A MULTI-PHASE APPROACH TO UNIVERSITY COURSE TIMETABLING

MINHAZ FAHIM ZIBRAN
Bachelor of Science in Computer Science and Information Technology
Islamic University of Technology (Bangladesh), 2002

A Thesis
Submitted to the School of Graduate Studies
of the University of Lethbridge
in Partial Fulfillment of the
Requirements for the Degree

MASTER OF SCIENCE

Department of Mathematics and Computer Science
University of Lethbridge
LETHBRIDGE, ALBERTA, CANADA

I dedicate this thesis to my beloved mother whose selfless support and inspiration has always been with me at each and every step of my life.

Abstract

Course timetabling is a well known constraint satisfaction optimization (CSOP) problem, which needs to be solved in educational institutions regularly. Unfortunately, this course timetabling problem is known to be NP-complete [7, 39].

This M.Sc. thesis presents a multi-phase approach to solve the university level course timetabling problem. We decompose the problem into several sub-problems with reduced complexity, which are solved in separate phases. In phase-1a we assign lectures to professors, phase-1b assigns labs and tutorials to academic assistances and graduate assistants. Phase-2 assigns each lecture to one of the two day-sequences (Monday-Wednesday-Friday or Tuesday-Thursday). In Phase-3, lectures of each single day-sequence are then assigned to time-slots. Finally, in phase-4, labs and tutorials are assigned to days and time-slots.

This decomposition allows the use of different techniques as appropriate to solve different phases. Currently different phases are solved using constraint programming and integer linear programming. The multi-phase architecture with the graphical user interface allows users to customize constraints as well as to generate new solutions that may incorporate partial solutions from previously generated feasible solutions.

Acknowledgments

"The problem is never how to get new, innovative thoughts into your mind, but how to get old ones out. Every mind is a building filled with archaic furniture. Clean out a corner of your mind and creativity will instantly fill it."

- Dee Hock

I take much pleasure to express my profound gratitude to my supervisor Dr. Shahadat Hossain for his persistent and inspiring supervision.

I also thank my M.Sc. supervisory committee members Dr. Daya Gaur and Dr. Saurya Das for their valuable suggestions and guidance.

Special thanks to the distinguished staff of the registrar's office and dean's office who attended the presentation I gave on the preliminary results of our timetabling implementation. Their feedback and suggestions were useful indeed.

My cordial thanks to NSERC and the University of Lethbridge for the financial and travel support. I am also thankful to all my fellow researchers and faculty members in the Department of Mathematics and Computer Science for their spontaneous cooperation and encouragement.

Above all, my sincere gratitude to the Almighty, who creates and makes things happen.

"So, before we end, and then begin, we'll drink a toast to how it's been..."

- Billy Joel, "Ive Loved These Days"

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Timetabling is the allocation of a set of activities in time and space subject to some constraints to achieve a set of desirable objectives. There are numerous NP-hardness results on various types of timetabling problems. Therefore, in general, timetabling problems are known to be NP-hard [39], i.e., no polynomial time algorithm is known to solve the problem.

Course timetabling is a well-known constraint satisfaction combinatorial optimization problem, which needs to be solved regularly at educational institutions. University level course timetabling is concerned with scheduling courses in an academic semester or year. This course timetabling typically involves assigning courses to teachers and classrooms, as well as the teacher-student meeting-times in days. Such a timetable or schedule must satisfy certain constraints such as no single teacher teaches more than one class at the same time, and so on. Further, it may try to achieve certain objectives such as maximum utilization of classrooms, assigning teachers to his or her preferred courses, etc. A typical university level course timetabling problem instance may involve thousands of courses, thousands of students, hundreds of instructors, hundreds of classrooms and other resources. The presence of such a wide range of constraints, preferences, and participants make the problem too hard to be solved in reasonable time.

There are a number of approaches to solve timetabling problems. Linear programming (LP), heuristics, and meta-heuristic methods have been in use for long. Constraint programming (CP) is a comparatively recent technology for solving constraint satisfaction problems. Its major advantage is its ability to give precise declarative description of the problem in terms of constraints (relationships among variables). It exploits the constraint satisfaction structure of the problem.

In this M.Sc. thesis we present a multi-phase approach to solve the university level course timetabling using integer linear programming and constraint programming techniques. Part of this thesis has appeared in [19, 37].

We decompose the entire timetabling problem into several subproblems and develop mathematical models for each of the subproblems. Each of these subproblems are solved in separate phases. Hence, at a time we deal with a smaller subproblem of reduced complexity. The multi-phase architecture also provides the flexibility to use different solution techniques as appropriate for the subproblems. In our implementation we use integer programming or constraint programming techniques deemed appropriate in solving different subproblems. The multi-phase implementation also makes the scheduling process more transparent to the user. The solutions obtained between phases may be examined and modified by the user. Thus the expertise of the user may be utilized efficiently.

The entire software implementation is developed by interweaving different loosely coupled modules. These modules are developed following software engineering principles keeping less dependencies between modules. This makes our implementation more manageable and easy to accommodate future modification and amendment. All the architectural complexities are hidden from the user by the graphical user interface carefully designed following HCI (Human Computer Interaction) principles.

Most of the works done so far on course timetabling focus on making schedules to minimize the students' conflicting classes. A major problem with this approach is that when students drop a course, it may not be offered any more, if the number of students registered in the course goes below a threshold. This necessitates last minutes changes in the schedule, which may affect other courses as well as the faculty members. In our approach the list of courses to be offered is determined based on program requirement and students' trend on choosing courses, while taking into account the expertise of the available faculty members. After the courses to be offered are finalized, the schedule is then made

based on the availability and preferences of the faculty members. This procedure reduces, though does not avoid the chances of last minute changes in the schedule. However, our implementation gives the flexibility to easily accommodate such changes in the schedule (see chapter 5).

This thesis is organized as follows. In *Chapter 2* we first give a short introduction to computational complexity, NP-completeness, and NP-hardness. Then we illustrate linear programming and constraint programming techniques.

Chapter 3 first describes the timetabling problem and related problems, such as scheduling, sequencing, and rostering. Then we focus on academic timetabling. Here we discuss the typical hard and soft constraints concerned with the university level course timetabling. Finally we present different approaches for solving timetabling problems.

In *Chapter 4* we describe the course timetabling problem at the University of Lethbridge. Here we discuss various hard and soft constraints and other specifications related to timetabling at this university. Finally, we present details of our multi-phase approach to solve the problem.

Chapter 5 presents the details about the implementation of our multi-phase approach. Here we describe the architecture of the software implementation and the input format. We also describe the graphical user interface and highlight its major features. Finally we discuss the flexibility and scope of customization our software implementation provides to the users.

In *Chapter 6* we present the experimental results, which evaluate the performance of our solution technique, and provide justification of our multi-phase approach.

Finally, in *Chapter 7* we conclude the thesis with some remarks and probable directions for extension of this work and further research in this area.

The ILP and CP models of all the phases are included in Appendix-A. Implementation of the Java GUI has more than 17,000 lines of code, whereas implementation the C++

module has more than 1,800 lines of code. Moreover, implementation of the timetabling data generator also has more than 1,000 lines of code. Due to the volume these codes are not included in this thesis. However, these are included in the companion CD-ROM. Appendix-B lists the contents of the CD-ROM.

Chapter 2

Background

Timetabling problems in general are "NP-hard". We begin this chapter with a brief discussion on the complexity of problems, NP-completeness, and NP-hardness. For solving the course timetabling problem we use constraint programming (CP) and integer linear programming (ILP). Section 2.2 illustrates linear programming and its variants including integer linear programming. In section 2.3 we discuss constraint programming and in section 2.4 we mention about several tools and frameworks for constraint programming and mathematical programming. We conclude the chapter with a summary.

2.1 Computational Complexity

In general, any polynomial-time algorithm for a certain problem is considered as a good algorithm. A polynomial-time algorithm may be defined to be one whose time complexity function is $O(p(n))$ for some polynomial function $p(n)$, where n is used to denote the input length [18]. Problems that are solvable by polynomial-time algorithm are considered as being tractable, or easy [40]. Such problems belong to the *class P*.

Problems that are "verifiable" in polynomial time belong to the *class NP*. For example, in the "Graph 3-Colorability" problem, given a graph $G = (V,E)$ it may be verified in polynomial time if this graph is 3-colorable, i.e., if there exists a function $f : V \mapsto \{1,2,3\}$, such that $f(u) \neq f(v)$, whenever $\{u,v\} \in E$. This verification may be done in $|V|$ time by checking for each of the vertices in V, if there is any vertex with degree higher than 3.

Formally, a problem Π is *NP-complete* if

1. $\Pi \in NP$, and

2. $\Pi' \propto \Pi$ for every $\Pi' \in NP$

Here, "$\Pi' \propto \Pi$" denotes that "Π' is polynomially transforms to Π". If a problem satisfies the second condition but not necessarily the first one, we say that the problem is *NP-hard*. NP-Complete problems are the hardest problems in NP. No polynomial time algorithm is known which can possible solve an NP-complete problem.

Therefore, when a problem is proved to be NP-complete or NP-hard, it suggests that the problem is hard to solve. The definition given above implies that there are two major steps to prove NP-completeness of a problem Π.

1. prove that $\Pi \in NP$, and

2. select a known NP-complete problem Π' and polynomially transform it to Π

The second step itself is sufficient to prove that the problem Π is NP-hard.

2.2 Linear Programming (LP)

Linear programming is a mathematical programming technique to solve optimization problems. The basic idea is to formulate the problem as a linear programming problem and solve it using linear programming algorithms. *A linear programming problem* is a mathematical formulation of an optimization problem defined in terms of an objective function and a set of constraints. The objective function is a linear function of the unknowns (variables) and the set of constraints consists of linear equalities and linear inequalities. It can be expressed in the following standard form [30].

$$\begin{aligned} \text{minimize} \quad & c^T x \\ \text{subject to} \quad & Ax = b \\ & x \geq 0 \end{aligned}$$

where, $(.)^T$ denotes matrix transposition operation, $x \in \Re^n$ is the vector of variables (called decision variables) to be determined, $A \in \Re^{m \times n}$ is a matrix of known coefficients, and $c \in \Re^n$ and $b \in \Re^m$ are known vectors. The expression $c^T x$ is called the objective function, and the equations $Ax = b$ and $x \geq 0$ are called the constraints. The assignment of values to the vector x of variables satisfying the given constraints is called a feasible solution. The feasible solution, in which the objective function obtains its minimum (or maximum in maximization problems) possible value, is said to be the optimal solution.

Integer linear programming (ILP) or simply *integer programming (IP)* is a subset of linear programming, where some or all of the variables are restricted to take only integer or whole number (as opposed to fractional) values. If all the variables are restricted to take only integer variables, the problem is called *pure integer linear programming problem*. If the restrictions are such that, some but not all of the variables can take only integer values, then such a problem is said to be *mixed integer linear programming problem*. If the variables may take either 0 or 1, then such a problem is called *binary integer linear programming problem*.

In our work we use integer linear programming as in the university course timetabling problem all the participants and the resources are integral.

2.2.1 Algorithms for Integer Programming

There are a number of algorithms for solving linear programming problems, such as simplex method, ellipsoid methods, and interior-point techniques [45]. But the simplex method developed by George Dantzig is the most widely used.

For solving integer linear programming problems algorithms such as "branch-and-bound", "branch-and-cut", and "cutting- plane" are used to preserve the integrality of the decision variables. In practice, most general-purpose large-scale ILP codes use "branch-and-bound" to search for an optimal integer solution by solving a sequence of related LP "relaxations" that allow some fractional values.

Suppose we need to solve the following ILP:

$$\begin{aligned} \text{maximize} \quad & z = 8x_1 + 5x_2 \\ \text{subject to,} \quad & x_1 + x_2 \leq 6 \\ & 9x_1 + 5x_2 \leq 45 \\ & x_1, x_2 \geq 0, \quad x_1, x_2 \textit{ integer} \end{aligned}$$

To begin, we find the LP relaxation of the given problem by removing the integrality constraint on the decision variables. We call this LP relaxation as subproblem_1, which is as follows.

$$\begin{aligned} \text{maximize} \quad & z = 8x_1 + 5x_2 \\ \text{subject to,} \quad & x_1 + x_2 \leq 6 \\ & 9x_1 + 5x_2 \leq 45 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Now we solve this subproblem_1 (using any LP algorithm, such as the simplex method). If the optimal solution to the LP relaxation has an integer solution, this optimal solution is also the optimal solution to the ILP. Otherwise, the objective value of the optimal solution to the LP relaxation gives an upper bound on the objective value of the ILP.

As we find the optimal solution to subproblem_1 with $z = 41.25, x_1 = 3.75$, and $x_2 = 2.25$, we see that none of the decision variables are integral.

So, take step-2, where we partition the feasible region of the LP relaxation to find out more about the location of the ILP's optimal solution. We arbitrarily choose a decision variable, say x_1, whose value is fractional in the optimal solution of the LP relaxation. Any feasible solution to the ILP must have $x_1 \leq 3$ or $x_1 \geq 4$. This observation leads us to branch on the variable x_1 creating the following two subproblems:

subproblem_2: subproblem_1 + constraint $x_1 \geq 4$, which is

$$\begin{aligned} \text{maximize} \quad & z = 8x_1 + 5x_2 \\ \text{subject to,} \quad & x_1 + x_2 \leq 6 \\ & 9x_1 + 5x_2 \leq 45 \\ & x_1 \geq 4 \\ & x_2 \geq 0 \end{aligned}$$

subproblem_3: subproblem_1 + constraint $x_1 \leq 3$, which is

$$\begin{aligned} \text{maximize} \quad & z = 8x_1 + 5x_2 \\ \text{subject to,} \quad & x_1 + x_2 \leq 6 \\ & 9x_1 + 5x_2 \leq 45 \\ & x_1 \geq 0 \\ & x_2 \leq 3 \\ & x_2 \geq 0 \end{aligned}$$

We now continue solving the feasible subproblems one after another. After solving a subproblem,

- If the optimal solution to the subproblem has all decision variables assigned integer values, we compare its objective value to the best (maximum for maximization problem and minimum for minimization problem) objective value (in solutions where decision variables are integral) found so far. If the objective value of the optimal solution to the current subproblem is better than the best objective value found do far, we keep a record on this subproblem pointing its objective value to be the best so far. Otherwise, we just discard the subproblem.

- If the optimal solution to the subproblem has any decision variable assigned to fractional value, we branch on any of the fractional decision variables creating two more subproblems.

When there is no subproblem left as candidate to be solved, the one with integral decision variables recorded to have the best objective value, is the optimal solution to the ILP.

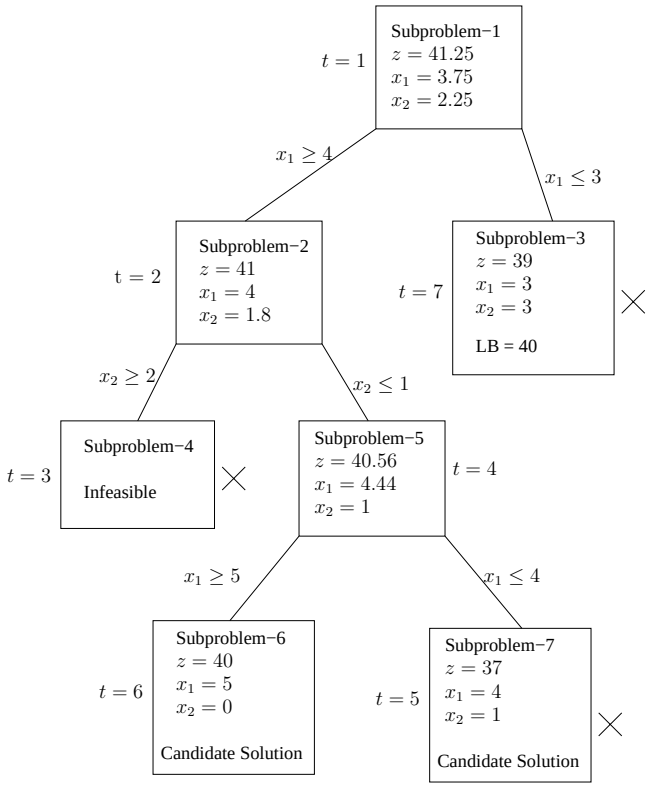

Figure 2.1: Branch-and-bound method for solving ILP

Figure 2.1 presents the tree, where each node refers to a subproblem. Each line connecting two nodes may be called an edge. The constraints associated to any node of the tree is the constraints for the LP relaxation plus the constraints associated with the edges leading from subproblem_1 to the node. The cross ($\times$) sign beside any node indicates that the node was fathomed. The label *t* besides the nodes indicates the chronological order in which the subproblems are solved. As we see, the optimal solution to subproblem_6 is the optimal solution to the above mentioned ILP.

