is to meet on Tuesday or Thursday. Unlike the simple method, time block method and time slot method described above, no expansion of time slots is necessary with the 2-or-3-color method, as 3-hour time slots on Monday, Wednesday, and Friday will coincide with three consecutive MWF 1-hour time slots, and 3-hour time slots on Tuesday and Thursday will coincide with two consecutive TR 1$\frac{1}{2}$-hour time slots. A 1-time-per week course $c_i$ will be assigned to a time slot based on the color assigned to its corresponding vertex $v_i$ as follows:

- If the color of $v_i =$ "a-b-c-Monday", then $c_i$ will meet on Monday during the 3-hour time slot corresponding to the three consecutive 1-hour time periods represented by colors $a$, $b$, and $c$.
- If the color of $v_i =$ "a-b-Tuesday", then $c_i$ will meet on Tuesday during the 3-hour time slot corresponding to the two consecutive 1$\frac{1}{2}$-hour time periods represented by colors $a$ and $b$.
- If the color of $v_i =$ "a-b-c-Wednesday", then $c_i$ will meet on Wednesday during the 3-hour time slot corresponding to the three consecutive 1-hour time periods represented by colors $a$, $b$, and $c$.
- If the color of $v_i =$ "a-b-Thursday", then $c_i$ will meet on Thursday during the 3-hour time slot corresponding to the two consecutive 1$\frac{1}{2}$-hour time periods represented by colors $a$ and $b$.
- If the color of $v_i =$ "a-b-c-Friday", then $c_i$ will meet on Friday during the 3-hour time slot corresponding to the three consecutive 1-hour time periods represented by colors $a$, $b$, and $c$.

Note: When scheduling 1-time-per week courses, it may also be preferable to
distribute them as evenly as possible throughout the week. For example, if we have \( j \) 1-time-per-week courses to schedule, we may wish to be able to schedule as close to \( \frac{j}{5} \) of these courses on each of Monday, Tuesday, Wednesday, Thursday, and Friday as possible. Of course, it is unlikely that a perfect (or almost perfect) \( \frac{j}{5}, \frac{j}{5}, \frac{j}{5}, \frac{j}{5}, \frac{j}{5} \) split will be possible. However, by employing a variation of the greedy coloring algorithm with an intelligent color search procedure (such as SMALLEST-FIRST-SEARCH GREEDY, for example, which should tend to balance the size of the color classes), we can optimistically attempt to get close. An overall even distribution of courses among time slots, as mentioned earlier, will likely aid in assigning courses to classrooms, and will also allow for a maximum utilization of university resources.

5.6 Assigning Courses to Classrooms

Once we have constructed our conflict-free timetable of courses, the task remains of assigning each of the courses an appropriate classroom in which to meet. Recall from Section 5.1 some of the essential and preferential timetabling conditions with regard to room assignment:

1. Two or more courses scheduled for the same time slot cannot be assigned the same classroom (this is an essential timetabling condition).

2. Each course must be scheduled in an available classroom that can accommodate its size (this is an essential timetabling condition).

3. Specific outlined room requirements/type for a particular course, if any (for example, a laboratory workspace, or a computer workstation with projection screen) should be taken into account (this is an essential timetabling condition).

4. An instructor may have a specific room request for a course, beyond the scope of the outlined room requirements specified above (this is an example illustrat-
ing the sometimes “fine line” between an essential and preferential timetabling condition).

5. Each course must be scheduled for exactly one room, to remain constant throughout the scheduling period (this is listed in Section 5.1 as an essential timetabling condition, though there are known cases where this condition has been “relaxed” and could be classified as otherwise “preferential”).

6. It might be preferable to assign each course to a classroom that is located in or close to the building in which that course’s department or school is based, and/or close to the office of the course’s instructor (this is a preferential timetabling condition).

7. Classrooms should be just large enough to hold the courses in them, in order to eliminate the presence of unused empty space (this is a preferential timetabling condition).

8. Minimize the number of classrooms used or needed when scheduling the courses (this is a preferential timetabling condition).

In Section 5.6.1 below, we discuss the assigning of courses to classrooms that takes place during the coloring of a “Method B” conflict graph. We follow in Section 5.6.2 with the presentation of a separate room assignment algorithm that assigns already-timetabled courses to classrooms, after the coloring of a “Method A” conflict graph has taken place.

5.6.1 Room Assignment from a “Method B” Conflict Graph

Recall that when properly coloring a “Method B” conflict graph, exactly one vertex in each “bag” of vertices representing a single course is assigned a color. That is to say, each “course” represented in the graph is assigned a color. Each color in a properly-colored “Method B” conflict graph corresponds to a different time slot. Furthermore,
coloring such a conflict graph assigns each course to an appropriate classroom as well. A colored vertex \( v_i r_k \) implies that course \( c_i \) has been assigned to room \( r_k \).

Our construction of a "Method B" conflict graph, as described in Section 5.3.2, ensures that when the graph has been colored and both time slots and classrooms have been assigned, timetabling condition 1 above will always be satisfied. Edges between identical pairs of room vertices \( v_i r_k \) and \( v_j r_k \) guarantee that if courses \( c_i \) and \( c_j \) are assigned the same slot (i.e., their corresponding colored vertices are colored the same color), they will not be assigned the same classroom. Furthermore, since each "bag" of vertices representing a single course contains only vertices corresponding to rooms of adequate size and type for that course, it follows that timetabling conditions 2 and 3 (and when applicable, timetabling condition 4) will also be satisfied. Each course will be assigned to a classroom that can accommodate its expected enrollment or maximum enrollment (as noted by an entry in the optional course data field CLASS_SIZE or CLASS_MAX_SIZE, respectively) and course type (as noted by an entry in the optional course data field ROOM_TYPE (for example, "lecture", "seminar", or "laboratory")), as well as any other specific room requests, within reason. In addition, since only one vertex in each "bag" of vertices representing a single course is colored, it follows that timetabling condition 5 above will necessarily be satisfied. Each course will be assigned to one and only one classroom.

We can also attempt to satisfy preferential timetabling conditions 6, 7, and 8 above by modifying or restricting our selection of vertices to make up each "bag" of vertices representing each course in our "Method B" conflict graph. For example, if we wish to try to assign a particular course \( c_i \) to a classroom that is located in or close to the building in which that course's department or school is based, and/or close to the office of the course's instructor (timetabling condition 6), then only those room vertices corresponding to such classrooms will be selected as candidates for vertices in the "bag" of vertices corresponding to course \( c_i \). If we wish to try to assign a particular course \( c_i \) to a classroom just large enough to hold it, in order to
eliminate the presence of unused empty space (timetabling condition 7), then only those room vertices corresponding to classrooms of adequate size will be selected as candidates for vertices in the "bag" of vertices corresponding to course \( c_i \). If we wish to try to minimize the number of classrooms used or needed when scheduling the courses (timetabling condition 8), we can subsequently limit the number of different classrooms that make up the different "bags" of vertices in our conflict graph.

### 5.6.2 Room Assignment from a "Method A" Conflict Graph

Recall that when properly coloring a "Method A" conflict graph, each vertex \( v_i \) representing a single course \( c_i \) is assigned a color. Each color in a properly-colored "Method A" conflict graph corresponds to a different time slot. Unlike coloring a "Method B" conflict graph, coloring a "Method A" conflict graph only partitions the courses into a set of independent color classes from which time slots can be assigned; it does not also assign each course to an appropriate classroom. Therefore, a separate method for room assignment is required when timetabling courses using a "Method A" conflict graph.