Integer programming problems are usually much harder to crack than ordinary linear programming problems [30]. The difficulty of a particular ILP instance is hard to predict.

Problems with no more than a hundred variables can be challenging, while others in tens of thousands of variables may be solved readily. The best explanations of why a particular ILP is difficult often rely on some insight into the system to be modelled and it is observed that the way the model is formulated is as important as the choice of a solver [20].

2.3 Constraint Programming (CP)

Constraint programming (CP) is comparatively recent computer programming technique for solving optimization problems. Now-a-days, constraint programming is widely used to solve problems on planning, scheduling, and optimization.

Early ideas leading to constraint programming are found in the area of Artificial Intelligence (AI) dating back to sixties and seventies [23]. The major step towards modern constraint programming was achieved when *logic programming* was noted to be just a particular kind of constraint programming. The basic idea behind logic programming, and *declarative programming* in general is that, the programmer state what to solve (to be satisfied) but not how. This declarative paradigm is quite close to the idea of constraints and constraint programming. Therefore, the necessary combination of logic programming and constraints led to the evolution of *constraint logic programming (CLP)* [23]. As constraint programming was first embedded in logic programming languages, the field was initially called constraint logic programming. The first implementations of constraint logic programming were Prolog III, CLP(R), and CHIP. Constraint programming is also integrated in imperative languages such as C++ and Java [31].

Constraint programming is a programming paradigm where relationships among variables are stated in the form of constraints. The idea is to formulate the problem in terms of variables and constraints, and solving it (instantiating the decision variables) satisfying the constraints.

The essence of modern constraint programming is a two-level architecture integrating a constraint store and a programming component [15]. The *constraint store* provides the basic operations of the architecture and consists of a system reasoning about fundamental properties of constraint systems such as satisfiability and entailment. The constraint store contains the constraints accumulated at some computation step and supports various queries and operations over these constraints. Operating around the constraint store is a *programming language component* that specifies how to combine the basic operations, often in non-deterministic ways.

A *constraint* may intuitively be considered as a restriction on a space of possibilities. A mathematical constraint is a precisely specifiable relationship among a set of variables taking values from their associated domains [23].

Suppose, there is a set of n decision variables $V = \{x_1, x_2, x_3, \ldots, x_n\}$, and a set of n domains $D = \{D_1, D_2, D_3, \ldots, D_n\}$ such that D_i is the domain of variable x_i. A constraint $c(x_1, x_2, x_3, \ldots, x_n)$ may be defined by a mathematical relation, that is, a subset S of $D_1 \times D_2 \times D_3 \times \ldots \times D_n$, such that if $(x_1, x_2, x_3, \ldots, x_n) \in S$, then the constraint is said to be satisfied [16].

Consider a set of three variables: $V = \{x, y, z\}$. The domain of all these variables is real numbers. A mathematical constraint c may be given as $x + y \leq z$. Such mathematical constraints usually have some remarkable properties [23, 31].

- Constraints may provide partial information about the possible values of a set of variables. A constraint may not specify the exact value of variables. For example, in the above mentioned example, constraint c does not specify any exact values to the variables. It only says that the summation of the value of x and y must be less than or equal to the value of z.

- Constraints are additive. The above mentioned constraint c and another constraint

c_1, say $x+y \geq z$ can be added resulting the conjunction $c_2 : x+y=z$.

- Constraints are rarely independent. For instance, once c and c_1 are imposed, it is the case that the constraint c_2 is implied.

- Constraints are non-directional, i.e., any of the variables may be chosen *arbitrarily* to infer constraints on it from the constraints of the other variables. For example, constraint on x may be inferred from the constraints on y and z, or constraint on y can be inferred from the constraints on x and z, and so on.

- Constraints are declarative. Typically a constraint specifies the relationship on the values of variables that must hold, without specifying the computational procedure to enforce the relationship.

To apply constraint programming for solving a problem first we formulate the problem as a constraint satisfaction problem. A *constraint satisfaction problem (CSP)* consists of a set of variables associated with finite domains and a set of constraints restricting the values that the variables can simultaneously take [42]. A feasible (consistent) complete solution of the problem is the assignment of values to all the variables from their respective domains while satisfying all the given constraints. Using the notations mentioned in above, a constraint satisfaction problem (CSP) typically has the following structure [16].

Given a set of n decision variables $V = \{x_1, x_2, x_3, \ldots, x_n\}$, a set of n domains $D = \{D_1, D_2, D_3, \ldots, D_n\}$, and a set of m constraints $c_1(x_1, x_2, x_3, \ldots, x_n)$, $c_2(x_1, x_2, x_3, \ldots, x_n)$, $\ldots, c_m(x_1, x_2, x_3, \ldots, x_n)$ find the value assignment of $x_1, x_2, x_3, \ldots, x_n$ such that

$$c_k(x_1, x_2, x_3, \ldots, x_n), \qquad 1 \leq k \leq m;$$
$$x_i \in D_i, \qquad 1 \leq i \leq n$$

This structure is suitable for finding any feasible solution of the problem. However, constraint programming also allows the use of an objective function to formulate an optimization problem. An objective function may be denoted as $f : D_1 \times D_2 \times D_3 \times \ldots, \times D_n \mapsto \Re$, so that at any feasible point of the problem the function $f(x_1, x_2, x_3, \ldots, x_n)$ may be evaluated. A constraint programming formulation of an optimization problem typically has the following structure [16].

$$\begin{aligned} \textit{minimize} \quad & f(x_1, x_2, x_3, \ldots, x_n) \\ \text{subject to,} & \\ c_k(x_1, x_2, x_3, \ldots, x_n), \quad & 1 \leq k \leq m; \\ x_i \in D_i, \quad & 1 \leq i \leq n \end{aligned}$$

2.3.1 Algorithms for Constraint Programming

To determine solutions to constraint satisfaction and optimization problems a constraint programming system typically use asystematic search applying constraint propagation and domain reduction. Based on the given constraints the domains of underlying variables are modified. When a variable's domain is modified, the effects of this modification are then propagated to any constraint involving that variable. This communication is called *constraint propagation* [1, 16].

For each constraint, a *domain reduction algorithm* detects inconsistencies among the domains of variables in that constraint and removes inconsistent values. When a particular variable's domain becomes empty, it may be determined that the constraint cannot be satisfied and backtracking may occur undoing an earlier choice.

Given a set of variables with their domains, and a set of constraints on those variables,

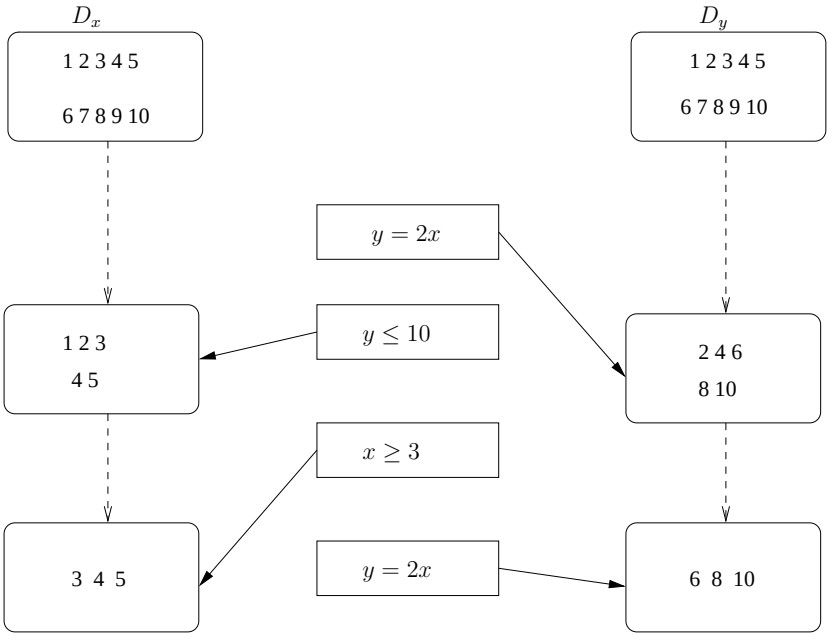

Figure 2.2: Constraint propagation and domain reduction

a constraint programming system repeatedly applies constraint propagation and domain reduction algorithm to make the domain of each variable as small as possible while keeping the entire system arc-consistent [1].

To illustrate, consider two variables x and y with their domains given as

$D_x = \{1,2,3,4,5,6,7,8,9,10\}$, and

$D_y = \{1,2,3,4,5,6,7,8,9,10\}$, and the three constraints,

Constraint_1: $y = 2x$,

Constraint_2: $y \leq 10$, and

Constraint_3: $x \geq 3$.

[1] A constraint is arc-consistent if all the values in the domains of all the variables in that constraint arc consistent. A constraint system is arc-consistent if all of the constraints are consistent [1, 16].

Figure 2.2 depicts the constraint propagation and domain reduction process. Clearly, Constraint_1 forces that y must be even. So the application of this constraint reduces the domain of y to $D_y = \{2,4,6,8,10\}$. Now, constraint_2 in conjunction with constraint_1 implies that $x \leq 5$. Therefore, the domain of x reduces to $D_x = \{1,2,3,4,5\}$. Again, constraint_3 reduces the domain of x to $D_x = \{3,4,5\}$. The effect of constraint_3 propagates to constraint_1, which again reduces the domain of y to $D_y = \{6,8,10\}$.

For optimization problem, the objective function is taken into account. The standard search procedure first finds a feasible solution ignoring the objective function $f(x_1, x_2, x_3, \ldots, x_n)$. Let us call each feasible solution a feasible point in the search space. Let, $(y_1, y_2, y_3, \ldots, y_n)$ represents a feasible point. The search space is then pruned by adding the constraint $f(y_1, y_2, y_3, \ldots, y_n) > f(v_1, v_2, v_3, \ldots, v_n)$. This additional constraint forces that any new feasible point must have a better objective value than the current point. Propagation of this constraint and domain reduction reduce the size of the search space. As the search proceeds, new feasible points have progressively better objective values. The procedure continues until no feasible point is found. When this happens, the last feasible point is considered to be the optimal solution [16].

In order to find the optimal solution, typical constraint programming solvers need to explore in the worst case all the feasible solutions and compare them based on the objective function's values. For practical problems often we do not want to find the best solution. A good enough solution near to the optimum suffices. For example, we may look for a solution with its objective function's value above (or below) a given threshold or the best solution found within certain time limit [42].

2.4 Tools for CP and LP

There are a number of solvers and packages available for linear programming and constraint programming. For example, AMPL, AIMMS, GLPK, Mathematica, LINDO, MOSEL, etc. support linear programming. Early implementations of constraint programming were Prolog III, CLP(R), CHIP, and O_2. Recently support for constraint programming is also incorporated in programming languages by introducing separate libraries (for example Disolver, Gcode for C++) and packages (JOpt package for Java).

ILOG's OPLStudio is a popular integrated development environment for mathematical and constraint programming. *Optimization programming language (OPL)* is a result of the attempt to unify the mathematical programming (modeling) and constraint programming languages, and their implementation technologies [24]. OPLStudio incorporates optimization programming language (OPL) with several engines and tools for constraint programming, linear programming and other mathematical programming problems. OPL was motivated by modeling languages such as AMPL and GAMS that provide computer equivalents to traditional algebraic notation. The most significant dimension of OPL is the support for constraint programming [15].

Besides implementing the two level architecture mentioned in section 2.3, OPL goes far beyond traditional modeling languages by supporting novel modeling concepts (such as activities, resources), customized search techniques, and some useful data and logical constructs. This is why we choose OPL (OPLStudio 3.7) for carrying out the integer linear programming and constraint programming part in the implementation of our course timetabling package described in chapter 4.

2.5 Summary

Operations Research (OR) has long been solving optimization problems using linear programming (LP). In the linear programming formulation of a problem the objective function as well as the constraints must be expressed as linear equalities or inequalities. Constraint programming (CP) comes with a higher level of modeling and solution methods that are easier to understand and manipulate [31]. It attempts to reduce the gap between the high-level description of an optimization problem and the algorithm implemented to solve it. Varieties of expressions, such as linear and quadratic expressions, implications, etc. can be used in constraint programming. In CP, a variable's domain may have holes (nonconsecutive values), but in LP domains are intervals [16]. Compared to linear programming, a weakness of constraint programming is, when formulated as minimization problem, lower bound for the objective function may not exist. But in integer linear programming a lower bound always exists because of the LP relaxation of the problem.

Whether ILP outperforms CP or the opposite, depends on the structure of the problem and careful implementation as well. For large combinatorial problems rather than finding the optimal solution we often want to find a good enough solution within a given time limit or a solution having objective value above (or below) a given threshold. In such cases, experimental results show that CP computes the desired feasible solutions faster than IP does.

Chapter 3

Timetabling Problems

This chapter describes timetabling problems with emphasis on academic timetabling. Section 3.1 depicts timetabling problems in general and its variants. Section 3.2 illustrates different types of timetabling problems in the academic arena. In section 3.3 we discuss the university level course timetabling, and in section 3.4 we describe different approaches for solving timetabling problems. Finally, section 3.5 concludes with a summary.

3.1 Timetabling and Related Problems

Intuitively, timetabling is the sequencing of certain events along with the records on the time of their occurrences subject to some given constraints. Such problems in general belongs to the complexity class "NP-complete" [39]. Sometimes, the word timetable is interchangeably used with other terms, such as schedule and sequence, as though they are synonymous. But, in fact there are some keen distinctions among these terms [4, 7, 8, 32, 35, 44].

A. Wren defines scheduling, timetabling, sequencing and rostering in the following ways [4]:

Definition 3.1 (sequencing) *Sequencing is the construction, subject to constraints, of an order in which activities are to be carried out or objects are to be placed in some representation of a solution.*

Definition 3.2 (timetabling) *Timetabling is the allocation, subject to constraints, of given resources to objects being placed in space-time, in such a way as to satisfy a set of desirable objectives as much as possible.*

Definition 3.3 (scheduling) *Scheduling is the allocation, subject to constraints, of resources to objects being placed in space-time, in such a way as to minimize the total cost of some set of the resources used.*

Definition 3.4 (rostering) *Rostering is the placing, subject to constraints, of resources into slots in a pattern. One may seek to minimize (or maximize) some objective, or simply to obtain a feasible allocation. Often the resources will rotate through a roster.*

The aforementioned definitions are not strict enough to classify problems into several disjoint sets. Some problems may fit more than one of the above definitions. Therefore, the terms are loosely used in the community.

Sequencing is simply the ordering of a set of activities, specifying which activity follows which. For example, the ordering of the jobs to be processed through machines in a factory. Sequencing may also take into account the costs related to one particular job following another. Sequencing with such considerations is called *flow shop problem* [42].

A *timetable* shows when a particular event takes place, often the duration of the event, as well. A bus or train timetable shows journeys are to be commenced on different particular routes and stoppages. It does not tell the passengers exactly which particular vehicle or driver is assigned to a particular route. This allocation of vehicles and drivers to routes is the part of the *scheduling* process. However, to ensure compatibility between the timetable and schedule, often the problem is addressed as a whole.

Rostering is the placing of resources in available slots satisfying a set of constraints. Consider the n-queen problem, where n number of queens have to be placed in a $n \times n$ grid, in such a way that at no queen is able to attack any other. This problem may be considered as a rostering problem.

3.2 Academic Timetabling

In the academic milieu there are different resources (courses, classrooms, etc.) and participants (students, teachers, etc.) in a timetable.

Andrea Schaef classifies academic timetabling problems into three major types as follows [3]:

i) *School Timetabling:* The weekly schedule for all classes of an elementary or a high school, avoiding teacher meeting at the same time and vice versa.

ii) *Course Timetabling:* The weekly schedule for all lectures of a set of university courses, minimizing overlaps of lectures of courses having common students.

iii) *Exam Timetabling:* The scheduling of the exams of a set of university courses, avoiding overlap for courses having common students, and spreading the exams for the students as much as possible.