We present a number of simple algorithms that can be used to assign already-timetabled courses to classrooms, after the coloring of a "Method A" conflict graph has taken place. These algorithms are loosely based on the concept of "bin packing". The bin packing problem is stated as follows: Given a finite set \( U = \{u_1, u_2, \ldots, u_n\} \) of "items", a positive integer "size" \( s(u) \in Z^+ \) for each item \( u \in U \), a positive integer bin capacity \( B \), and a positive integer \( k \), find a partition of \( U \) into \( k \) disjoint subsets \( U_1, U_2, \ldots, U_k \) such that the sum of the sizes of the items in each \( U_i \) is no more than \( B \). Each subset \( U_i \) is viewed as specifying a set of items to be placed in a single "bin" of capacity \( B \). The bin packing problem is NP-complete, though several simple approximation algorithms exist for solving it, including "First Fit", "Best Fit", "First Fit Decreasing", and "Best Fit Decreasing". The First Fit bin packing algorithm places the items into the bins, one at a time in order of increasing
index. It does so according to the following rule: always place the next item \( u_i \) into the lowest-indexed bin for which the sum of the sizes of the items in that bin does not exceed \( B - s(u_i) \). In other words, \( u_i \) is always placed into the first bin in which it will fit (i.e., without exceeding the bin capacity). The Best Fit bin packing algorithm is a modification of the First Fit algorithm that uses the more sophisticated placement rule: always place the next item \( u_i \) in that bin which has current contents closest to, but not exceeding, \( B - s(u_i) \), choosing the bin with the lowest index in case of ties. The First Fit Decreasing bin packing algorithm is yet another modification of the First Fit algorithm that first sorts the items from \( U \) by size and reindexes them so that \( s(u_1) \geq s(u_2) \geq \cdots \geq s(u_n) \). The Best Fit Decreasing bin packing algorithm incorporates the placement rule of the Best Fit algorithm along with the item ordering rule present in the First Fit Decreasing algorithm. [32].

The problem of assigning already-timetabled courses to classrooms can be viewed as a special case of the above bin packing problem. The "items" to pack are the \( n \) courses \( u_1, u_2, \ldots, u_n \), and the "bins" are the \( m \) available classrooms \( r_1, r_2, \ldots, r_m \). The "size" \( s(c_i) \) of course \( c_i \) is the expected enrollment or maximum enrollment of \( c_i \) (as noted by an entry, for example, in the course data field CLASS_SIZE or CLASS_MAX_SIZE). Each classroom can hold at most one course, so there will be at most one "item" in each "bin". Furthermore, each "bin" (i.e., each room \( r_j \)) has its own size or capacity, denoted \( s(r_j) \).

We therefore present the following two simple room assignment algorithm variations based on bin packing: 1) a FIRST FIT DECREASING room assignment (FFDra) and 2) a BEST FIT DECREASING room assignment (BFDra). For each time slot, each course scheduled during that time slot is ultimately assigned to a single room for that course. Both algorithm variations assume that we have a list of courses that have already been timetabled for each time slot, along with the expected enrollment or maximum enrollment of each course. The algorithm variations also assume that we have a complete list of available classrooms and their capacities for each time
slot, and that for each time slot, the number of courses does not exceed the number of available classrooms.

For each time slot in our timetable, we can assign that time slot's courses to classrooms as follows:

First, let \( n \) denote the number of courses that have been assigned to that time slot, and let \( m \) denote the number of rooms that are available during that time slot. Again, we assume that \( n \leq m \). Next, list the \( n \) courses and \( m \) rooms in size order, largest first, so that \( s(c_1) \geq s(c_2) \geq \cdots \geq s(c_n) \) and \( s(r_1) \geq s(r_2) \geq \cdots \geq s(r_m) \). We assume that the largest-sized available course can fit in the largest-sized available room (i.e., \( s(c_n) \leq s(r_m) \)), and that the smallest-sized available course can fit in the smallest-sized available room (i.e., \( s(c_1) \leq s(r_1) \)). Then, starting with the largest course, assign each course to a single classroom, one at a time, according to one of the following placement rules:

**FIRST FIT DECREASING room assignment (FFDra):** Always assign the next course \( c_i \) to the lowest-indexed unoccupied room \( r_j \) for which \( s(c_i) \leq s(r_j) \). In other words, \( c_i \) is always assigned to the *first* classroom in which it will fit (i.e., not exceed the room capacity).

**OR**

**BEST FIT DECREASING room assignment (BFDra):** Always assign the next course \( c_i \) to the unoccupied room \( r_j \) which minimizes \( s(r_k) - s(c_i) \), with \( s(r_k) - s(c_i) \geq 0 \), over all unoccupied rooms \( r_k \) from \( r_1, r_2, \ldots, r_m \), choosing the unoccupied room with the lowest index in case of ties. In other words \( c_i \) is always assigned to the classroom in which it will *best* fit (i.e., leave the minimum amount of unused empty space).

To illustrate each of these room assignment algorithm variations we provide the
following simple example. Note: In the figure below, we denote the size \( s(c) \) of an
course \( c \) by \( c^* \). Suppose that we have a course timetable that has scheduled during
the same time slot the three courses ENGL101, SPAN201, and COMP412 with sizes 96, 75, and 37, respectively. Further suppose that there are four available classrooms
to which these three courses may be assigned: room \( r_1 \) (with capacity 100), room \( r_2 \)
(with capacity 80), room \( r_3 \) (with capacity 60), room \( r_4 \) (with capacity 40) and room
\( r_5 \) (with capacity 20). Figure 5.12 (a) shows the results of assigning these courses to
these rooms using a FIRST FIT DECREASING room assignment, while Figure 5.12
(b) shows the results of assigning these courses to these rooms using a BEST FIT
DECREASING room assignment.

<table>
<thead>
<tr>
<th>Room (size)</th>
<th>(a): FFDra</th>
<th>(b) : BFDra</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r_4 ) (100)</td>
<td>ENGL101\textsuperscript{96}</td>
<td>ENGL101\textsuperscript{96}</td>
</tr>
<tr>
<td>( r_3 ) (80)</td>
<td>SPAN201\textsuperscript{75}</td>
<td>SPAN201\textsuperscript{75}</td>
</tr>
<tr>
<td>( r_2 ) (60)</td>
<td>COMP412\textsuperscript{37}</td>
<td></td>
</tr>
<tr>
<td>( r_1 ) (40)</td>
<td></td>
<td>COMP412\textsuperscript{37}</td>
</tr>
</tbody>
</table>

Figure 5.12: Course room assignment: FFDra and BFDra

By using either a FIRST FIT DECREASING room assignment or a BEST FIT
DECREASING room assignment, timetabling conditions 1 and 2 listed at the be-
inning on Section 5.6 will always be satisfied. Two or more courses scheduled for
the same time slot will not be assigned to the same room, and each course will be
assigned to an available room that can accommodate its size. Furthermore, to satisfy
timetabling condition 3, we can, for each time slot, partition the list of courses by
type (as noted by an entry in the course data field ROOM\_TYPE) and perform a sepa-
rate room assignment on each sublist of courses (for example, assign lecture courses
to lecture rooms, then laboratory courses to laboratory rooms, and then seminars
to seminar rooms). In the event of specific room requests (timetabling condition 4), we can choose to preassign certain courses to specific rooms, before performing our FIRST FIT DECREASING room assignment or BEST FIT DECREASING room assignment on the remaining courses, in an attempt to honor such requests. Each course will be assigned to exactly one room, so timetabling condition 5 will always be satisfied.