Carter and Laporte [22] further classifies academic timetabling into five subproblems:

i) *Teacher assignment* only addresses assignment of teachers to courses, without considering how the courses will be allocated to time and classrooms.

ii) *Class-teacher timetabling* only considers scheduling of the teachers' teaching times ensuring that no teacher teaches more than one classes at the same time. It assumes that the teachers are already assigned to classes they teach.

iii) *Course scheduling* focuses on the allotment of the courses to time and classrooms.

iv) *Student scheduling* takes into account the scheduling of courses to time in such a way that courses taken by the same student are not scheduled concurrently.

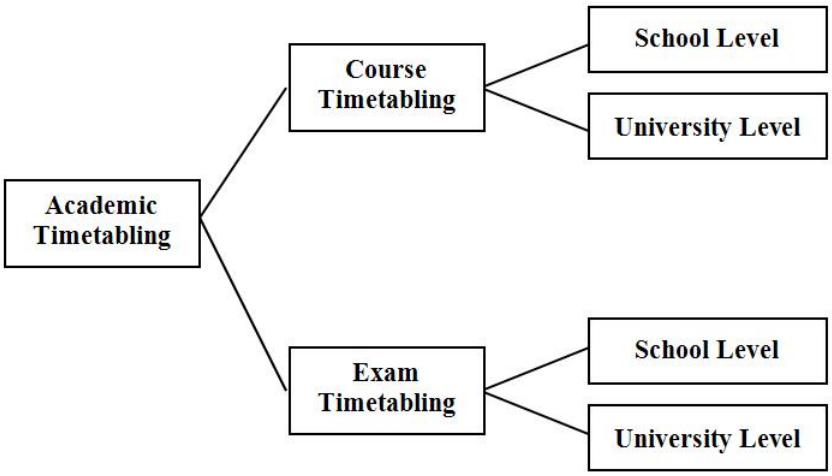

Figure 3.1: Classification of academic timetabling problems

v) *Classroom assignment* allocates classrooms to courses considering special needs of the courses, such as required equipments, seating capacity, etc.

As we perceive, these subproblems are not independent of each other. However, Aldy Gunawan and et. al. [2] presents a more systematic classification as shown in figure 3.1.

A *course timetable* at school level typically specifies when each class has a particular lesson, taught by which teacher, and in which classroom it is to be taught. On the other hand, course timetabling at the university level is much more complicated. The main difference from the school timetabling is that the university courses often have common students, whereas school classes consist of disjoint sets of students. Therefore, in university timetabling, courses with common students should not be scheduled in overlapping time. Moreover, in university course timetabling, availability of rooms, their seating capacity, and equipments play an important role, which is often neglected in school timetabling.

An *examination timetable* describes which exam takes place in which day and time. There is not much obvious difference between exam timetabling in university level and school level. Examination timetabling usually has the following characteristics, which

differentiates it from course timetabling [3]:

- There is only one exam for each subject.
- The conflict condition is more strict. Course timetabling may force a student to drop a course, but exam timetabling cannot force a student to skip an exam.
- There may be more than one exam per room.
- There are usually different types of constraints, such as, for each student there may be at most one exam per day, and exams of each student needs to be scattered over days as much as possible.
- The number of periods may vary, in contrast to course timetabling where it is fixed.

In this thesis we focus on university level course timetabling problem.

3.3 University Course Timetabling

As mentioned earlier, university course timetabling is more complicated than school course timetabling. This is because of the large number and type of constraints to be taken care of in university course timetabling. Many researchers like to classify these constraints into two categories [8]:

Hard constraints: Hard constraints are those that must be satisfied in any feasible timetable.

Soft constraints Soft constraints are those, which are desirable to be satisfied but not mandatory.

Some common hard constraints include the following [42].

- At any time no relocatable resource (student, staff, etc.) can be used in more than one place.
- No non-relocatable resource (classroom, etc.) can be used by more than one party (class or course) at any time.
- For each period there should be sufficient resources (classrooms, staff, etc.) available for all the events scheduled for that period.

Examples of soft constraints are

Course assignment: A course may need to (or should not) be taught by certain instructor and be scheduled on certain day and period.

Course Ordering: A course may need to be scheduled before or after another course.

Overlapping: It may be desirable that a set of courses be scheduled in non-overlapping periods.

Instructor's preference: An instructor may want to teach certain courses, in certain classrooms on certain days, and at certain periods.

Spreading of courses: For courses taught by the same instructor it may be desirable for them to be scheduled in non-consecutive periods.

Continuity: Any constraint whose main purpose is to ensure certain regularities and consistences. For instance, lectures of the same course may only be scheduled in the same room, or at the same time of day.

Typically, a university has several academic departments offering a multitude of courses. A student may choose courses offered by different departments. There may be certain courses across departments which need to be scheduled in non-overlapped time. Such

courses are often called *critical courses*. In most cases the department is responsible for preparing own timetable taking care of the critical courses across other departments.

Formally, the course timetabling problem may be defined as a search problem as stated below [3, 7].

Definition 3.5 (Course timetabling) *Let, $C = \{C_1, C_2, C_3, \ldots, C_q\}$ be a set of courses, and for each i $(1 \leq i \leq q)$, course C_i consists of k_i lectures. There are r curricula $S_1, S_2, S_3, \ldots, S_r$, which are groups of courses that have common students; $S_l \subseteq C$ for each $l \quad (1 \leq l \leq r)$. This means that all courses in S_l must be scheduled at non-overlapping times. The number of periods is p, and l_t is the maximum number of courses that can be scheduled at period t (i.e., the number of rooms available at period t). The formulation is the following: find $y_{it} (i = 1, \ldots, q;\ t = 1, \ldots, p)$, so that*

$$\sum_{t=1}^{p} y_{it} = k_i, \qquad i = 1, \ldots, q$$

$$\sum_{i=1}^{q} y_{it} \leq l_t, \qquad t = 1, \ldots, p$$

$$\sum_{i \in S_l} y_{it} \leq 1, \qquad t = 1, \ldots, p; \quad l = 1, \ldots, r$$

$$y_{it} \in \{0, 1\}, \qquad i = 1, \ldots, q; \quad t = 1, \ldots, p$$

where y_{it} = 1 if a lecture of course C_i is scheduled at period t, and y_{it} = 0 otherwise.

Here, the first constraint ensures that each course is composed of the correct number of lectures. The second one imposes that at any period the number of lectures are not more than available rooms. The third constraint prevents the conflicting lectures to be scheduled at the same period.

The course timetabling problem as stated in definition 3.5 includes the following objective function [3, 7].

Definition 3.6 (Course timetabling objective function) *The function*

$$f(y) = \sum_{i=1,\ldots,q} \sum_{t=1,\ldots,p} d_{it} \times y_{it}$$

where d_{it} is the desirability of scheduling a lecture of course C_i at period t.

3.4 Approaches to Solve Timetabling Problems

Timetabling problems have been traditionally solved using greedy heuristics, meta-heuristics such as hill climbing and tabu search, integer linear programming [36], and more recently using constraint programming techniques [23]. Cambazard et al. [13] proposed an algorithm for solving course timetabling problem using constraint programming that allows automatic relaxations of constraints. Lotfi, Vahid and Cerveny [43] introduced a multi-phase algorithm to solve a final exam scheduling problem.

Edmund K. Burke and Sanja Petrovic [10] present a rough categorization of different approaches to the timetabling problems as follows.

3.4.1 Sequential Approach

In sequential methods, at first the events are ordered using domain heuristics. Then these events are assigned to time periods so that they don't conflict with each other [21]. Timetabling problems are usually modeled as graph coloring problem, where events are represented as vertices. Conflicts between events are represented by edges between the vertices corresponding to the conflicting events. This graph may be colored in such a way that no adjacent vertices have same color. Here the colors represent the time periods. Thus a feasible coloring of the graph ensures a conflict free schedule.

Course Timetabling Reduces to Graph Coloring

Course timetabling problem (as defined in Definition 3.5) can be reduced to graph coloring problem, which is a well-known NP-Hard problem. De Werra [7] presents the following reduction, which proves that the timetabling problem (as defined in Definition 3.5) is NP-hard.

For each lecture l_i of each course C_j take a introduce a vertex m_{ij}. For each course C_j produce a clique among vertices m_{ij} (for i = 1, ... , q). Introduce all edges between the clique for C_{j1} and the clique for C_{j2} whenever C_{j1} and C_{j2} are conflicting.

In case of unavailability, introduce a set of p new vertices all connected to each other. Each of these new vertices corresponds to a period. This complete connectivity among these new p vertices enforces each vertex taking a different color. If a lecture cannot be scheduled at a given period, then the vertex corresponding to that lecture is also connected to the vertex corresponding to that period. Conversely, if a lecture must take place at a given period, then the vertex corresponding to that lecture is connected to all of the p vertices except the one corresponding to that very period.

3.4.2 Cluster Approach

Cluster method divides the events into groups, which satisfy the hard constraints and then the groups are assigned to time periods to satisfy the soft constraints. For example, the approach used in [43] for scheduling final exams may be considered as a cluster approach. In this approach, solution may be found quickly but in some cases they may result in poor timetable if there are much dependencies between events of different clusters.

3.4.3 Heuristic and Meta-heuristic Methods

A wide variety of heuristics (greedy algorithms, hill climbing, etc.), meta-heuristic methods (such as simulated annealing, tabu search, genetic algorithms, etc.), and hybrid approaches are available.

At every iteration of the greedy algorithm a single event is assigned the best possible time period, which does not conflict with the already assigned events. This method works efficiently for the earlier events, but may cause conflicts for the later events. Further, once the schedule is constructed it may not be possible to include a new event in the schedule.

Meta-heuristic methods begin with one or more initial solutions and employ search strategies that try to avoid local optima. It takes a feasible solution and tries to find an optimal solution by iteratively searching in its neighborhood. A neighborhood is a set of feasible solutions obtained by changing be one parameter of the current solution.

Heuristic and meta-heuristic methods can produce high quality solutions but may have higher computational overhead.

3.4.4 Constraint Based Approaches

Constraint based approaches consider the timetabling problem as a constraint satisfaction problem. The problem is usually modeled as a set of variables and their domains (values). Typically variables represent the events and the values represent the resources such as, time periods, rooms, etc. Variables are to be assigned to the suitable values, satisfying a number of constraints [3, 32, 35]. Usually a number of rules are defined for assigning values to the variables. Backtracking occurs, when no rule is found applicable to the current partial solution [42].

3.5 Summary

University level course timetabling has to deal with a large number and types of participants and resources, each of which adds constraints and preferences to the problem. Consequently, the problem has considerably large number of constraints of various types. Therefore, university level timetabling is typically more complicated than other academic timetabling problems. Moreover, the problem instances of university course timetabling are usually large involving hundreds of courses, hundreds of instructors, thousands of students, and so on. Unfortunately such problems belong to the complexity class "NP-hard". So, the efficiency in algorithm design and careful implementation plays a vital role in solving such problems.

Chapter 4

UofL Timetabling Problem

This chapter illustrates the issues specific to the course timetabling problem at the University of Lethbridge. In section 4.1 we introduce some terminology used to describe the problem and solution approach. Section 4.2 depicts the problem domain and the parameters. Section 4.2.1 illustrates the set of constraints applicable to the course timetabling problem at the University of Lethbridge. In section 4.3 we describe our multi-phase approach to solve the problem. Finally, section 4.4 concludes the chapter with a summary.

4.1 Terminology

We use the following terminology to describe the problem and solution strategies.

Course: A course is a set of related lectures, labs and tutorials. For example, "CS1000: Computer Basics" is a course, which may include

- two lectures: CS1000 - Section A and CS1000 - Section B, or two hours of lecture.
- three labs: CS1000 - Lab1, CS1000 - lab2 and CS1000 - Lab3,
- two tutorials: CS1000 - Tutorial1 and CS1000 - Tutorial2.

Instruction-unit: An instruction-unit refers to a single lecture or lab or tutorial.

Section: A course may be divided into one or more sections. Each section consists of one or more lectures, labs, and tutorials. For example, the course CS1000 may have two sections: section-A has one lecture, one lab, and a tutorial. Section-B has one

lecture, two labs, and one tutorial. Typically, all instruction-units belonging to the same section of a course needs to be scheduled in different times.

Instructor: An instructor is a person who teaches/conducts an instruction-unit. There may be different types of instructors, such as Professors, Academic Assistants and Graduate Assistants (Teaching Assistants).

Week-day: A week-day is a working day of a week from Monday through Friday.

Day-Sequence: A day-sequence is a set of week-days. Week-days are divided into two day-sequences: MWF (Monday, Wednesday and Friday) and TR (Tuesday and Thursday).

Time-slot: A time-slot refers to the unit of time-span specified by starting and ending time. Each week-day is divided into a fixed number of time-slots enumerated as 1, 2, 3, and so on. Each time-slot may be allotted to courses to be taught during that period. Duration of 3 time-slots in MWF equals to the duration of 2 time-slots in TR. Each time-slot in MWF is 50 minutes long, whereas each time-slot TR spans 75 minutes . Therefore, a lecture of a 3-credit course if scheduled in MWF gets 150 minutes teaching time in three 50-minute time-slots one on each of Monday, Wednesday, and Friday. Similarly, such a lecture when scheduled in TR also gets 150 minutes teaching time in two 75-minute time-slots one on Tuesday and the other on Thursday.

Overlapping time-slot: Two time-slots of the same day are said to be overlapped if the starting time and ending time of one is the same as those of the other.

Teaching capacity: Teaching capacity of any instructor is the maximum number of instruction-units he or she may teach.

4.2 The Problem

One year prior to the start of an academic semester, each academic department at the University of Lethbridge determines the courses to be offered in that semester. Each department is allotted a number of classrooms and time-slots available for the courses offered they offer.

Within an academic department, every instructor submits 3 types of preferences such as,

i) set of courses the instructor wants to teach ordered according to preferences. For example, an instructor may mention his/her preferred courses as follows:

   1st preference: {course-1, course-5, course-8}

   2nd preference: {course-2, course-4}

   3rd preference: {course-3, course-6, course-7, course-9}

ii) preferred day-sequence (either MWF or TR)

iii) time preference (either morning or evening)

The department prepares a timetable assigning courses to instructors, days or day-sequences and time-slots respecting the individual preferences as much as possible. The exact procedure may vary across academic departments but the procedure described above can be regarded as a general procedure that a typical academic department follows.

After the timetable is prepared within each department, the Dean's office accumulate them resolving any remaining conflicts if any. And finally the timetable is made available to the students to choose courses and register for their chosen courses.

Our work is mainly on the intra-department timetabling problem that every department regularly deals with.

4.2.1 Constraints

There are a number of common constraints (hard constraints) applicable to course timetabling problem in any university. Such general constraints applicable to the University of Lethbridge include the following.

- No instructor teaches more than one course at a given time.
- No two courses are taught in the same class-room at a given time.
- The number of courses taught during any time-slot must not exceed the maximum number of classrooms available during that time-slot.

Besides these common constraints, depending on the institution, additional constraints and preferences (soft constraints) also apply. The constraints and preferences specific to the timetabling problem at the University of Lethbridge are as follows.

- Professors conduct only lectures, academic assistants conduct labs and tutorials, and graduate students conduct only labs. Each instructor has an upper limit on the number of courses he or she may teach. In practice each instructor is assigned a fixed number of instruction-units in a given semester.
- A lecture has to be scheduled in one of the two day-sequences. When a lecture is assigned to a day-sequence, it is taught at the same time on each day of the day sequence.
- A lab or tutorial has to be assigned to only one of the five week-days.
- There should be a gap of at least one time-slot between courses taught by the same instructor.

- Instruction-units (lectures, labs and tutorials) belonging to the same section should not be scheduled in overlapping time-slots.

4.2.2 Desired Solution

A *solution* to the course timetabling problem at the University of Lethbridge is a complete assignment of instruction-units to instructors, days or day-sequences and time-slots, such that

- All the hard constraints are satisfied.
- Instructor's preferences (soft constraints) are satisfied as much as possible.

In general a feasible schedule may be said to be good if it satisfies much of the preferences (on course, day, and time) given by the instructors. For each phase, quality is measured by the difference between the value of the objection function and its upper bound. Computation of the upper bound on the objective function's value is illustrated in section 4.3. The basic idea is to relax some of the constraints and determine the objective functions value of this relaxed problem instance. This upper bound facilitates a way to evaluate the quality of a solution measuring how far the achieved objective value is from its upper bound. The lower the difference, the better the achieved solution is.

High quality solutions are desired to be computed without taking much computation time.

4.3 Multi-phase Approach

The solution approach we propose for solving the course timetabling problem splits the entire problem into four major subproblems, which are then solved in separate phases.

This careful decomposition of the large problem allows us to work at a time on a smaller subproblem of reduced complexity.

In usual circumstances, a lecture is taught on Tuesday and Thursday, or on Monday, Wednesday, and Friday. But labs and tutorials are usually not scheduled in this pattern. A lecture may not occupy more than one time-slot but labs and tutorials usually span longer. Moreover, labs must be scheduled in particular type of rooms having certain facilities. Therefore, we handle labs and tutorials separately from the lectures. Such a clustering leads to decomposition of the problem into four major phases.

In phase-1 we assign instruction-units (lectures, labs, and tutorials) to the instructors (professors, academic assistants and graduate students). Phase-2 assigns lectures to the day-sequences. In phase-3, lectures of a single day-sequence are assigned to the time-slots available on that day-sequence. And finally, phase-4 assigns labs and tutorials to week-days and available time-slots within the week-days. At each phase, the objective is to maximize the concerned preferences (given as an objective function) as much as possible while satisfying the given constraints.

The ordering of the phases may be altered based on structure of the problem instance. For example, if there are considerably large number of labs or tutorials needing consecutive time-slots, then solving phase-4 before phase-2 may result in better schedule as well as reduced overall computation time. However, phase-2 must precede phase-3 and phase-2 should be solved at the beginning of the scheduling process.

4.3.1 Terminology

We use the following variables and sets used in the mathematical formulations of the problems concerned with different phases.

$Prof$ is the set of all professors.

$AcadAsst$ is the set of all academic assistants.

$GradStud$ is the set of all graduate students doing teaching assistantship.

I is the set of all instructors. Hence $I = Prof \cup AcadAsst \cup GradStud$.

$Lectures$ is the set of all lectures from all courses.

$Labs$ is the set of all labs from all courses.

$Tutorials$ is the set of all tutorials from all courses.

C is the set of all instruction-units. Hence, $C = Lectures \cup Labs \cup Tutorials$.

$AllSec$ is the set of sections over all courses.

$teach_{ic}$ is 1 if instruction-unit c is assigned to instructor i, and 0 otherwise.

$DaySeq$ is the set of day-sequences.

T_D is total the number of time-slots available in a day-sequence D.

R_D is the total number of classrooms available in a day of day-sequence D.

R_{dt} is the number of time-slots available in time-slot t of day d.

$TimeSlots_d$ is the set of time-slots available in day d.

$Slots_D$ is the set of time-slots available in a day of day-sequence D.

Lec_D is the set of lectures assigned to day-sequence D.

$Duration_c$ is the duration of instruction-unit c in terms of number of time-slots.

$Limit_i$ is the maximum number of courses may be assigned to instructor i.

4.3.2 Phase-1: Assigning Courses to Instructors

Phase-1 assigns instruction-units (lectures, labs and tutorials) to instructors (professors, academic assistants and graduate students) based on instructors' preferences on courses.

The mathematical formulations of phase-1 problem use the decision variable, X_{ic}, and the weight, W_{ic}. Here, $X_{ic} = 1$ if instructor i is assigned to instruction-unit c, and 0 otherwise. W_{ic} is the preference level of instructor i on instruction-unit c.

$$W_{ic} = \begin{cases} 4 & \text{if instructor } i \text{ has highest preference on instruction-unit } c \\ 3 & \text{if instructor } i \text{ has medium preference on instruction-unit } c \\ 2 & \text{if instructor } i \text{ has low preference on instruction-unit } c \\ 1 & \text{if instructor } i \text{ has no preference on instruction-unit } c \end{cases}$$

In this weighting scheme, we use a linear weight W_{ic}. The values of W_{ic} differs monotonously. So, the model does not become biased on any specific level of preference. This scheme may become biased to the instructors for whom the preferred courses include instruction-units equal to the maximum number of instruction-units they may teach. However, this issue is handle in the implementation level. When an instructor mention a course that he or she wants to teach, based on the type of the instructor all the appropriate instruction-units (lectures, labs, or tutorials) belonging to the course become the instruction-units preferred by the instructor. For example, professor p wants to teach course c, which has 3 lectures. Since, the instructor is a professor, all of these 3 lectures become his or her preferred instruction-units. Thus, eventually for each instructor we no longer have a very narrow range of instruction-units that he or she wants to teach.

Given below is the mathematical formulation of the phase-1 problem.

$$\textit{maximize} \quad \sum_{i \in I} \sum_{c \in C} X_{ic} W_{ic} \tag{4.1}$$

Subject to,

$$\sum_{c \in C} X_{ic} \leq Limit_i, \quad i \in I \tag{4.2}$$

$$\sum_{i \in I} X_{ic} = 1, \quad c \in C \tag{4.3}$$

$$X_{ic} = 0, \quad i \in Prof, c \in Labs \cup Tutorials \tag{4.4}$$

$$X_{ic} = 0, \quad i \in AcadAsst, c \in Lectures \tag{4.5}$$

$$X_{ic} = 0, \quad i \in GradStud, c \in Lectures \cup Tutorials \tag{4.6}$$

Here,

The objective function (Equation 4.1) satisfies the instructors' preferences on instruction-units as much as possible.

Equation 4.2 imposes that the number of instruction-units assigned to any instructor must not exceed the maximum capacity of the instructor.

Equation 4.3 ensures that each instruction-unit is assigned to exactly one instructor.

Equation 4.4 enforces that the the professors are not assigned to labs or tutorials.

Equation 4.5 imposes that the academic assistants are not assigned to lectures.

Equation 4.6 ensures that the graduate students are not assigned to lectures or tutorials.

Typically, professors teach only lectures, while academic assistants and graduate students do not teach lectures. Lab instructors usually teach more hours than professors. So, lectures and professors may further be handled separately. Therefore, we further split the

phase-1 problem into two independent subproblems: phase-1a for assigning lectures to professors, and phase-1b for assigning labs and tutorials to academic assistants and graduate students.

Phase-1a: Assigning Lectures to Professors

Phase-1a assigns lectures to professors. The mathematical formulation of phase-1a problem is given below,

$$\textit{maximize} \qquad \sum_{i \in Prof} \sum_{c \in Lectures} X_{ic} W_{ic} \tag{4.7}$$

subject to,

$$\sum_{c \in Lectures} X_{ic} \leq Limit_i, \qquad i \in Prof \tag{4.8}$$

$$\sum_{i \in Prof} X_{ic} = 1, \qquad c \in Lectures \tag{4.9}$$

Here,

The objective function (Equation 4.7) satisfies the professors' preferences on lectures as much as possible.

Equation 4.8 imposes that the number of lectures assigned to any professor must not exceed the maximum capacity of the professor.

Equation 4.9 ensures that each lecture is assigned to exactly one professor.

Phase-1b: Labs and Tutorials to Instructors

In phase-1b, labs and tutorials are assigned to academic assistants and graduate students (teaching assistants). The mathematical formulation of phase-1b problem is as follows.

$$\textit{maximize} \quad \sum_{i \in AcadAsst \cup GradStud} \; \sum_{c \in Labs \cup Tutorials} X_{ic} W_{ic} \tag{4.10}$$

subject to,

$$\sum_{c \in Labs \cup Tutorials} X_{ic} \leq Limit_i, \qquad i \in AcadAsst \cup GradStud \tag{4.11}$$

$$\sum_{i \in AcadAsst \cup GradStud} X_{ic} = 1, \qquad c \in Labs \cup Tutorials \tag{4.12}$$

Here,

The objective function (Equation 4.10) satisfies the preferences of academic assistants and graduate students on labs and tutorials as much as possible.

Equation 4.11 imposes that the number of labs or tutorials assigned to any academic assistant or graduate student must not exceed his/her maximum capacity.

Equation 4.12 ensures that each lab or tutorial is assigned to exactly one academic assistant or graduate student.

Phase-1 Heuristic

Those instructors who have higher teaching capacity are likely to be assigned to more instruction-units, and consequently may cause more conflicts. Therefore, in phase-1 (phase-1a and phase1-b) we first try to assign instruction-units to those instructors who have higher teaching capacity.

Upper Bound for Phase-1 Objective Value

Consider two instructors: instructor A with teaching capacity 3, and instructor B with teaching capacity 2. The sets of instruction-units preferred by A and B are given as

$P_A = \{c_1, c_2, c_3, c_4, c_5\}$ and,

$P_B = \{c_6, c_7, c_8, c_9\}$

Suppose the instruction-units as shown are given in the order of preferences. Assuming that A is assigned $\{c_1, c_2, c_3\}$ and B is assigned $\{c_6, c_7\}$, phase-1 obtains the highest objective value. Hence, an upper bound for the phase-1 objective function may be calculated in the following way.

Let us consider the following two assumptions:

- instructors' preferences on instruction-units are mutually exclusive, i.e., no two instructors wants to teach the same instruction-unit.
- there are sufficient instruction-units so that each instructor may be assigned the number of instruction-units equal to the maximum number of instruction-units he or she can teach.

If these two assumptions holds for an instance of the phase-1 problem, the optimal solution is expected to have assigned each instructor to only those instruction-units that he or she wants to teach, without exceeding the instructors' teaching capacity. The objective value of this solution may be regarded as an upper bound on the objective value of the actual problem. However, such an upper bound is not a very good one. Nonetheless, this upper bound gives a way to measure how good the optimal solution is. The closer the optimal objective value to its upper bound, the better the optimal solution is. Algorithm 1 depicts how we compute this upper bound on the objective function's value of phase-1.

In this algorithm, for phase-1a the set of instructors *Inst* include only professors, and

Algorithm 1 Computing upper bound for phase-1 objective function

```
1: Inst = the set of instructors,
2: pref_li = the number of instruction-units on which instructor i has preference level l
3: procedure ComputePhase1MaxObj
4:     maxObj ← 0
5:     for all i ∈ Inst do
6:         n ← limit_i
7:         for l ← 0,...,2 do                    ▷ 3 levels of preferences 0 through 2
8:             if n ≤ pref_li then
9:                 maxObj ← maxObj + n(4 − l)          ▷ the highest value of W_ic is 4
10:            else
11:                n ← n − pref_li
12:                maxObj ← maxObj + (4 − l)pref_li
13:            end if
14:        end for
15:        if n > 0 then
16:            maxObj ← maxObj + n
17:        end if
18:    end for
19:    return maxObj
20: end procedure
```

for phase-1b *Inst* include only the academic assistants and graduate students.

4.3.3 Phase-2: Assigning Lectures to Day-sequences

In phase-2 lectures are assigned to one of the two day-sequences while satisfying the instructors' preferences on day-sequence as much as possible. The mathematical formulation is given below. Here the decision variable $X_{cD} = 1$ if lecture c is assigned to day-sequence D, and 0 otherwise.

W_{cD} is the weight function, which gets higher value when preference of the concerned professor is satisfied. Its value depends on whether or not the assignment of lecture c to day-sequence D violates the concerned professor's (who teaches lecture c) preference on day-sequence.

$$W_{cD} = \begin{cases} 3 & \text{if professor's preference on day-sequence is satisfied} \\ 2 & \text{if professor has no preference on day-sequence} \\ 1 & \text{if professor's preference on day-sequence is violated} \end{cases}$$

$$\textit{maximize} \qquad \sum_{D \in DaySeq} \sum_{c \in Lectures} X_{cD} W_{cD} \tag{4.13}$$

subject to,

$$\sum_{c \in Lectures} X_{cD} \leq T_D R_D, \qquad D \in DaySeq \tag{4.14}$$

$$\sum_{D \in DaySeq} X_{cD} = 1, \qquad c \in Lectures \tag{4.15}$$

Here,

The objective function (Equation 4.13) satisfies professors' preferences on day-sequences

as much as possible.

Equation 4.14 restricts that the number of lectures assigned to a day-sequence does not exceed the number of available time-slots in that day-sequence.

Equation 4.15 ensures that each lecture is assigned to exactly one day-sequence.

Phase-2 Heuristic

In phase-2, preference violation applies only to those professors who gave preferences on day-sequences. Therefore, first we try to assign day-sequences to lectures taught by such professors. Then we assign day-sequences to the lectures taught by professors who don't have preferences on day-sequences.

Upper Bound for Phase-2 Objective Value

The objective function's value may achieve the maximum possible value if, for each professor, all lectures taught by him or her are scheduled in his or her preferred day-sequence. We assume that each professor is assigned to the number lectures equal to his or her teaching capacity. Then we compute the upper bound on the phase-2 objective value using the following expression.

$$\sum_{p \in Prof} Limit_p \times pref_p$$

where,

$$pref_p = \begin{cases} 3 & \text{if professor } p \text{ has preference on day-sequence} \\ 2 & \text{if professor } p \text{ has no preference on day-sequence} \end{cases}$$

4.3.4 Phase-3: Assigning Lectures to Time-slots

After phase-2 assigns lectures to day-sequences, phase-3 operates on lectures of a single day-sequence at a time and assigns them to available time-slots. The constraint programming (CP) and integer programming (IP) formulation of phase-3 problem are given in the following sections 4.3.4 and 4.3.4.

Here the decision variable $X_{ct} = 1$ if lecture c is assigned to time-slot t, and 0 otherwise.

W_{ct} is the weight function, which gets higher value if the preference of the concerned professor is satisfied. The value of W_{ct} depends on whether or not the assignment of lecture c to time-slot t violates the concerned professor's (who teaches lecture c) preference on time (morning or afternoon).

$$W_{ct} = \begin{cases} 3 & \text{if professor's preference on time is satisfied} \\ 2 & \text{if professor has no preference on time} \\ 1 & \text{if professor's preference on time is violated} \end{cases}$$

Let $D \in DaySeq$ is the current day-sequence on which phase-3 operates.

Phase-3: CP Formulation

Constraint programming (CP) formulation of phase-3 problem is as follows.

$$\textit{maximize} \quad \sum_{c \in Lec_D} \sum_{t \in Slots_D} X_{ct} W_{ct} \tag{4.16}$$

subject to,

$$\sum_{t \in Slots_D} X_{ct} = 1, \quad c \in Lec_D \tag{4.17}$$

$$\sum_{c \in Lec_D} X_{ct} \leq R_D, \quad t \in Slots_D \tag{4.18}$$

$$X_{c_1 t} X_{c_2 t} = 0, \quad S \in AllSec;\ c_1, c_2 \in S;\ t \in Slots_D \tag{4.19}$$

$$X_{c_1 t_1} teach_{ic_1} X_{c_2 t_2} teach_{ic_2} = 1 \Rightarrow |t_1 - t_2| > 1, \quad c_1, c_2 \in Lec_D;\ t_1, t_2 \in Slots_D \tag{4.20}$$

Here,

The objective function (Equation 4.16) satisfies professors' preferences on time as much as possible.

Equation 4.17 enforces that each lecture is assigned to exactly one time-slot.

Equation 4.18 restricts that the number of lectures assigned to any time-slot does not exceed the number of available classrooms during that period.

Equation 4.19 enforces that the lectures of the same section are not scheduled in overlapping times-lots.

Equation 4.20 ensures at least one time-slot gap between lectures of the same instructor.

Phase-3: IP Formulation

The integer programming formulation of phase-3 problem is given below.

$$\textit{maximize} \quad \sum_{c \in Lec_D} \sum_{t \in Slots_D} X_{ct} W_{ct} \tag{4.21}$$

subject to,

$$\sum_{t \in Slots_D} X_{ct} = 1, \quad c \in Lec_D \tag{4.22}$$

$$\sum_{c \in Lec_D} X_{ct} \leq R_D, \quad t \in Slots_D \tag{4.23}$$

$$X_{c_1 t} + X_{c_2 t} \leq 1, \quad S \in AllSec;\ c_1, c_2 \in S;\ t \in Slots_D \tag{4.24}$$

$$7(X_{c_1 t_1} teach_{ic_1} + X_{c_2 t_2} teach_{ic_2}) \leq 12 + |t_1 - t_2|, \quad c_1, c_2 \in Lec_D;\ t_1, t_2 \in Slots_D \tag{4.25}$$

Here,

The objective function (Equation 4.21) satisfies professors' preferences on time as much as possible.