To attempt to satisfy preferential timetabling condition 6, we can, for each room type, partition the list of courses by department or school or instructor (as can be determined by the course data field COURSE.ID or the course data field INSTRUCTOR) and partition the list of available rooms by building, and perform a separate room assignment on each sublist of courses so as to only assign courses to classrooms that are located in or close to the building in which that course’s department or school is based, and/or close to the office of the course’s instructor. If we wish to attempt to satisfy preferential timetabling condition 7, to assign courses to classrooms just large enough to hold them, in order to minimize the presence of unused empty space, we can favor a BEST FIT DECREASING room assignment over a FIRST FIT DECREASING room assignment, since the former method will specifically seek to accomplish this goal. If we wish to try to minimize the number of classrooms used or needed when scheduling the courses (preferential timetabling condition 8), we can subsequently limit the number of different classrooms that make up our list of available classrooms to which courses can be assigned.

In order to ensure our assumption above that for each time slot the largest-sized available course can fit in the largest-sized available room, we may wish or need to preschedule sufficiently large courses to time slots and preassign them to rooms before beginning the timetabling and room assignment process. We can set a size threshold parameter $mathcal{Q}$ such that, for a course $c_i$, if $s(c_i) \geq Q$, we will preschedule course $c_i$ in this manner, before timetabling and assigning rooms for the rest of the courses.
5.7 Some Final Notes on Final Exam Timetabling

5.7.1 From Course Timetable to Final Exam Timetable

Once we have successfully constructed a satisfactory and acceptable conflict-free timetable of courses, and each course has been assigned an appropriate time slot at which to meet, and classroom in which to meet during the semester, it can be relatively easy to transform this course timetable to a final exam timetable or schedule for the end of the semester. As mentioned in Section 4.3, the University of St. Thomas uses a semester's course timetable to directly schedule final exams for that same semester. We can therefore use our newly-constructed conflict-free timetable of courses to easily construct a final exam timetable. All courses that were scheduled for the same 3-time-per-week, 2-time-per-week, or 1-time-per-week time slot during the semester will have their final exams scheduled for the same fixed-length final exam time slot. Also, each final exam will be held in the same classroom in which the course giving that exam was held during the semester, in order to avoid time and room conflicts for students taking final exams. We naturally assume that each final exam to be scheduled at the end of a particular semester corresponds to exactly one course offered during that same semester. However, not all courses will necessarily have scheduled final exams. A course may have a self-scheduled final exam (such as is often the case at an institution such as Rice University), or it may have a take-home exam, or it may not have a final exam at all, but rather a end-of-semester term paper or project to complete. Based on this assumption, if \( C \) denotes the number of courses to be scheduled during a given semester, and \( F \) denotes the number of final exams to be scheduled at the end of that same semester, then we have \( F \leq C \).

A final exam schedule at a college or university usually consists of a number of fixed-length time slots (usually 2 or 3 hours in length) lasting over a period of several days, perhaps even as long as a week or two weeks. Since we are constructing our final exam timetable (and final exam room assignment) based directly on our conflict-
free course timetable (and course room assignment), our final exam timetable will be conflict-free as well for both instructors and students, and all exams will take place in rooms that can accommodate their sizes.

5.7.2 Minimizing Consecutive Exams via Integer Programming

Because all time slots during a final exam schedule are of fixed length, because all courses scheduled for the same 3-time-per-week, 2-time-per-week, or 1-time-per-week time slot during the semester are scheduled to hold their exams during the same fixed-length final exam time slot, and because all final exams are held in the same classroom in which the course giving that exam was held during the semester, we can arrange the final exam schedule time slots in any order that we wish, and the final exam schedule will remain conflict-free. Why might we want to arrange the final exam schedule time slots in a particular order? Suppose, for example, that we wish to minimize the number of occurrences of students having to take “two finals in a row” (i.e., take a final exam in each of two consecutive final exam time slots, with either both exams during the same day, or the first at the end of one day and the second at the beginning of the next). Such a request to minimize consecutive final exams for students is a rather common and often desired one [13, among others].

We propose an integer programming (IP) formulation which can be used to determine an “optimal” ordering of the final exam schedule time slots, based on such a desire as described above.

Suppose that we have $p$ time slots $t_1, t_2, \ldots, t_p$ to be ordered. Our IP formulation contains $p$ integer variables $x_1, x_2, \ldots, x_p$, one for each time slot $t_i$. We will assign each variable $x_i$ a different number from $\{1, 2, \ldots, p\}$; this will determine the chronological order of the $p$ time slots. An optimal ordering is one which minimizes the number of occurrences of students taking final exams in consecutive time slots.

For each pair of time slots $t_i$ and $t_j$, let $c_{ij}$ denote the number of students taking
an exam for a course that met during time slot \( t_i \) during the semester and an exam for a course that met during time slot \( t_j \) during the semester. We will thus have \( pC_2 \) of these \( c_{ij} \) values to compute.

The IP formulation is as follows:

\[
S = \sum_{i=1}^{p-1} \sum_{j=i+1}^{p} c_{ij} y_{ij}
\]

subject to

1. \( x_i = a \), for each index \( i \) and number \( a \); \( i = 1, 2, \ldots, p; \ a = 1, 2, \ldots, p \)

2. \( 1 \leq x_i \leq p; \ i = 1, 2, \ldots, p \)

3. \( 1 \leq |x_i - x_j| \leq p - 1; \ i = 1, 2, \ldots, p; \ i \neq j \)

4. \( |x_i - x_j| z_{ij} = y_{ij}; \ i = 1, 2, \ldots, p - 1; \ j = i + 1, i + 2, \ldots, p \)

5. \( |x_i - x_j| + z_{ij} > 1; \ i = 1, 2, \ldots, p - 1; \ j = i + 1, i + 2, \ldots, p \)

6. \( \sum_{i=1}^{p-1} \sum_{j=i+1}^{p} y_{ij} = p - 1 \)

7. \( x_i \) integer; \( i = 1, 2, \ldots, p \)

8. \( y_{ij} \) binary; \( i = 1, 2, \ldots, p - 1; \ j = i + 1, i + 2, \ldots, p \)

9. \( z_{ij} \) binary; \( i = 1, 2, \ldots, p - 1; \ j = i + 1, i + 2, \ldots, p \)

An optimal solution \( \{x_1, x_2, \ldots, x_p\} \) to the above integer program is one which assigns each variable \( x_i \) a different number from \( \{1, 2, \ldots, p\} \) subject to the above constraints, such that the number of occurrences of students taking consecutive final exams (as calculated by the double sum \( \sum_{i=1}^{p-1} \sum_{j=i+1}^{p} c_{ij} y_{ij} \)) is minimized. Each variable \( x_i \) corresponding to a time slot \( t_i \) must be assigned a unique number from \( \{1, 2, \ldots, p\} \), and thus each number from \( \{1, 2, \ldots, p\} \) will be the value assigned to one of the variables \( x_1, x_2, \ldots, x_p \). This assignment determines the ordering of the
time slots in the final exam schedule: if \( x_i = 1 \), then time slot \( t_i \) will take place first in the final exam schedule, followed by time slot \( t_j \), where \( x_j = 2 \), and so forth. The last time slot \( t_k \) will have \( x_k = p \).