Equation 4.22 ensures that each lecture is assigned to exactly one time-slot.

Equation 4.23 restricts that the number of lectures assigned to any time-slot does not exceed the number of available classrooms during that period.

Equation 4.24 enforces that the lectures of the same section are not scheduled in overlapping times-lots.

Equation 4.25 enforces at least one time-slot gap between lectures of the same instructor.

Phase-3 Heuristic

In phase-3, the question of preference violation applies only to those professors who gave preferences on time (morning or afternoon). So, for avoiding preference violation we first try to assign time-slots to lectures taught by such professors. Then we assign time-slots to the lectures taught by professors who don't have preferences on time.

Upper Bound for Phase-3 Objective Value

In phase-3 the objective function's value may achieve the maximum possible value if, for each professor, all lectures taught by him or her are scheduled in his or her preferred time (morning or afternoon). Again, assuming that each professor is assigned to the number lectures equal to his or her teaching capacity, we compute the upper bound on the phase-3 objective value using the following expression.

$$\sum_{p \in Prof} Limit_p \times pref_p$$

where,

$$pref_p = \begin{cases} 3 & \text{if professor } p \text{ has preference on time} \\ 2 & \text{if professor } p \text{ has no preference on time} \end{cases}$$

4.3.5 Phase-4: Labs and Tutorials to Days and Time

Phase-4 is the last phase. In this phase, labs and tutorials are assigned to one of the five week-days and to available time-slots within the days.

First we present the constraint programming (CP) formulation of the phase-4 problem. Then a linear integer programming (ILP) formulation is also given. In the formulations

the decision variable $X_{cdt} = 1$ if instruction-unit c is assigned to time-slot t of day d, and 0 otherwise; $X_{cdt} = 1$ if instruction-unit c is scheduled in time-slot t of day d, and 0 otherwise.

W_{ct} is the weight function, which gets higher value if the concerned instructor's preference on time is satisfied. Phase-4 takes care of only the instructors' preferences on time (morning or afternoon).

The value of W_{ct} depends on whether or not the assignment of instruction-unit c to time-slot t violates the concerned instructor's (who teaches instruction-unit c) preference on time (morning or afternoon).

$$W_{ct} = \begin{cases} 3 & \text{if instructor's preference on time is satisfied} \\ 2 & \text{if instructor has no preference on time} \\ 1 & \text{if instructor's preference on time is violated} \end{cases}$$

Phase-4: CP Formulation

The constraint programming (CP) formulation of the phase-4 problem is as follows.

$$maximize \quad \sum_{c \in Labs \cup Tutorials} \sum_{d \in Days} \sum_{t \in TimeSlots_d} X_{cdt} W_{ct} \tag{4.26}$$

subject to,

$$\sum_{d \in Days} \sum_{t \in TimeSlots_d} X_{cdt} = Duration_c, \quad c \in Labs \cup Tutorials \tag{4.27}$$

$$\sum_{c \in Labs \cup Tutorials} X_{cdt} \leq R_{dt}, \quad d \in Days, t \in TimeSlots_d \tag{4.28}$$

$$X_{cdt_1} X_{cdt_2} |t_1 - t_2| \leq 1, \quad c \in Labs \cup Tutorials;\ d \in Days;\ t_1, t_2 \in TimeSlots_d \tag{4.29}$$

$$X_{cd_1t_1} \times X_{cd_2t_2} \times |d_1 - d_2| = 0,$$

$$c \in Labs \cup Tutorials;\ d_1, d_2 \in Days;\ t_1 \in TimeSlots_{d_1};\ t_2 \in TimeSlots_{d_2} \quad (4.30)$$

$$X_{c_1dt_1} teach_{ic_1} X_{c_2dt_2} teach_{ic_2} = 1 \Rightarrow |t_1 - t_2| > 1,$$

$$c_1, c_2 \in Labs \cup Tutorials;\ d \in Days;\ t_1, t_2 \in TimeSlots_d \quad (4.31)$$

$$X_{c_1dt} X_{c_2dt} = 0, \qquad S \in AllSec;\ c_1, c_2 \in S;\ d \in Days;\ t \in TimeSlots_d \quad (4.32)$$

$$X_{c_1dt} Y_{c_2dt} = 0,$$

$$S \in AllSec;\ c_1 \in S \cap (Labs \cup Tutorials);\ c_2 \in S \cap Lectures;\ d \in Days;\ t \in TimeSlots_d \quad (4.33)$$

Here,

The objective function (equation 4.26) satisfies instructors' preferences on time as much as possible.

Equation 4.27 ensures that each lab or tutorial is assigned to the number of time-slots exactly equal to its duration.

Equation 4.28 ensures that the number of labs and tutorials assigned to any time-slot does not exceed the number of classrooms available during that period.

Equation 4.29 restricts each lab or tutorial to be assigned to consecutive time-slots.

Equation 4.30 imposes that each lab or tutorial is assigned to time-slots of the same day.

Equation 4.31 imposes that there should be a gap of at least one time-slot between courses taught by the same instructor.

Equation 4.32 enforces that the labs and tutorials of the section are not scheduled in overlapping time-slots.

Equation 4.33 ensures that labs and tutorials are scheduled in time-slots not overlapping with the lectures of the same section.

Phase-4: IP Formulation

The integer programming (IP) formulation of the phase-4 problem is given below.

$$\textit{maximize} \qquad \sum_{c \in Labs \cup Tutorials} \sum_{d \in Days} \sum_{t \in TimeSlots_d} X_{cdt} W_{ct} \quad (4.34)$$

subject to,

$$\sum_{d \in Days} \sum_{t \in TimeSlots_d} X_{cdt} = Duration_c, \qquad c \in Labs \cup Tutorials \quad (4.35)$$

$$\sum_{c \in Labs \cup Tutorials} X_{cdt} \leq R_{dt}, \qquad d \in Days, t \in TimeSlots_d \quad (4.36)$$

$$2 - X_{cd_1t_1} - X_{cd_2t_2} + |d_1 - d_2| \leq 4(2 - X_{cd_1t_1} - X_{cd_2t_2}),$$
$$c \in Labs \cup Tutorials;\ d_1, d_2 \in Days;\ t_1 \in TimeSlots_{d_1};\ t_2 \in TimeSlots_{d_2} \quad (4.37)$$

$$1 - X_{cd_1t_1} - X_{cd_2t_2} + |t_1 - t_2| \leq 7(2 - X_{cd_1t_1} - X_{cd_2t_2}),$$
$$c \in Labs \cup Tutorials;\ d_1, d_2 \in Days;\ t_1 \in TimeSlots_{d_1};\ t_1 \in TimeSlots_{d_2} \quad (4.38)$$

$$7(X_{c_1dt_1} teach_{ic_1} + X_{c_2dt_2} teach_{ic_2}) \leq 12 + |t_1 - t_2|,$$
$$c_1, c_2 \in Labs \cup Tutorials;\ t_1, t_2 \in TimeSlots_d; \quad (4.39)$$

$$X_{c_1dt} + X_{c_2dt} \leq 1, \qquad S \in AllSec;\ c_1, c_2 \in S;\ d \in Days;\ t \in TimeSlots_d \quad (4.40)$$

$$X_{c_1dt} + Y_{c_2dt} \leq 1,$$
$$S \in AllSec; c_1 \in S \cap (Labs \cup Tutorials);\ c_2 \in S \cap Lectures;\ d \in Days;\ t \in TimeSlots_d \quad (4.41)$$

Here,

The objective function (equation 4.34) satisfies instructors' preferences on time as much as possible.

Equation 4.35 ensures that each lab or tutorial is assigned to the number of time-slots exactly equal to its duration.

Equation 4.36 ensures that the number of labs and tutorials assigned to any time-slot does not exceed the number of classrooms available during that period.

Equation 4.37 imposes that each lab or tutorial is assigned to time-slots of the same day.

Equation 4.38 restricts each lab or tutorial to be assigned to consecutive time-slots.

Equation 4.39 imposes that there should be a gap of at least one time-slot between courses taught by the same instructor.

Equation 4.40 enforces that the labs and tutorials of the section are not scheduled in overlapping time-slots.

Equation 4.41 ensures that labs and tutorials are scheduled in time-slots not overlapping with the lectures of the same section.

Phase-4 Heuristic

In phase-4 we assign labs and tutorials to days and time-slots. Most of the labs and some of the tutorials typically may need more than one time-slots. Such time-slots to be consecutive need to be of the same day as well as consecutive. This makes the phase-4 problem hard. Experiments (see chapter 6) show that solving phase-4 takes much time in comparison to the other phases. For large problem instance this time requirement becomes a serious issue. In such circumstances we may be interested to find a feasible solution (rather than the optimal) within a given time bound. However, we want this feasible solution to be close to the optimal as much as possible. To enforce this, we use the heuristic presented in algorithm 2 for ordering the variables.

Since, labs are more likely to need more than one consecutive time-slots, we first assign labs to days and time-slots. All labs are ordered according to decreasing duration (number of time-slots needed). Days are also ordered decreasingly in terms of the number of available time-slots in each day. Further, time-slots are also ordered decreasingly according to the number of available classrooms available during that periods. From all the unassigned labs we pick the one with highest duration. From the available days we choose the one having maximum number of available time-slots. Then from the available time-slots of

our chosen day we select the one during which maximum number of classrooms available. Then we try to assign that lab to the selected time-slot of our chosen day. If this assignment does not conflict with previous assignment, we commit. If conflict arises, we choose the next available time-slot of our chosen day and try to make the assignment. If none of the time-slots of our selected day allows conflict free assignment, backtrack and pick the next available day and try to make the assignment selecting time-slots one after another. None of the available days allows conflict free assignment, we undo some previous assignment and continue iterating the process mentioned above until all labs are assigned.

After all labs are assigned, we assign the tutorials following the same procedure.

Upper Bound for Phase-4 Objective Value

In phase-4 instructors' preferences on time (morning or afternoon) are taken into account. The objective function's value may achieve the maximum possible value if, for each instructor, all labs and tutorials taught by him or her are scheduled in his or her preferred time (morning or afternoon). We assume that each instructor is assigned to the number labs and tutorials equal to his or her teaching capacity. Then using the following expression we compute the upper bound on the phase-4 objective value.

$$\sum_{c \in Labs \cup Tutorials} duration_c \times \rho$$

where, ρ is the value associated with an instructor's highest level of preference on time. In our implementation $\rho = 3$.

Algorithm 2 Phase-4 Heuristic

```
1: Labs = set of all labs.
2: Tutorials = set of all tutorials.
3: Days = set of all weekdays.
4: T_d = set of all time-slots in day d.
5: R_dt = set of rooms during the time-slot t of day d.
6: for all c ∈ Labs ordered by decreasing Duration_c do
7:     for all d ∈ Days ordered by decreasing |T_d| do
8:         for all t ∈ T_d ordered by decreasing |R_dt| do
9:             if assignment of c to d and t does not conflict then
10:                assign of c to d and t.
11:            else
12:                unassign c from d and t.
13:            end if
14:        end for
15:    end for
16: end for
17: for all c ∈ Tutorials ordered by decreasing Duration_c do
18:    for all d ∈ Days ordered by decreasing |T_d| do
19:        for all t ∈ T_d ordered by decreasing |R_dt| do
20:            if assignment of c to d and t does not conflict then
21:                assign of c to d and t.
22:            else
23:                unassign c from d and t.
24:            end if
25:        end for
26:    end for
27: end for
```

4.4 Summary

In this chapter we described the course timetabling problem at the University of Lethbridge, which is currently solved more or less manually! We apply a multi-phase approach for solving the problem.

Practical course timetabling problem instances are all usually quite large. Our multi-phase approach splits the entire problem into several subproblems, each of which are solved separately. Thus, at a time we deal with a small subproblem with reduced complexity (in terms of number of concerned resources and constraints), which is easier to solve. This decomposition also allows to apply different solution techniques as appropriate to each subproblem. For phase-1 and phase-2 we use ILP whereas for phase-3 and phase-4 we have separate CP and ILP implementations. Moreover, after each phase the solution to the concerned subproblem may be examined. The user may decide to proceed on to the next phase or recompute the current phase after fine tuning. This makes the multi-phase implementation more interactive and allows better utilization of users' expertise.

Chapter 5

Implementation Detail

In this chapter we present the software implementation detail of our multi-phase approach to solve the course timetabling problem. Section 5.1 describes the data format that our software implementation takes as input. In section 5.2 we describe the architecture of our software implementation. In section 5.3 we discuss about the flexibility and options for customization that our software implementation provides to the users. Finally we conclude the chapter in section 5.4.

5.1 Input Format

Before going into the details of our software implementation let us first understand in which format data are given as input to our application. To keep the data input procedure as simple as possible, our application takes input in the form of Microsoft Excel Spreadsheets. The input data are provided in two separate spreadsheets: one containing the list of instructors, and the other containing the list of courses. These two spreadsheets have to be registered as an ODBC (Open database Connectivity) data source in the System DSN (Data Source Name), so that data can be read from them using "open database connectivity". Sample input format for instructors and courses are shown in tables 5.1 and 5.2.

Before execution of phase-1, we have another phase, which we call phase-0. The purpose of phase-0 is only to read data from the MS Excel spreadsheets.

Name	Type	Course Limit	Day Sequence	Time	1st Preference	2nd Preference	3rd Preference
Rogers, Michael	A	14	MWF	M	CPSC2620, STAT1770	NONE	NONE
Cruise, Tom	A	11	TR	M	MATH1560, STAT1770	NONE	NONE
Pit, Brad	A	14	TR	N	CPSC1620, CPSC2660	NONE	NONE
Gibson, Mel	A	1	TR	A	MATH0500	NONE	NONE
Williams, Robin	A	12	MWF	M	MATH1410, MATH2560	NONE	NONE
Khan, Shahrukh	A	14	MWF	M	MATH1560, MATH2865	NONE	NONE
Rai, Ashwaria	A	5	MWF	A	CPSC2690	NONE	NONE
Heiden, Mathew	A	9	MWF	N	CPSC1000	NONE	NONE
Ponting, Ricky	A	2	TR	N	MATH1560	NONE	NONE
Martin, Ricky	G	1	N	N	CPSC1000	NONE	NONE
Dravid, Rahul	G	1	N	N	CPSC1000	NONE	NONE
Khan, Imran	G	1	N	N	CPSC1000	NONE	NONE
Korbet, Jim	P	2	MWF	M	STAT1770	NONE	NONE
Homes, Sharlok	P	2	MWF	M	MATH2000, MATH3410	NONE	NONE

Table 5.1: Sample input for instructors' information

Course No.	Section	Title	Duration
CPSC1620	A	Fundamentals of Programming I	01
CPSC1620	B	Fundamentals of Programming I	01
CPSC1620	L1-AB	Lab for AB	01
CPSC1620	L2-AB	Lab for AB	01
CPSC1620	L3-AB	Lab for AB	01
CPSC1620	L4-AB	Lab for AB	01
CPSC1620	T1-AB	Tutorial for AB	01
CPSC1620	T2-AB	Tutorial for AB	01
CPSC1620	T3-AB	Tutorial for AB	01
CPSC2610	A	Introduction to Digital Systems	01
CPSC2620	A	Fundamentals of Programming II	01

Table 5.2: Sample input for courses' information

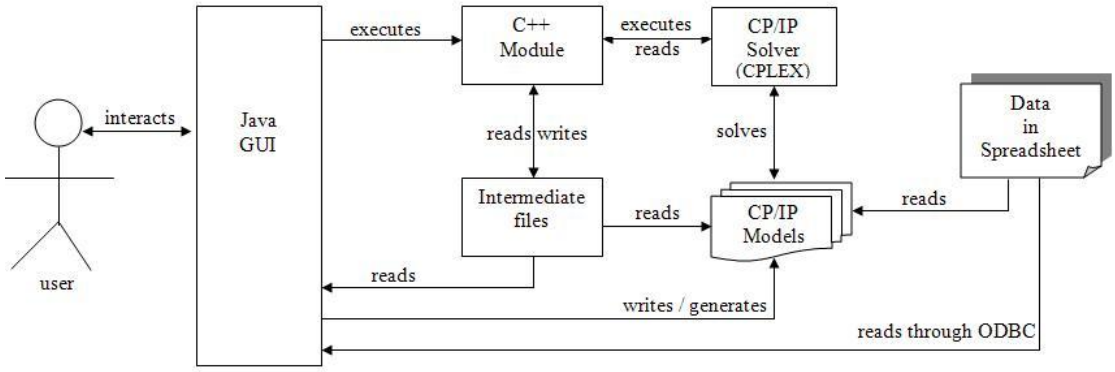

Figure 5.1: Architecture of the timetabling implementation

5.2 Software Architecture

Our software implementation of the multi-phase course timetabling problem is a desktop application in Windows XP platform. Its modular architecture is shown in in figure 5.1. At the heart of the implementation are the ILP and CP models of different phases (phase-1 through phase-4). These models are implemented in OPL (Optimization Programming Language). A commercial solver, ILOG's CPLEX (CPLEX 9.0 and Solver 6.0) is used for solving the OPL models. The solution obtained from one phase after necessary formatting feeds into the following phase(s). A C++ module is used to integrate all these phases and enable data propagation among phases. This C++ module is actually a stand-alone executable (.exe), which works like a controller. At the front end there is a graphical user interface (GUI), which is developed in Java 1.6 using "Swing" components. Users interact with this GUI and users' requests are forwarded to the C++ controller as necessary. Based on the request, the C++ controller invokes the CPLEX solver.