Constraint 1 above maintains that each element of the set \( \{1, 2, \ldots, p\} \) must be the value of exactly one of the integer variables \( x_1, x_2, \ldots, x_p \) (i.e., each place in the final exam timetable, from 1st to \( p^{th} \), will be occupied by a single final exam time slot \( t_i \)). Constraint 2 is a variable bound constraint that states that each variable \( x_i \) must have a value between 1 and \( p \) (i.e., each final exam time slot \( t_i \) must have a place in the final exam timetable, from 1st to \( p^{th} \)). Constraint 3 maintains that no two final exam time slots \( t_i \) and \( t_j \) may be assigned the same place in the final exam timetable, and that \( t_i \) and \( t_j \) will be no more than \( p - 1 \) places apart from each other in the final exam timetable. Constraint 4 and 5 (together with constraints 8 and 9) guarantee that if \( |x_i - x_j| = 1 \), then \( y_{ij} = 1 \) and \( z_{ij} = 1 \), and if \( |x_i - x_j| \neq 1 \), then \( y_{ij} = 0 \) and \( z_{ij} = 0 \). Constraint 6 ensures that exactly \( p - 1 \) of the \( pC_2 y_{ij} \) binary variables will be equal to 1, while the rest will be equal to 0. A binary variable \( y_{ij} \) equal to 1 corresponds to a pair of consecutive final exam time slots \( t_i \) and \( t_j \) in the final exam timetable. Constraint 7 reflects the fact that each of the values of \( x_1, x_2, \ldots, x_p \) is an integer, Constraint 8 restricts each of the \( pC_2 y_{ij} \) variables to be equal to either 0 or 1 (i.e., binary), and constraint 9 places the same restriction on each of the \( pC_2 z_{ij} \) variables. The objective function calculates the number \( S \) of occurrences of students taking consecutive final exams (i.e., taking final exams in consecutive time slots). Again, an optimal solution \( \{x_1, x_2, \ldots, x_p\} \) to the above IP corresponds to an ordering of the set of final exam time slots \( \{t_1, t_2, \ldots, t_p\} \) so as to make \( S \) as small as possible, subject to constraints 1 through 9.

### 5.7.3 Packing Multiple Exams in a Single Classroom

Recall that in the final exam timetabling model discussed above, all courses that were scheduled for the same time slot during the semester have their final exams scheduled
for the same fixed-length final exam time slot, and each final exam is assigned to the same classroom in which the course giving that exam was held during the semester. Since no two courses were held in the same room during the same time slot during the semester, it follows that no two final exams held during the same final exam time slot are assigned to the same classroom. Each classroom will contain at most one final exam during each final exam time slot.

Suppose that we would like to reduce the number of classrooms that are used during the final exam period, by allowing for multiple exams to occupy a particular room at the same time, subject to room capacity constraints, of course. We may, for example, have a large-sized lecture course $c_i$ of 150 students that meets in room $r_k$ during the “Monday/Wednesday/Friday, 9:00–10:00 a.m.” time slot” during the semester, but does not have a final exam to be scheduled. Under the final exam timetabling model discussed above, room $r_k$ would not be used during the final exam time slot corresponding to the “Monday/Wednesday/Friday, 9:00–10:00 a.m.” time slot. Suppose now that we have five smaller-sized lecture courses $c_v, c_w, c_x, c_y,$ and $c_z$ of 30 students each that meet during the same “Monday/Wednesday/Friday, 9:00–10:00 a.m.” time slot during the semester, and offer final exams $e_v, e_w, e_x, e_y,$ and $e_z$, respectively, to their students. We could reassign final exams $e_v, e_w, e_x, e_y,$ and $e_z$ all to room $r_k$, and reduce the number of classrooms used for these five final exams from five to one.

We present the following room assignment algorithm that reassigns final exams to classrooms, once an initial final exam timetable has been transformed from the corresponding course timetable, as described in Section 5.7.1. The algorithm is patterned after an algorithm by Burke, Elliman, and Weare [13] that focuses on accommodating a preference for larger and fewer rooms during the exam timetabling process.

For each time slot in the final exam period, each course scheduled to give a final exam during that time slot is ultimately assigned to a single room for that exam, allowing for multiple exams to be assigned to the same room at the same time. Some
exams may be reassigned to new rooms, while others may remain in their originally scheduled rooms. The algorithm assumes that we have a list of final exams that have already been timetabled for each time slot in the final exam period, along with the number of students taking each exam. The algorithm also assumes that we have a complete list of available classrooms and their capacities for each final exam time slot, as well as an initial room assignment for each exam, as obtained directly from our course timetable. Since we are not reassigning exams to time slots, but rather only reassigning exams to rooms within a particular time slot, and since the total number of final exams \( F \) scheduled does not exceed the total number of courses \( C \), we can be assured of having enough classroom space to which we can reassign some or all of the exams within each final exam time slot.

For each time slot in the final exam period, we reassign that time slot’s final exams to classrooms as follows:

First, let \( n \) denote the number of final exams that have been assigned to that time slot, and let \( m \) denote the number of rooms that are available to hold exams during that time slot. Again, we have \( n \leq m \), since \( F \leq C \). Next, list the \( n \) exams and \( m \) rooms in size order, smallest first, as \( e_1, e_2, \ldots, e_n \) and \( r_1, r_2, \ldots, r_m \), respectively. By the “size” of an exam \( e_i \) (denoted by \( |e_i| \)) we mean the number of students taking the exam, and by the “size” of a room \( r_i \) (denoted by \( |r_i| \)) we mean the number of students that can fit in that room (i.e., the capacity of the room). We thus have \( |e_1| \leq |e_2| \leq \cdots \leq |e_n| \) and \( |r_1| \leq |r_2| \leq \cdots \leq |r_m| \). We can safely assume that the largest-sized exam, \( e_n \), can fit in the largest-sized available room, \( r_m \). In other words, we have \( |e_n| \leq |r_m| \). Also, we can safely assume that the smallest-sized exam, \( e_1 \), can fit in the smallest-sized room \( r_1 \). In other words, \( |e_1| \leq |r_1| \).

---

**Step 1:** Set \( i \leftarrow 1 \) and \( j \leftarrow 1 \).

**Step 2:** If exam \( e_i \) is not in room \( r_j \), then continue from Step 5.

**Step 3:** Set \( k \leftarrow j + 1 \).
Step 4: If there is space in room $r_k$ for exam $e_i$, then move exam $e_i$ from room $r_j$ to room $r_k$. Otherwise, set $k ← k + 1$, and if $k ≤ m$, then repeat Step 4, else set $j ← j + 1$ and $i ← 1$, and if $j ≠ m$, then repeat from Step 2. If $j = m$, then STOP.

Step 5: Set $i ← i + 1$. If $i = n$, then set $j ← j + 1$ and $i ← 1$. If $j = m$, then STOP, otherwise repeat from Step 2.

The algorithm above works by first ignoring the empty rooms, then starting with the smallest exam in the smallest room, going up the list until it can be placed in another room, continuing until no movement is possible.

To further explain this algorithm, we provide the following simple examples. Note: In the figures below, we denote the size $s$ of an exam $e$ by $e^s$. As a first example, as illustrated below in Figure 5.13, suppose that we have a course timetable (Figure 5.13 (a)) that has scheduled during the same time slot the four courses GOVT509, HIST417, MATH428, and BIOL327 with sizes 36, 50, 54, and 69, respectively, to rooms $r_1$ (having capacity 40), $r_4$ (having capacity 100), $r_2$ (having capacity 60), and $r_3$ (having capacity 80), respectively. Further suppose that each of these four courses is giving a final exam to its students. Our initial final exam timetable (Figure 5.13 (b)) will have each of the exams given by these courses in the same rooms in which the courses were held during the semester. However, after we run the above algorithm, which attempts to reassign exams to courses in such a way as to accommodate a preference for larger and fewer rooms to be used, as well as allow multiple exams in a single room at the same time, we see that the GOVT509 exam can be moved up to share room $r_4$ with the HIST417 exam (Figure 5.13 (c)).

As a second example, as illustrated below in Figure 5.14, suppose that we have a course timetable (Figure 5.14 (a)) that has scheduled during the same time slot the four courses COMP412, ARTS333, SPAN201, and ENGL101 with sizes 37, 42, 75, and 96, respectively, to rooms $r_1$ (having capacity 40), $r_2$ (having capacity 60),
<table>
<thead>
<tr>
<th>Room (size)</th>
<th>(a)</th>
<th>(b)</th>
<th>(c)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_4$ (100)</td>
<td>HIST417$^{50}$</td>
<td>HIST417$^{50}$</td>
<td>GOVT509$^{36}$, HIST417$^{50}$</td>
</tr>
<tr>
<td>$r_3$ (80)</td>
<td>BIOL327$^{69}$</td>
<td>BIOL327$^{69}$</td>
<td>BIOL327$^{69}$</td>
</tr>
<tr>
<td>$r_2$ (60)</td>
<td>MATH428$^{54}$</td>
<td>MATH428$^{54}$</td>
<td>MATH428$^{54}$</td>
</tr>
<tr>
<td>$r_1$ (40)</td>
<td>GOVT509$^{36}$</td>
<td>GOVT509$^{36}$</td>
<td></td>
</tr>
</tbody>
</table>

Figure 5.13: Exam room reassignment: fewer exam rooms used

$r_3$ (having capacity 80), and $r_4$ (having capacity 100), respectively. Further suppose that only three of these courses, namely COMP412, SPAN201, and ENGL101, are giving final exams to their students. Our initial final exam timetable (Figure 5.14 (b)) will have the exams for COMP412, SPAN201, and ENGL101 in the same rooms in which these courses were held during the semester. However, after we run our exam room reassignment algorithm, we see that the COMP412 exam can be moved up into a larger room, namely room $r_2$, which was vacated due to the absence of an exam for ARTS333 (Figure 5.14 (c)).

<table>
<thead>
<tr>
<th>Room (size)</th>
<th>(a)</th>
<th>(b)</th>
<th>(c)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_4$ (100)</td>
<td>ENGL101$^{96}$</td>
<td>ENGL101$^{96}$</td>
<td>ENGL101$^{96}$</td>
</tr>
<tr>
<td>$r_3$ (80)</td>
<td>SPAN201$^{75}$</td>
<td>SPAN201$^{75}$</td>
<td>SPAN201$^{75}$</td>
</tr>
<tr>
<td>$r_2$ (60)</td>
<td>ARTS333$^{42}$</td>
<td></td>
<td>COMP412$^{37}$</td>
</tr>
<tr>
<td>$r_1$ (40)</td>
<td>COMP412$^{37}$</td>
<td>COMP412$^{37}$</td>
<td></td>
</tr>
</tbody>
</table>

Figure 5.14: Exam room reassignment: larger exam rooms used

Alternatively, one of the bin packing algorithm variations described in Section 5.6.2 may instead be used to reassign exams to classrooms in such a way as to allow for multiple exams to occupy a particular room at the same time, while accommodating a preference for larger and fewer rooms during the exam timetabling process.
Chapter 6

Computational Results

6.1 Rice University

6.1.1 Rice University, Fall 2000

Using course data provided by Rice University for the Fall 2000 semester, together with aspects of our new timetabling-by-graph-coloring model, we constructed a “Method A” course conflict graph $G$ with 749 vertices (one vertex for each of the 749 courses to be scheduled) and 147,440 edges (reflecting potential timetabling conflicts due to either essential or preferential timetabling conditions, as described in Section 5.3.1), and conducted computational experiments involving each of the 16 greedy coloring variations that we implemented:

- SIMPLE-SEARCH GREEDY: SPECIFIED ORDER (Figure 6.1);
- SIMPLE-SEARCH GREEDY: DECREASING DEGREE (Figure 6.2);
- SIMPLE-SEARCH GREEDY: INCREASING DEGREE (Figure 6.3);
- SIMPLE-SEARCH GREEDY: RANDOM ORDER (Figure 6.4);
- LARGEST-FIRST-SEARCH GREEDY: SPECIFIED ORDER (Figure 6.5);
- LARGEST-FIRST-SEARCH GREEDY: DECREASING DEGREE (Figure 6.6);
LARGEST-FIRST-SEARCH GREEDY: INCREASING DEGREE (Figure 6.7);
LARGEST-FIRST-SEARCH GREEDY: RANDOM ORDER (Figure 6.8);
SMALLEST-FIRST-SEARCH GREEDY: SPECIFIED ORDER (Figure 6.9);
SMALLEST-FIRST-SEARCH GREEDY: DECREASING DEGREE (Figure 6.10);
SMALLEST-FIRST-SEARCH GREEDY: INCREASING DEGREE (Figure 6.11);
SMALLEST-FIRST-SEARCH GREEDY: RANDOM ORDER (Figure 6.12);
RANDOM-SEARCH GREEDY: SPECIFIED ORDER (Figure 6.13);
RANDOM-SEARCH GREEDY: DECREASING DEGREE (Figure 6.14);
RANDOM-SEARCH GREEDY: INCREASING DEGREE (Figure 6.15);
and RANDOM-SEARCH GREEDY: RANDOM ORDER (Figure 6.16),

as well as a constraint programming graph coloring formulation (749 variables, 147,440 constraints) compatible with the OPL programming environment and OPLStudio (Figure 6.17).

Each of the following bar charts in Figures 6.1 through 6.17 below provides the number of colors used to properly color the course conflict graph, together with the number of vertices (i.e., courses) in each color class). The computing time taken to perform and verify each coloring is also given in the caption of each figure. Our computations using OPLStudio prove that the chromatic number of $G$ is in fact equal to 11; attempts to properly color $G$ with 10 colors or less result in OPLStudio declaring that "no solution exists". Therefore $\chi(G) = 11$, meaning that one would need a minimum of 11 colors to properly color $G$. Ten colors would not suffice.