5.2.1 Back-end Computation

When the user requests through the GUI for feasible schedule from the given input data, the C++ module is executed. This C++ module invokes the CPLEX solver, which solves the CP or ILP models. The C++ executable carries out the intermediate computations and formatting of data as needed for the next phases to be executed. These intermediate results are sometimes written in the secondary memory as temporary files for convenience. Communication between the GUI and the C++ controller is done through a command prompt, which acts like a virtual message board for them. C++ controller writes on the command prompt, the GUI reads from it, and vice versa. However, this handshaking between the GUI and C++ module remains invisible to the end users.

After providing the input data, the user may customize his or her need by choosing constraints to be applied in the computation for the desired timetable. In such circumstances the Java GUI, before invoking the C++ module, generates the CP or IP models according to the customizations made. Then the C++ controller invokes the solver to solve those customized models.

After successful execution of the phases, the C++ controller accumulates the results of all the phases to determine the complete schedule. This schedule is then written in a file. The Java GUI then reads the schedule from the file and renders to the user in a presentable way.

5.2.2 Graphical User Interface (GUI)

The graphical user interface (GUI) of our timetabling implementation is shown in figure 5.2. This graphical user interface (GUI) serves two purposes: firstly, it allows users' interaction with the software implementation in a user-friendly manner. Using the WIMP

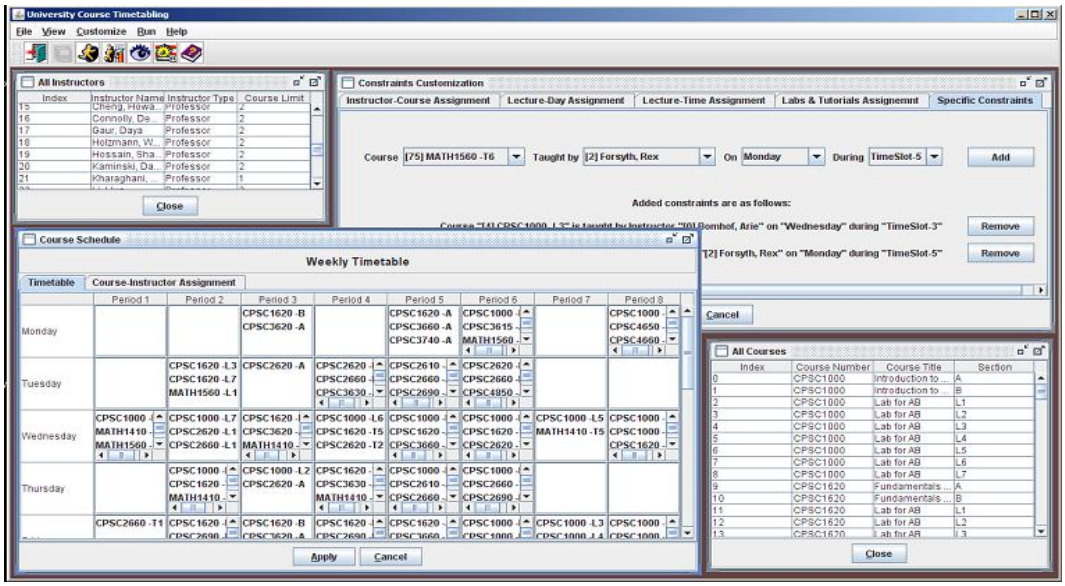

Figure 5.2: GUI of the timetabling implementation

(Window-Icon-Menu-Pointer) interface the users can easily interact simply through mouse-clicks without typing any command. Secondly, it displays the timetable in an organized way and allows the user to manipulate it. The GUI also provides the user a flexible interface for customizing constraints. Thus all the architectural complexities are hidden behind the GUI while keeping the flexibility to customize users' need. We will discuss more on this constraint customization facility later in this chapter.

The GUI wrapping makes the users comfortable in using our software implementation and guides them to interact with it. The integrated "User Guide" implemented using "JavaHelp" technology would be useful for anyone to learn how to use the software.

5.3 Flexibility and Customization

Our timetabling implementation provides the flexibility to customize a problem in three different ways:

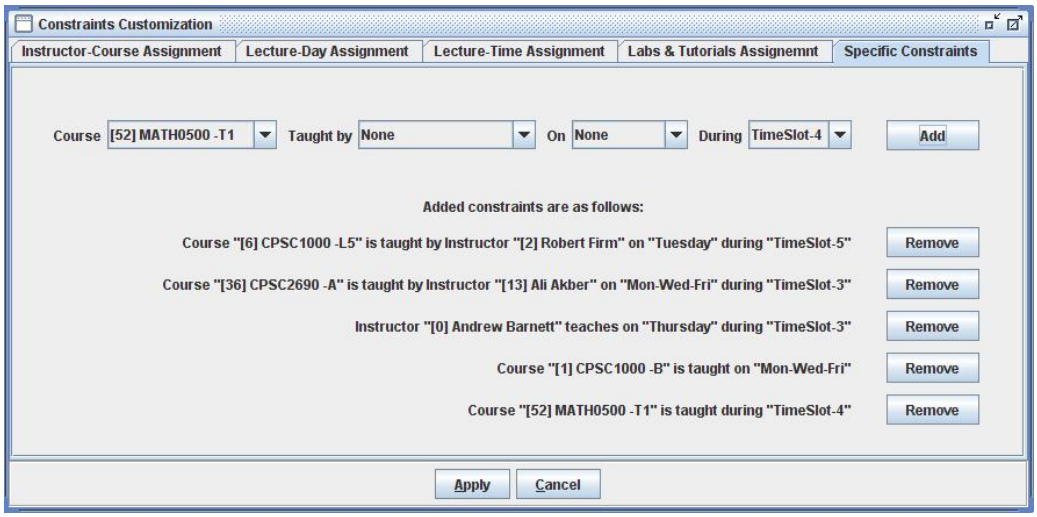

Figure 5.3: GUI for fixing instructor-course-day-time assignment

i) assignment of resources (instruction-units, instructors, days, and time-slot) prior to scheduling.

ii) choosing constraints from the constraint repository.

iii) modifying a previously generated solution and computing a new solution based on the modification.

After necessary customization on constraints, the user can choose to run the program using these customized settings. The system then computes and generates new solution based on these settings.

5.3.1 Assignment of Resources

Before generating the timetable, using the graphical user interface (as shown in figure 5.3) the user can do resource assignment of the following types

- assign instruction-unit to instructor

- assign instruction-unit to instructor and day or day-sequence (as appropriate)
- assign instruction-unit to instructor and day or day-sequence (as appropriate) as well as specific time-slot
- assign instruction-unit to day or day-sequence (as appropriate) and specific time-slot
- assign instruction-unit to day or day-sequence (as appropriate)
- assign instruction-unit to specific time-slot, so that the course is taught during that time-slot on whatever day(s) it is scheduled
- assign instructor to day or day-sequence (as appropriate) as well as specific time-slot so that all instruction-units taught by this instructor are taught on that specific time-slot of that specific day or day-sequence. Obviously, in such cases, the instructor cannot teach more than one instruction-unit
- assign instructor to day or day-sequence (as appropriate) so that all instruction-units taught by this instructor are taught on that specific day
- assign instructor to time-slot so that all instruction-units taught by that instructor are scheduled during that specific time-slot of any day(s)

However, care should be take while doing the resource assignments before computing the schedule. Conflicting assignments may make the underlying problem instance infeasible.

5.3.2 Choosing Constraints from Constraint-Repository

Our timetabling implementation provides the user a constraint repository. From this the user may choose constraints applicable for his or her requirement. To do this, the user interacts with the user interface shown in figure 5.4. This feature is quite useful for the users

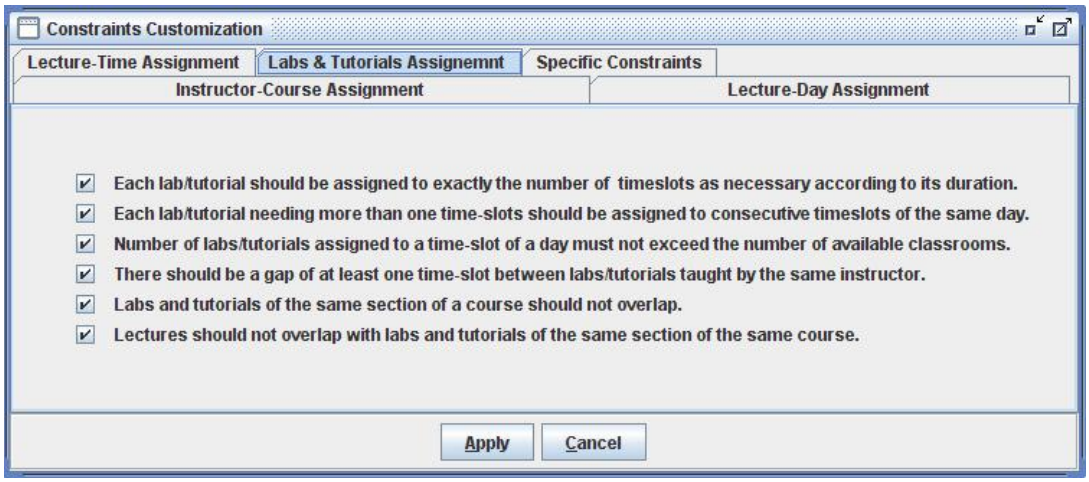

Figure 5.4: GUI for choosing constraints from the constraint-set

who have some knowledge about the detail of the timetabling requirements. Proper selection of constraints may reduce computation time, while improper selection of constraints may make the problem instance infeasible.

For the convenience of users who may not have much idea about constraints or timetabling requirements, a set of constraints are made preselected by default. The user may accept these constraints and get the schedule computed based on the default settings.

5.3.3 New Solution Based on Previous Solution

When a schedule is generated, the course-day-time assignment can be viewed in the graphical user interface shown in figure 5.5.

This interface also allows the user to remove instruction-units from any time-slot of any of the five weekdays.

Further, the assignment of instruction-units to instructors may be viewed in the user interface shown in figure 5.6. Using this interface the user may also unassign certain

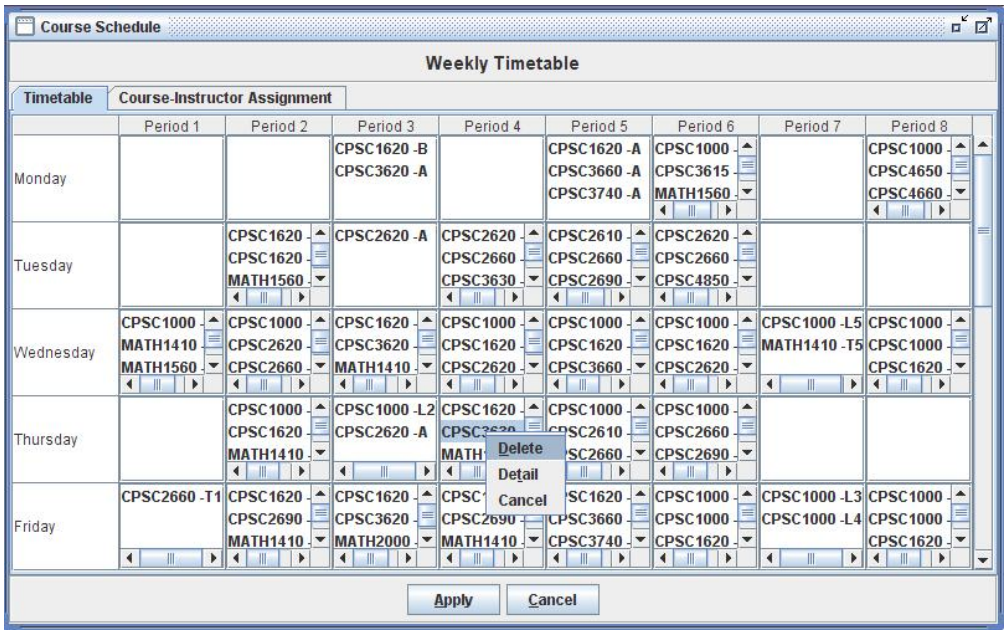

Figure 5.5: GUI for modifying course-day-time assignment of a computed solution

instruction-units from certain instructors.

Keeping the remaining assignment of instruction-units to instructors, days and time-slots, new solutions may be computed to schedule the unassigned instruction-units.

This feature may be very useful in many universities, where the timetable for one semester to another does not differ much. In such universities, the timetable for an earlier semester may be used to create the timetable for a new semester. Without much effort the old timetable may be loaded and then modified using the GUI's shown in figure 5.5 and 5.6. This feature should also be useful to accommodate the last minute changes needed. In such cases it does not compute starting from the scratch, and so the solution is generated quickly.

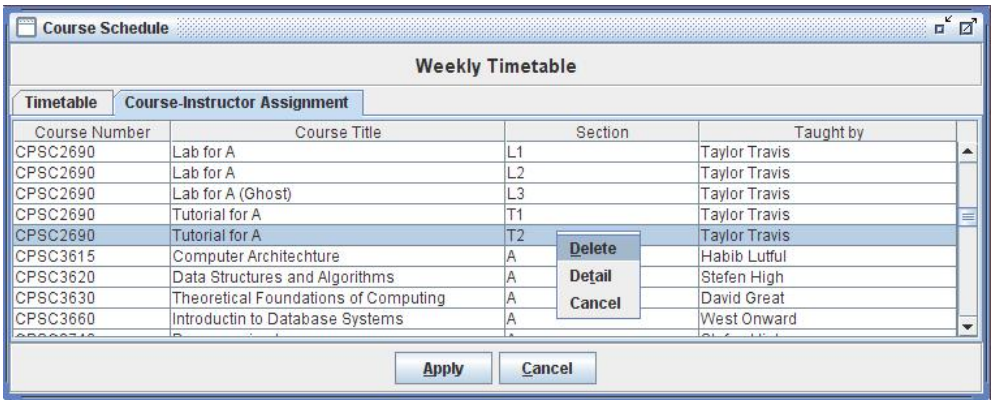

Figure 5.6: GUI for modifying instructor-course assignment of a computed solution

5.4 Summary

The software implementation of our timetabling implementation has a modular architecture. Different modules, such as the GUI, C++ controller, or the ILP and CP models may be modified easily without much affecting others. This makes our software implementation well manageable. ILOG's CPLEX solver may be replaced by any other solver without affecting the Java GUI at all. The computational and architectural complexities are hidden from the user by the GUI. This graphical user interface allows different types of customizations and fine tuning as needed by users with different expertise levels. This customization enables our implementation to handle the cross-department course assignment issues. For example, there may be some courses from different departments which must be scheduled in different times. Such courses may easily be handled by assigning them time-slots prior to computation for the entire schedule.

Chapter 6

Experiments and Evaluation

In this chapter we present the results of experimental evaluations of our timetabling implementation. As mentioned in chapter 5, we use ILOG's CPLEX (OPLStudio 3.7 with CPLEX 9.0 and Solver 6.0) for solving the ILP and CP models. All experiments presented in this chapter are run on 994 MHz AMD Athlon(tm) 64 bit processor 3500+ with 512 MB RAM in Windows XP environment. Section 6.1 presents the result of experiments with the real world timetabling problem instances at the University of Lethbridge. In section 6.2 we present the experimental results on generated problem instances. Finally, section 6.3 summarizes the chapter.