All computations were performed on a Sun Enterprise 220R workstation with two 450-MHz UltraSPARC II cpu's, 2 Gb memory and 2 Gb swap space (harvey.caam.rice.edu).
Figure 6.1: Rice University, Fall 2000 (SSG:SO, Coloring time: 0.04 sec)

Figure 6.2: Rice University, Fall 2000 (SSG:DD, Coloring time: 0.06 sec)
Figure 6.3: Rice University, Fall 2000 (SSG:ID, Coloring time: 0.07 sec)

Figure 6.4: Rice University, Fall 2000 (SSG:RO, Coloring time: 0.09 sec)
Figure 6.5: Rice University, Fall 2000 (LFSG:SO, Coloring time: 0.03 sec)

Figure 6.6: Rice University, Fall 2000 (LFSG:DD, Coloring time: 0.07 sec)
Figure 6.7: Rice University, Fall 2000 (LFSG:ID, Coloring time: 0.07 sec)

Figure 6.8: Rice University, Fall 2000 (LFSG:RO, Coloring time: 0.04 sec)
Figure 6.9: Rice University, Fall 2000 (SFSG:SO, Coloring time: 0.04 sec)

Figure 6.10: Rice University, Fall 2000 (SFSG:DD, Coloring time: 0.07 sec)
Figure 6.11: Rice University, Fall 2000 (SFSG:ID, Coloring time: 0.07 sec)

Figure 6.12: Rice University, Fall 2000 (SFSG:RO, Coloring time: 0.04 sec)
Figure 6.13: Rice University, Fall 2000 (RSG:SO, Coloring time: 0.03 sec)

Figure 6.14: Rice University, Fall 2000 (RSG:DD, Coloring time: 0.07 sec)
Figure 6.15: Rice University, Fall 2000 (RSG:ID, Coloring time: 0.06 sec)

Figure 6.16: Rice University, Fall 2000 (RSG:RO, Coloring time: 0.04 sec)
Figure 6.17: Rice University, Fall 2000 (OPLStudio, Coloring time: 40.71 sec)

From the above computations, we can conclude that performing a greedy coloring on our course conflict graph by using a SMALLEST-FIRST-SEARCH color searching procedure, together with either a SPECIFIED ORDER initial vertex ordering (as described in Section 5.4) or a DECREASING DEGREE initial vertex ordering can best help to achieve our goals in terms of satisfying the timetabling conditions alluded to above. These two particular greedy coloring methods produced $\chi(G)$-colorings of essential-timetabling-condition- and preferential-timetabling-condition-influenced course conflict graphs with highly satisfactory color class distributions in minimal time. See Figures 6.9 and 6.10. Color classes 6 and 11, of sizes 2 and 33 respectively in Figure 6.9, and color classes 1 and 2, of sizes 2 and 33 respectively in Figure 6.10, contain vertices corresponding to courses that may have specifically requested to be timetabled for evening time slots. Due to the preferential timetabling condition that fewer courses be scheduled during evening hours than during morning and afternoon hours (i.e., “normal business hours”), these color classes contain significantly fewer
vertices than the others.
6.1.2 Rice University, Spring 2001

Using course data provided by Rice University for the Spring 2001 semester, together with aspects of our new timetabling-by-graph-coloring model, we constructed a "Method A" course conflict graph $G$ with 760 vertices (one vertex for each of the 760 courses to be scheduled) and 173,730 edges (reflecting potential timetabling conflicts due to either essential or preferential timetabling conditions, as described in Section 5.3.1), and conducted computational experiments involving each of the 16 greedy coloring variations that we implemented:

- SIMPLE-SEARCH GREEDY: SPECIFIED ORDER (Figure 6.18);
- SIMPLE-SEARCH GREEDY: DECREASING DEGREE (Figure 6.19);
- SIMPLE-SEARCH GREEDY: INCREASING DEGREE (Figure 6.20);
- SIMPLE-SEARCH GREEDY: RANDOM ORDER (Figure 6.21);
- LARGEST-FIRST-SEARCH GREEDY: SPECIFIED ORDER (Figure 6.22);
- LARGEST-FIRST-SEARCH GREEDY: DECREASING DEGREE (Figure 6.23);
- LARGEST-FIRST-SEARCH GREEDY: INCREASING DEGREE (Figure 6.24);
- LARGEST-FIRST-SEARCH GREEDY: RANDOM ORDER (Figure 6.25);
- SMALLEST-FIRST-SEARCH GREEDY: SPECIFIED ORDER (Figure 6.26);
- SMALLEST-FIRST-SEARCH GREEDY: DECREASING DEGREE (Figure 6.27);
- SMALLEST-FIRST-SEARCH GREEDY: INCREASING DEGREE (Figure 6.28);
- SMALLEST-FIRST-SEARCH GREEDY: RANDOM ORDER (Figure 6.29);
- RANDOM-SEARCH GREEDY: SPECIFIED ORDER (Figure 6.30);
- RANDOM-SEARCH GREEDY: DECREASING DEGREE (Figure 6.31);
- RANDOM-SEARCH GREEDY: INCREASING DEGREE (Figure 6.32);

and RANDOM-SEARCH GREEDY: RANDOM ORDER (Figure 6.33),

as well as a constraint programming graph coloring formulation (760 variables, 173,730 constraints) compatible with the OPL programming environment and OPLStudio
(Figure 6.34).

Each of the following bar charts in Figures 6.19 through 6.34 below provides the number of colors used to properly color the course conflict graph, together with the number of vertices (i.e., courses) in each color class). The computing time taken to perform and verify each coloring is also given in the caption of each figure. Our computations using OPLStudio prove that the chromatic number of $G$ is in fact equal to 13; attempts to properly color $G$ with 12 colors or less result in OPLStudio declaring that “no solution exists”. Therefore $\chi(G) = 13$, meaning that one would need a minimum of 13 colors to properly color $G$. Twelve colors would not suffice.

All computations were performed on a Sun Enterprise 220R workstation with two 450-MHz UltraSPARC II cpu's, 2 Gb memory and 2 Gb swap space (harvey.caam.rice.edu).
Figure 6.18: Rice University, Spring 2001 (SSG:SO, Coloring time: 0.04 sec)

Figure 6.19: Rice University, Spring 2001 (SSG:DD, Coloring time: 0.07 sec)
Figure 6.20: Rice University, Spring 2001 (SSG:ID, Coloring time: 0.07 sec)

Figure 6.21: Rice University, Spring 2001 (SSG:RO, Coloring time: 0.04 sec)
Figure 6.22: Rice University, Spring 2001 (LFSG:SO, Coloring time: 0.03 sec)

Figure 6.23: Rice University, Spring 2001 (LFSG:DD, Coloring time: 0.07 sec)
Figure 6.24: Rice University, Spring 2001 (LFSG:ID, Coloring time: 0.07 sec)

Figure 6.25: Rice University, Spring 2001 (LFSG:RO, Coloring time: 0.05 sec)
Figure 6.26: Rice University, Spring 2001 (SFSG:SO, Coloring time: 0.03 sec)

Figure 6.27: Rice University, Spring 2001 (SFSG:DD, Coloring time: 0.07 sec)
Figure 6.28: Rice University, Spring 2001 (SFSG:ID, Coloring time: 0.07 sec)