6.1 Experiment with Real World Data

We experiment with the course timetabling data from a number of academic departments at the University of Lethbridge. The name of the departments are suppressed to maintain confidence. For the first experiment we have

- 120 instruction units. Out of these 120 instruction-units 35 are lectures and rest 85 are labs or tutorials.
- Total 33 instructors consisting of 21 professors, 9 academic assistants and 3 graduate students.
- 20 classrooms.
- 8 time-slots in each day of MWF and 6 time-slots in each day of TR.
- 30 instructors mention their preferences on days or day-sequences.

	Phase-1		Phase-2	Phase-3		Phase-4
	1a	1b		TR	MWF	
time	0.043	0.047	0.016	0.02	0.02	0.95
Obj	137	322	105	44	53	193
maxObj	142	392	105	48	57	276

Table 6.1: Time and objective value of solution to the timetabling problem instance of department (a) at the University of Lethbridge.

- 23 instructors mention their preferences on time (morning or afternoon).
- All instructors mention the courses (set of courses with highest level of preference) that they want to teach.
- 3 instructors each mention a set of 3 courses as his or her preferred courses. 12 instructors each mention a set of 2 courses that he or she wants to teach. 18 instructors each mentioned only 1 preferred course.

Table 6.1 presents the result of our experiment with the timetabling problem. For phase-1a, phase-1b, and phase-2 we found the optimal solutions by applying ILP. However, we applied CP for phase-3 and phase-4, and the best solutions found in 20 seconds (CPU time) time limit are presented here. The row labeled as maxObj presents the upper bounds on the objective functions.

We further experiment with timetabling data of more academic departments applying ILP for all the phases. Results are presented in table 6.2, where each row presents the time in seconds (CPU time) and objective values of optimal solutions to different phases of a single problem instance. Given below are the terms and their meanings that we use to describe the experimental results.

C = total number of instruction-units.

Lec = number of lectures.

lab = number of labs.

T = number of tutorials.

I = total number of instructors.

P = number of professors.

A = number of academic assistants.

G = number of graduate students.

R = number of available classrooms.

For each problem instance (corresponding to each row of the table) the values in the second left most column describes the problem size and specification. The remaining columns except the left most two present the time needed for finding solutions to the subproblem of each phase. The objective values attained in each solution are also shown in parenthesis. The upper bounds on the objective values (as calculated following the procedures described in chapter 4) are shown between square brackets ([.]).

For explanation, the second experiment (with department (b)) is carried out on an instance having total 110 instruction-units among which there are 106 lectures, 4 labs, and 0 tutorial. Moreover, there are 54 professors, 3 academic assistants, and 0 graduate students resulting 57 instructors in total. Number of available classrooms is 20. The results in the table indicates that the optimal solution to the phase-1a subproblem is solved in 0.219 second with objective value 424, whereas the upper bound on the objective value is 488. Similarly, phase-3 from day-sequence TR is solved in 0.14 second with objective value 155, whereas the upper bound is 171.

Dept.	Experiments (C, Lec, Lab, T, I, P, A, G, R)	Phase-1 time (obj) [maxObj]		Phase-2 time (obj) [maxObj]	Phase-3 time (obj) [maxObj]		Phase-4 time (obj) [maxObj]
		1a	1b		TR	MWF	
(a)	120, 35, 15, 60, 33, 21, 9, 3, 20	0.953 (137) [142]	0.047 (322) [392]	0.015 (105) [105]	0.015 (44) [48]	0.015 (53) [57]	3.235 (241) [276]
(b)	110, 106, 4, 0, 57, 54, 3, 0, 20	0.219 (424) [488]	0.016 (16) [20]	0.015 (318) [321]	0.14 (155) [171]	0.047 (128) [147]	0.203 (15) [15]
(c)	41, 18, 23, 0, 20, 14, 6, 0, 20	0.343 (69) [76]	0.00 (86) [107]	0.00 (54) [57]	0.016 (27) [27]	0.00 (27) [27]	2.875 (135) [135]
(d)	40, 14, 0, 26, 14, 10, 4, 0, 20	0.078 (56) [61]	0.016 (101) [113]	0.015 (40) [43]	0.016 (22) [24]	0.016 (16) [18]	0.563 (64) [78]

Table 6.2: Time and objective values of *optimal* solutions to timetabling problem instances of different academic departments at the University of Lethbridge

6.2 Evaluation with Generated Data

We continue experimenting with our generated data. We pseudo-randomly generate feasible problem instances of various sizes. While generating problem instances we enforce feasibility taking into account the following

- $$|Lectures| \leq \sum_{i \in Prof} Limit_i$$

  which means that the number of lectures should not exceed the total teaching capacity of the professors.

- $$|Labs \cup Tutorials| \leq \sum_{i \in AcadAsst \cup GradStud} Limit_i$$

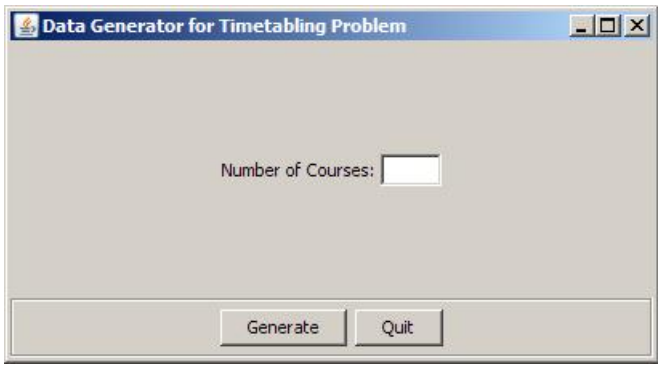

Figure 6.1: User interface of the problem instance generator

which means that the number of labs and tutorials should not exceed the total teaching capacity of the academic assistants and graduate students.

- $$nRoom \geq \lceil (\sum_{c \in C} duration_c) / \sum_{d \in Days} TimeSlots_d) \rceil$$

  which computes minimum number of class-rooms required if there were no constraints to be imposed.

We have developed a problem instance generator written in Java (see figure 6.1), which takes the number of courses as input and generates a feasible problem instance taking care of the above mentioned issues.

We experiment with problem instances of various sizes and apply ILP to solve each phase. Table 6.3 shows the experimental results.

We further experiment with these problem instances applying CP (instead of ILP) for phase-3 and phase-4. The results are presented in Table 6.4.

Here, applying ILP we find the optimal solutions to phase-1a, phase-1b, and phase-2 for all the problem instances. But we found that for phase-3 and phase-4, CP techniques take quite long time to find the optimal solutions. Therefore we allocate fixed time for the

Serial No.	Experiments (C, Lec, Lab, T, I, P, A, G, R)	Phase-1 time (obj) [maxObj]		Phase-2 time (obj) [maxObj]	Phase-3 time (obj) [maxObj]		Phase-4 time (obj) [maxObj]
		1a	1b		TR	MWF	
1	279, 56, 105, 118, 104, 45, 42, 17, 20	0.156 (218) [276]	0.453 (856) [1087]	0.016 (150) [150]	0.031 (96) [108]	0.016 (54) [60]	32.203 (799) [888]
2	340, 66, 136, 138, 147, 51, 65, 31, 20	0.109 (257) [325]	0.86 (1038) [1324]	0.016 (174) [174]	0.047 (112) [123]	0.031 (64) [75]	56.735 (941) [1065]
3	392, 83, 157, 152, 162, 61, 64, 37, 20	0.172 (317) [402]	1.047 (1147) [1531]	0.016 (228) [228]	0.047 (136) [153]	0.047 (88) [96]	310.515 (1117) [1236]
4	444, 89, 192, 163, 159, 64, 67, 28, 20	0.187 (344) [434]	1.063 (1333) [1744]	0.016 (242) [242]	0.047 (162) [180]	0.031 (73) [87]	311.891 (1250) [1386]
5	504, 104, 201, 199, 202, 80, 84, 38, 20	0.844 (391) [496]	1.719 (1489) [1935]	0.016 (286) [286]	0.047 (174) [195]	0.031 (106) [117]	361.047 (1382) [1536]
6	543, 108, 235, 200, 213, 77, 103, 33, 50	0.265 (422) [525]	2.266 (1678) [2127]	0.016 (289) [289]	0.062 (175) [201]	0.047 (116) [123]	520.32 (1547) [1695]
7	647, 126, 250, 271, 277, 99, 126, 52, 50	0.437 (490) [623]	3.687 (1988) [2565]	0.016 (336) [339]	0.094 (235) [264]	0.047 (103) [114]	667.344 (1769) [1962]
8	661, 135, 284, 242, 261, 101, 116, 44, 50	0.89 (498) [636]	3.453 (1975) [2585]	0.032 (366) [366]	0.125 (222) [249]	0.047 (134) [156]	805.391 (1826) [2019]
9	675, 130, 282, 263, 269, 99, 126, 44, 50	0.968 (500) [633]	3.953 (2050) [2654]	0.016 (348) [348]	0.188 (212) [246]	0.031 (134) [144]	1112.438 (1903) [2094]
10	714, 140, 308, 266, 272, 109, 125, 38, 50	0.547 (536) [689]	3.922 (2183) [2813]	0.032 (375) [378]	0.041 (185) [219]	0.078 (179) [201]	1066.422 (1939) [2205]
11	765, 163, 304, 298, 335, 129, 139, 67, 50	1.313 (628) [784]	5.50 (2333) [2957]	0.016 (434) [434]	0.078 (274) [303]	0.062 (167) [186]	1113.391 (2130) [2343]

Table 6.3: Time and objective values of *optimal* solutions to different problem instances

Serial No.	Experiments (C, Lec, Lab, T, I, P, A, G, R)	Phase-1 time (obj) [maxObj]		Phase-2 time (obj) [maxObj]	Phase-3 time (obj) [maxObj]		Phase-4 time (obj) [maxObj]
		1a	1b		TR	MWF	
1	279, 56, 105, 118, 104, 45, 42, 17, 20	0.156 (218) [276]	0.453 (856) [1087]	0.016 (150) [150]	0.02 (96) [108]	0.03 (54) [60]	2.34 (621) [888]
2	340, 66, 136, 138, 147, 51, 65, 31, 20	0.109 (257) [325]	0.86 (1038) [1324]	0.016 (174) [174]	0.02 (112) [123]	0.00 (64) [75]	2.42 (763) [1065]
3	392, 83, 157, 152, 162, 61, 64, 37, 20	0.172 (317) [402]	1.047 (1147) [1531]	0.016 (228) [228]	0.03 (136) [153]	0.02 (88) [96]	3.09 (861) [1236]
4	444, 89, 192, 163, 159, 64, 67, 28, 20	0.187 (344) [434]	1.063 (1333) [1744]	0.016 (242) [242]	0.02 (162) [180]	0.02 (73) [87]	3.50 (956) [1386]
5	504, 104, 201, 199, 202, 80, 84, 38, 20	0.844 (391) [496]	1.719 (1489) [1935]	0.016 (286) [286]	0.05 (174) [195]	0.02 (106) [117]	3.63 (1064) [1536]
6	543, 108, 235, 200, 213, 77, 103, 33, 50	0.265 (422) [525]	2.266 (1678) [2127]	0.016 (289) [289]	0.03 (175) [201]	0.02 (116) [123]	4.16 (1185) 5.08 (1187) [1695]
7	647, 126, 250, 271, 277, 99, 126, 52, 50	0.437 (490) [623]	3.687 (1988) [2565]	0.016 (336) [339]	0.09 (235) [264]	0.03 (103) [114]	4.95 (1361) [1962]
8	661, 135, 284, 242, 261, 101, 116, 44, 50	0.89 (498) [636]	3.453 (1975) [2585]	0.032 (366) [366]	0.05 (222) [249]	0.06 (134) [156]	11.00 (1332) 11.08 (1334) 14.50 (1336) [2019]
9	675, 130, 282, 263, 269, 99, 126, 44, 50	0.968 (500) [633]	3.953 (2050) [2654]	0.016 (348) [348]	0.11 (212) [246]	0.03 (134) [144]	5.23 (1457) [2094]
10	714, 140, 308, 266, 272, 109, 125, 38, 50	0.547 (536) [689]	3.922 (2183) [2813]	0.032 (375) [378]	0.03 (185) [219]	0.05 (179) [201]	19.25 (1507) [2205]
11	765, 163, 304, 298, 335, 129, 139, 67, 50	1.313 (628) [784]	5.50 (2333) [2957]	0.016 (434) [434]	0.14 (274) [303]	0.03 (167) [186]	19.19 (1640) [2343]

Table 6.4: Time and objective values of solutions to different problem instances

CP implementations of phase-3 and phase-4 to find feasible solutions. Computation for phase-3 and phase-4 continued for 20 seconds (CPU time) in experiments 1 through 5, and 50 seconds (CPU time) in experiments 5 through 11. All the solutions found within these time limits are presented in table 6.4.

For explanation, the instance of experiment 8 has in total 661 instruction-units, which include 135 lectures, 284 labs, and 242 tutorials. Besides, it has 261 instructors, among which 101 are professors, 116 are academic assistants, and the rest 44 are graduate students. There are 50 classrooms available.

The optimal solution to the phase-1a problem is found in 0.89 seconds (CPU time) with objective value 498, where the upper bound on the objective value is 636. Similarly, the optimal solutions to the phase-1b and phase-2 problems are found in 3.453 and 0.032 seconds (CPU time) with objective values 1975 and 366 respectively. The upper bound on the objective values of phase-1b and phase-2 are 2585 and 366 respectively. Here it is remarkable that for this problem instance the objective value achieved is equal to its upper bound. For the TR day-sequence of phase-3, within the 50 seconds time limit, a feasible solution found in 0.05 seconds with objective value 222, whereas the upper bound on the objective value is 249. Likewise, a solution with objective value 134 (upper bound 134) is found in 0.06 seconds for the MWF day-sequence of phase-3. Within the 50 seconds time limit, 3 solutions are found for the phase-4 problem. The first solution is found in 11.00 seconds with objective value 1332, the second one with objective value 1334 is found in 11.08 seconds, and in 14.50 seconds the third solution with objective value 1336 is found. For this instance the upper bound on the phase-4 objective value is 2019.

Figure 6.2, figure 6.3, and figure 6.4 present the performance (time required for solving problems of different sizes) of phase-1a, phase-1b, and phase-2 respectively. Figure 6.5 shows the time needed by phase-3 for solving different problem instances for scheduling instruction-units in day-sequence TR (Tuesday-Thursday). Figure 6.6 displays the same for

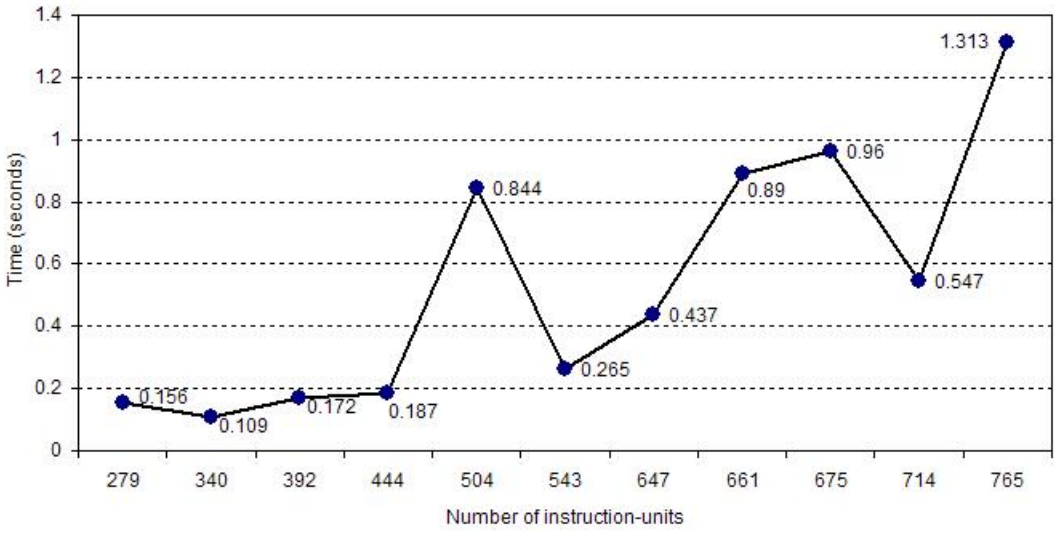

Figure 6.2: Performance of phase-1a

instruction-units in day-sequence MWF (Monday-Wednesday-Friday). Figure 6.7 presents the time needed by phase-4 for solving problems of different sizes.