Figure 6.29: Rice University, Spring 2001 (SFSG:RO, Coloring time: 0.05 sec)
Figure 6.30: Rice University, Spring 2001 (RSG:SO, Coloring time: 0.04 sec)

Figure 6.31: Rice University, Spring 2001 (RSG:DD, Coloring time: 0.07 sec)
Figure 6.32: Rice University, Spring 2001 (RSG:ID, Coloring time: 0.07 sec)

Figure 6.33: Rice University, Spring 2001 (RSG:RO, Coloring time: 0.04 sec)
Figure 6.34: Rice University, Spring 2001 (OPLStudio, Coloring time: 55.15 sec)

From the above computations, we can also conclude that performing a greedy coloring on our course conflict graph by using a SMALLEST-FIRST-SEARCH color searching procedure, together with either a SPECIFIED ORDER initial vertex ordering (as described in Section 5.4) or a DECREASING DEGREE initial vertex ordering can best help to achieve our goals in terms of satisfying the timetabling conditions alluded to above. These two particular greedy coloring methods produced $\chi(G)$-colorings of essential-timetabling-condition- and preferential-timetabling-condition-influenced course conflict graphs with highly satisfactory color class distributions in minimal time. See Figures 6.26 and 6.27. Color classes 6, 12, and 13, of sizes 1, 21, and 18 respectively in Figure 6.26, and color classes 1, 2, and 3, of sizes 1, 19, and 19 respectively in Figure 6.27, contain vertices corresponding to courses that may have specifically requested to be timetabled for evening time slots. Again, due to the preferential timetabling condition that fewer courses be scheduled during evening hours than during morning and afternoon hours (i.e., “normal business hours”), these
color classes contain significantly fewer vertices than the others.
6.2 The University of St. Thomas

6.2.1 The University of St. Thomas, Spring 2001

Using course data provided by the University of St. Thomas for the Spring 2001 semester, together with aspects of our new timetabling-by-graph-coloring model, we constructed a "Method A" course conflict graph $G$ with 708 vertices (one vertex for each of the 708 courses to be scheduled) and 86,858 edges (reflecting potential timetabling conflicts due to either essential or preferential timetabling conditions, as described in Section 5.3.1), and conducted computational experiments involving each of the 16 greedy coloring variations that we implemented:

SIMPLE-SEARCH GREEDY: SPECIFIED ORDER (Figure 6.35);
SIMPLE-SEARCH GREEDY: DECREASING DEGREE (Figure 6.36);
SIMPLE-SEARCH GREEDY: INCREASING DEGREE (Figure 6.37);
SIMPLE-SEARCH GREEDY: RANDOM ORDER (Figure 6.38);
LARGEST-FIRST-SEARCH GREEDY: SPECIFIED ORDER (Figure 6.39);
LARGEST-FIRST-SEARCH GREEDY: DECREASING DEGREE (Figure 6.40);
LARGEST-FIRST-SEARCH GREEDY: INCREASING DEGREE (Figure 6.41);
LARGEST-FIRST-SEARCH GREEDY: RANDOM ORDER (Figure 6.42);
SMALLEST-FIRST-SEARCH GREEDY: SPECIFIED ORDER (Figure 6.43);
SMALLEST-FIRST-SEARCH GREEDY: DECREASING DEGREE (Figure 6.44);
SMALLEST-FIRST-SEARCH GREEDY: INCREASING DEGREE (Figure 6.45);
SMALLEST-FIRST-SEARCH GREEDY: RANDOM ORDER (Figure 6.46);
RANDOM-SEARCH GREEDY: SPECIFIED ORDER (Figure 6.47);
RANDOM-SEARCH GREEDY: DECREASING DEGREE (Figure 6.48);
RANDOM-SEARCH GREEDY: INCREASING DEGREE (Figure 6.49);
and RANDOM-SEARCH GREEDY: RANDOM ORDER (Figure 6.50),
as well as a constraint programming graph coloring formulation (708 variables, 86,858 constraints) compatible with the OPL programming environment and OPLStudio (Figure 6.51).

Each of the following bar charts in Figures 6.35 through 6.51 below provides the number of colors used to properly color the course conflict graph, together with the number of vertices (i.e., courses) in each color class. The computing time taken to perform and verify each coloring is also given in the caption of each figure. Our computations using OPLStudio prove that the chromatic number of $G$ is in fact equal to 16; attempts to properly color $G$ with 15 colors or less result in OPLStudio declaring that “no solution exists”. Therefore $\chi(G) = 16$, meaning that one would need a minimum of 16 colors to properly color $G$. Fifteen colors would not suffice.

All computations were performed on a Sun Enterprise 220R workstation with two 450-MHz UltraSPARC II cpu’s, 2 Gb memory and 2 Gb swap space (harvey.caam.rice.edu).
Figure 6.35: U. of St. Thomas, Spring 2001 (SSG: SO, Coloring time: 0.01 sec)

Figure 6.36: U. of St. Thomas, Spring 2001 (SSG: DD, Coloring time: 0.01 sec)
Figure 6.37: U. of St. Thomas, Spring 2001 (SSG:ID, Coloring time: 0.03 sec)

Figure 6.38: U. of St. Thomas, Spring 2001 (SSG:RO, Coloring time: 0.02 sec)
Figure 6.39: U. of St. Thomas, Spring 2001 (LFSG:SO, Coloring time: 0.01 sec)

Figure 6.40: U. of St. Thomas, Spring 2001 (LFSG:DD, Coloring time: 0.02 sec)
Figure 6.41: U. of St. Thomas, Spring 2001 (LFGS:ID, Coloring time: 0.02 sec)

Figure 6.42: U. of St. Thomas, Spring 2001 (LFGS:RO, Coloring time: 0.01 sec)
Figure 6.43: U. of St. Thomas, Spring 2001 (SFSG:SO, Coloring time: 0.00 sec)

Figure 6.44: U. of St. Thomas, Spring 2001 (SFSG:DD, Coloring time: 0.02 sec)
Figure 6.45: U. of St. Thomas, Spring 2001 (SFSG:ID, Coloring time: 0.03 sec)

Figure 6.46: U. of St. Thomas, Spring 2001 (SFSG:RO, Coloring time: 0.02 sec)
Figure 6.47: U. of St. Thomas, Spring 2001 (RSG:SO, Coloring time: 0.01 sec)

Figure 6.48: U. of St. Thomas, Spring 2001 (RSG:DD, Coloring time: 0.01 sec)
Figure 6.49: U. of St. Thomas, Spring 2001 (RSG:ID, Coloring time: 0.01 sec)

Figure 6.50: U. of St. Thomas, Spring 2001 (RSG:RO, Coloring time: 0.01 sec)
Figure 6.51: U. of St. Thomas, Spring 2001 (OPLStudio, Coloring time: 23.88 sec)

From the above computations, we can further conclude that performing a greedy coloring on our course conflict graph by using a SMALLEST-FIRST-SEARCH color searching procedure, together with either a SPECIFIED ORDER initial vertex ordering (as described in Section 5.4) or a DECREASING DEGREE initial vertex ordering can best help to achieve our goals in terms of satisfying the timetabling conditions alluded to above. These two particular greedy coloring methods produced $\chi(G)$-colorings of essential-timetabling-condition- and preferential-timetabling-condition-influenced course conflict graphs with highly satisfactory color class distributions in minimal time. See Figures 6.43 and 6.44. Color classes 7, 8, 15, and 16, of sizes 20, 20, 24, and 24 respectively in Figure 6.43, and color classes 1, 2, 3, and 4, of sizes 20, 20, 24, and 24 respectively in Figure 6.44, contain vertices corresponding to courses that may have specifically requested to be timetabled for evening time slots. Again, due to the preferential timetabling condition that fewer courses be scheduled during evening hours than during morning and afternoon hours (i.e., “normal business hours”), these
color classes contain fewer vertices than the others.
Chapter 7

Conclusions and Future Work

7.1 Conclusions

The main objective of this dissertation and its accompanying research was to improve our understanding of and ability to both model and solve university course and final examination timetabling problems using graph coloring techniques. We examined these problems from both mathematical and computational points of view, and to some extent, a sociological point of view as well. In practice, the university timetabling problem for courses and/or final examinations involves fixing in time and space a sequence of meetings between instructors and students, while simultaneously attempting to satisfy a number of various essential and/or preferential conditions or constraints. “Essential timetabling conditions” (also commonly referred to as “hard constraints”) are conditions or constraints that must be satisfied in order to produce a legal or feasible timetable. “Preferential timetabling conditions” (also commonly referred to as “soft constraints”) are additional conditions or constraints that need not necessarily be satisfied to produce a legal or legitimate timetable, but if satisfied may very well produce a more “acceptable” timetable for students and/or faculty members. Preferential conditions are requests that should be fulfilled, if possible. These conditions are often reasonable, but it may or may not be possible to sat-
isfy all of them in addition to all of the essential conditions. It is ultimately up to the scheduler’s discretion as to which, if any, preferential timetabling conditions he or she values most and/or wishes to satisfy in order to produce a timetable that is “acceptable”.

In this thesis, we discussed the relevance of university timetabling problems as a natural and practical application of graph coloring. We then surveyed and highlighted some of the major published research regarding graph coloring approaches to solving timetabling problems, as well as alternative approaches to timetabling such as mathematical programming and other methods. Through interviews conducted with staff in the Office of the Registrar at Rice University and at the University of St. Thomas, and in the Department of Registration and Academic Records at the University of Houston, we learned of and subsequently were able to present in this thesis a detailed description and analysis of course and final examination timetabling methods currently in practice at these universities.

The major contribution of this dissertation is the development of a new mathematical and computational model for solving university timetabling problems that blends graph coloring with the satisfaction of both essential and preferential timetabling conditions as described above, in an effort to produce satisfactory timetables. This multi-step model involves assembling the university course data, creating a course conflict graph based on the assembled data using either one of two methods, which we refer to as “Method A” and “Method B”, performing a proper coloring of the graph, transforming the coloring to a conflict-free timetable, and finally assigning courses to classrooms, based on room capacity and availability. Once a conflict-free course timetable has been constructed, we can use it to create a similarly conflict-free timetable of final examinations as well. Again, it is primarily up to the scheduler to determine which, if any, of the preferential timetabling conditions he or she is most wanting to satisfy, and which, if any, can be sacrificed in lieu of obtaining a feasible and acceptable timetable. The scheduler may very well wish to use our timetabling
model multiple times, incorporating different sets of preferential timetabling conditions each time, until finally arriving at a timetable that is ultimately most suitable to his or her liking.

To support our research ideas, we presented computational results and other observations using aspects of the new timetabling-by-graph-coloring model above along with actual data supplied by Rice University and the University of St. Thomas. These results demonstrate the effectiveness of graph coloring methods, particularly methods that employ intelligent color-searching and vertex-ordering techniques, in efficiently producing timetables that satisfy various essential and preferential timetabling conditions. One of these timetabling conditions is one that seeks as best of an even distribution of courses among time slots as possible, in an effort to aid in room assignment and also allow for a maximum utilization of university resources. To obtain these results we implemented 16 variations of the greedy approach to graph coloring, each variation incorporating one of four color searching procedures (SIMPLE-SEARCH, LARGEST-FIRST-SEARCH, SMALLEST-FIRST-SEARCH, or RANDOM-SEARCH) and one of four initial vertex orderings (SPECIFIED ORDER, DECREASING DEGREE, INCREASING DEGREE, or RANDOM), along with a constraint programming graph coloring formulation compatible with the OPL programming environment and the OPLStudio software package, and tested these implementations on the provided data. All of the running times were extremely short, with the greedy coloring implementations requiring run times on the order of only a few hundredths of a second, and the constraint programming implementation requiring run times on the order of a few seconds. It appears that performing a greedy coloring on a course conflict graph by using a SMALLEST-FIRST-SEARCH color searching procedure, together with either a SPECIFIED ORDER initial vertex ordering (as described in Section 5.4) or a DECREASING DEGREE initial vertex ordering can achieve our goals in terms of satisfying the timetabling conditions alluded to above. These two particular greedy coloring methods produced $\chi(G)$-colorings of essential-timetabling-condition-
and preferential-timetabling-condition-influenced course conflict graphs with highly satisfactory color class distributions in minimal time. Due to such promising results, we strongly hypothesize that our timetabling-by-graph-coloring model incorporating such specific graph coloring methods will ultimately produce more satisfactory course and final examination timetables for a wide range of colleges and universities.

7.2 Future Work

As with all research, there is always more that can be explored. Proposed and other possible items for future research in this area include:

- further examination and analysis of restricted (list) coloring approaches, as well as the use of mathematical programming methods and/or other approaches, for modeling and solving university course and final exam timetabling problems;

- continued computational experiments involving actual data from other colleges and universities and/or computer-generated data applicable to university timetabling problems;

- continued conversation with representatives from academic institutions of various sizes and types, in efforts to further determine ways to enhance the timetabling process of courses and final exams and make it more practical, useful, and efficient; and

- completion of the implementation of our new timetabling-by-graph-coloring model, which will serve as the driving force behind the design and implementation of a new timetabling software package that users (e.g., registrars at colleges and universities) can obtain and use to effectively construct satisfactory schedules of courses and final exams for their respective institutions.

In addition, to perhaps ignite a potential spark of even more interest in this area, we can consider further examining the relationships between timetabling problems
and other useful and important applications of graph coloring (e.g., register allocation or frequency assignment problems).

Graph coloring is a powerful tool whose theory maintains a central position in discrete mathematics, is of great interest for its many useful applications, and continually surprises us by producing new and often unexpected answers to a number of interesting problems.
Bibliography


[17] P.M. Cavanaugh and C. Tran. Private communication, University of Houston, Houston, TX.


[30] W. Erben and J. Keppler. A genetic algorithm solving a weekly course-
timetabling problem. In Proceedings of the First International Conference on

[31] A. Gamst. Some lower bounds for a class of frequency assignment problems.


[33] C.C. Gotlieb. The construction of class-teacher time-tables. Proc. IFIP Congress,

[34] A. Hertz and D. de Werra. Using tabu search technique for graph coloring.


editor, Conference Record of the 14th Annual ACM Symposium on Principles of
Programming Languages, Munich, Germany, January 1987.


[53] A. Mislove. Private communication, Rice University, Houston, TX.

[54] J. Montag and L. Branch. Private communication, Rice University, Houston, TX.


[56] Y. Obregon. Private communication, University of St. Thomas, Houston, TX.