6.2.1 Combined versus Decomposed Phase-1

We solve phase-1 problem by decomposing it into two phases: phase-1a and phase-1b. We compare the performance of this decomposed phase-1 with that of the combined phase-1, where phase-1 is not decomposed.

The table 6.2.1 presents the comparative time and objective values for problem instances of different sizes. As we see, for each problem instance the objective value of the combined phase-1 is equal to the summation of the objective values of the split phase-1a and and phase-1b. However, in case of the time required to find the optimal solution, of combined phase-1 always took more time than the summation of time required by phase-1a and phase-1b. This is depicted in figure 6.8. As expected, it is evident here that phase-1

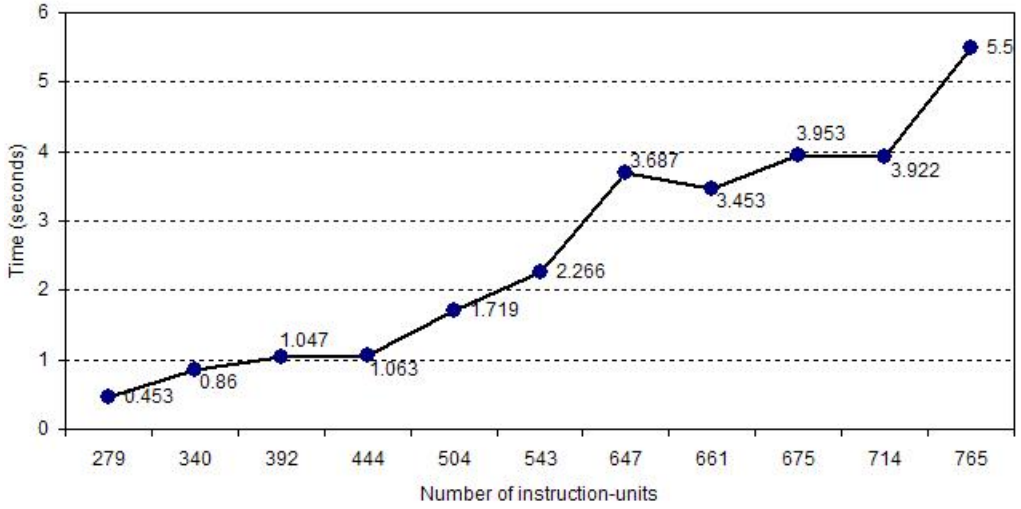

Figure 6.3: Performance of phase-1b

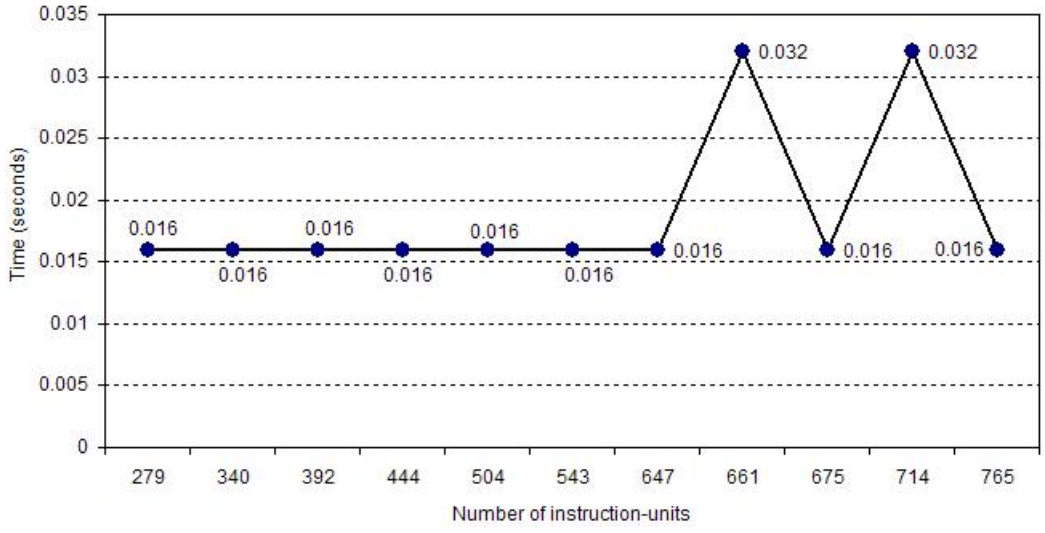

Figure 6.4: Performance of phase-2

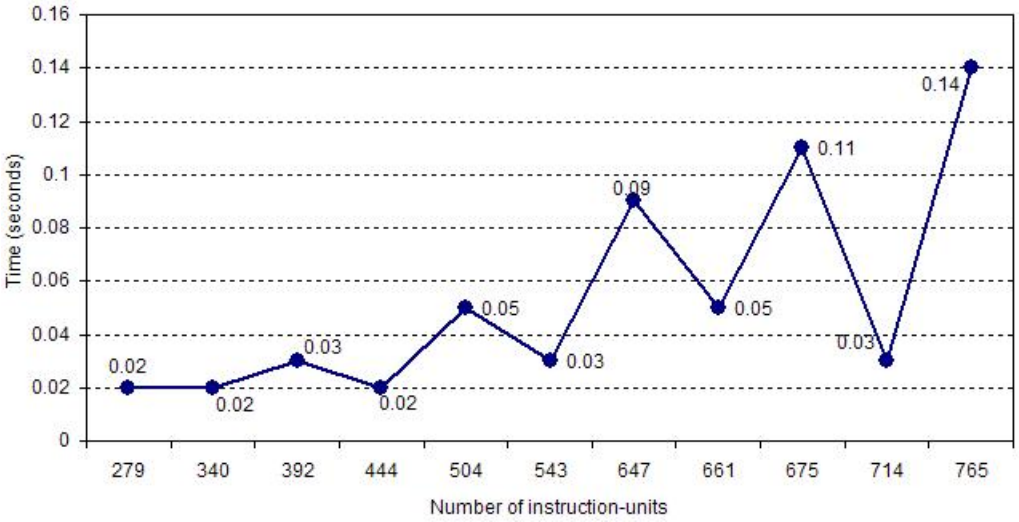

Figure 6.5: Performance of phase-3 on TR

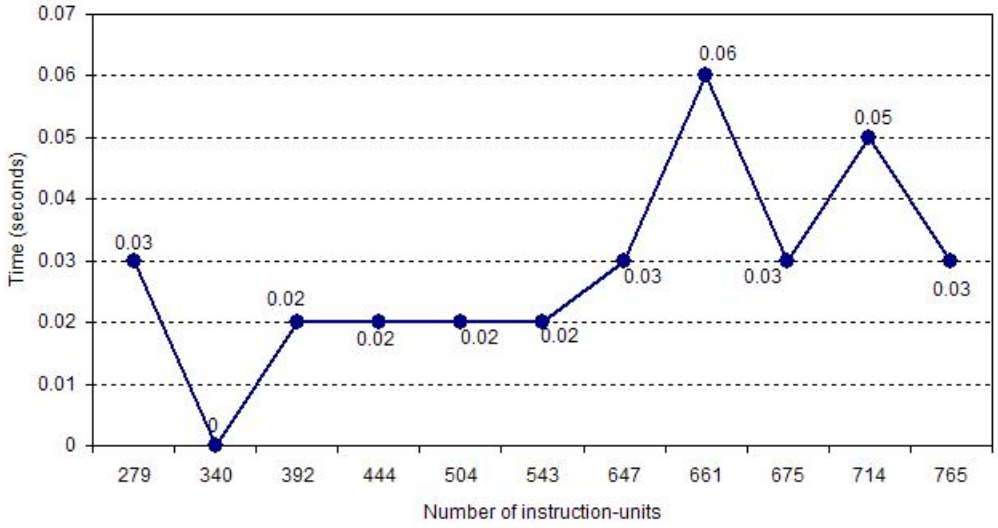

Figure 6.6: Performance of phase-3 on MWF

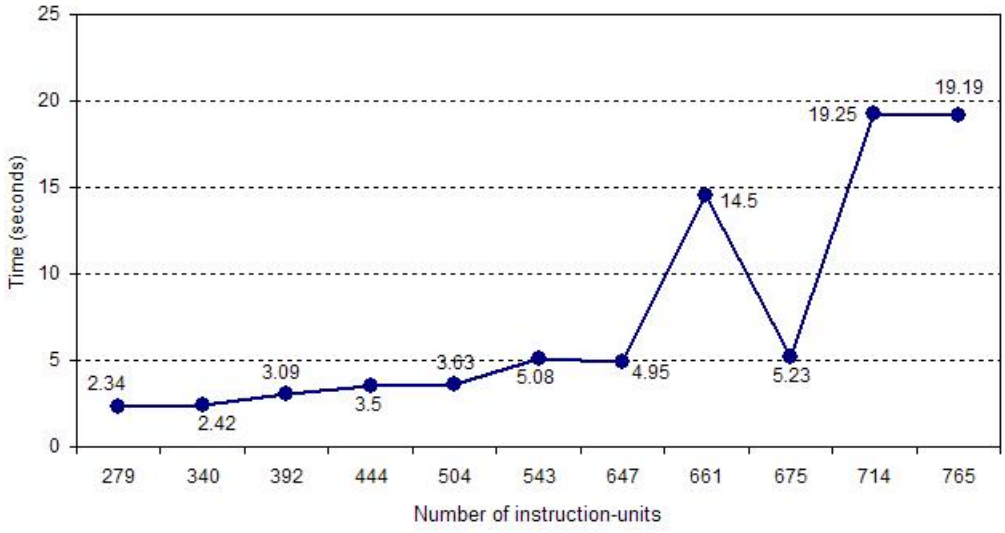

Figure 6.7: Performance of phase-4

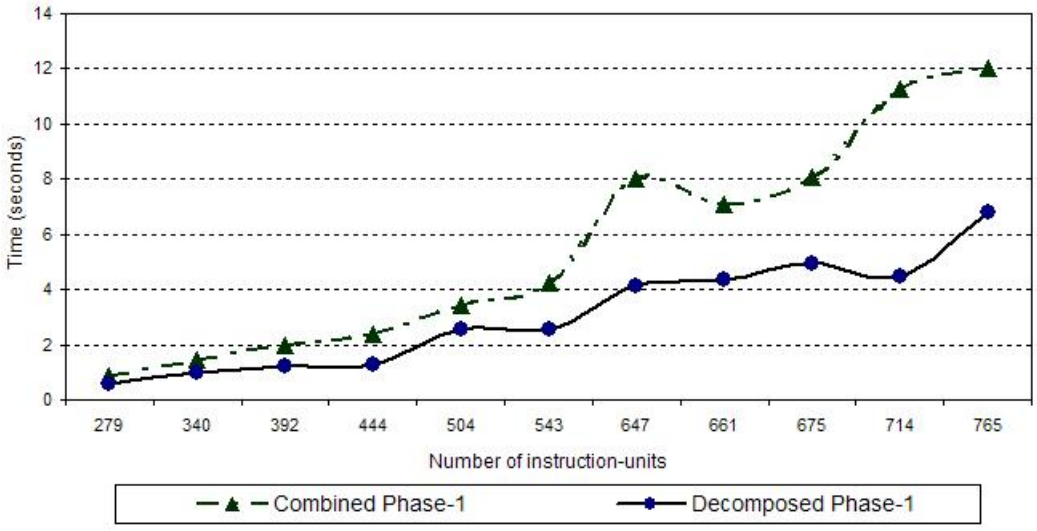

Figure 6.8: Comparison between combined and decomposed phase-1

Serial	Experiments (C, Lec, Lab, T, I, P, A, G, R)	Combined phase-1	Decomposed phase-1		
			1a	1b	1a + 1b
1	279, 56, 105, 118, 104, 45, 42, 17, 20	0.8251 (1074)	0.156 (218)	0.453 (856)	0.609 (1074)
2	340, 66, 136, 138, 147, 51, 65, 31, 20	1.4642 (1295)	0.109 (257)	0.86 (1038)	0.969 (1295)
3	392, 83, 157, 152, 162, 61, 64, 37, 20	1.9796 (1464)	0.172 (317)	1.047 (1147)	1.219 (1464)
4	444, 89, 192, 163, 159, 64, 67, 28, 20	2.4048 (1677)	0.187 (344)	1.063 (1333)	1.25 (1677)
5	504, 104, 201, 199, 202, 80, 84, 38, 20	3.4031 (1880)	0.844 (391)	1.719 (1489)	2.563 (1880)
6	543, 108, 235, 200, 213, 77, 103, 33, 50	4.2387 (2100)	0.265 (422)	2.266 (1678)	2.531 (2100)
7	647, 126, 250, 271, 277, 99, 126, 52, 50	8.0188 (2478)	0.437 (490)	3.687 (1988)	4.124 (2478)
8	661, 135, 284, 242, 261, 101, 116, 44, 50	7.063 (2473)	0.89 (498)	3.453 (1975)	4.343 (2473)
9	675, 130, 282, 263, 269, 99, 126, 44, 50	8.047 (2550)	0.968 (500)	3.953 (2050)	4.921 (2550)
10	714, 140, 308, 266, 272, 109, 125, 38, 50	11.281 (2719)	0.547 (536)	3.922 (2183)	4.469 (2719)
11	765, 163, 304, 298, 335, 129, 139, 67, 50	12.031 (2961)	1.313 (628)	5.50 (2333)	6.813 (2961)

Table 6.5: Comparison between combined and decomposed phase-1

problem is solved quicker when it is solved using two separate phases.

6.2.2 CP versus ILP Implementation

As mentioned before, we compared the performance of CP and IP applying them for solving phase-3 and phase-4. Table 6.6 presents the comparison between the performance of CP and ILP implementations.

We further experiment applying CP and ILP separately to phase-4 for comparing the their performance more accurately. (In this case ILP is applied to the earlier phases from phase-1 through phase-3.) For each problem we first apply ILP to find the optimal solution. Then we apply CP putting a time limit equal to the time ILP takes for finding the optimal solution. Comparative results are presented in table 6.7.

Each row in the table compares the performance of phase-4 on the same problem instance. The third column shows the time that ILP needs to find the optimal solution. The fourth column records the objective value of the optimal solution. The rightmost column shows the objective value of the best feasible solution found by applying CP with the given time limit. For example, in experiment 11, ILP implementation finds the optimal solution in 1113.391 seconds with objective value 2130, whereas in 1113.391 seconds the best feasible solution found by CP implementation has objective value 1679.

As we see from the experimental results, CP takes less time in finding *feasible* solutions. But experiments show that though CP finds feasible solutions quickly it takes considerably long time to verify optimality. On the other hand, in our experiments IP finds the optimal solution in comparatively less time, but it does not give any early non-optimal feasible solution. Therefore, for large problems, when a feasible but not necessarily optimal solution is sufficient, CP may be a wise choice to be applied.

Serial No.	Experiments (C, Lec, Lab, T, I, P, A, G, R)	CP Implementation			ILP Implementation		
		Phase-3 Time (obj)		Phase-4 Time (obj)	Phase-3 Time (obj)		Phase-4 Time (obj)
		TR	MWF		TR	MWF	
1	279, 56, 105, 118, 104, 45, 42, 17, 20	0.02 (96)	0.03 (54)	2.34 (621)	0.031 (96)	0.016 (54)	32.203 (799)
2	340, 66, 136, 138, 147, 51, 65, 31, 20	0.02 (112)	0.00 (64)	2.42 (763)	0.047 (112)	0.031 (64)	56.735 (941)
3	392, 83, 157, 152, 162, 61, 64, 37, 20	0.03 (136)	0.02 (88)	3.09 (861)	0.047 (136)	0.047 (88)	310.515 (1117)
4	444, 89, 192, 163, 159, 64, 67, 28, 20	0.02 (162)	0.02 (73)	3.50 (956)	0.047 (162)	0.031 (73)	311.891 (1250)
5	504, 104, 201, 199, 202, 80, 84, 38, 20	0.05 (174)	0.02 (106)	3.63 (1064)	0.047 (174)	0.031 (106)	361.047 (1382)
6	543, 108, 235, 200, 213, 77, 103, 33, 50	0.03 (175)	0.02 (116)	5.08 (1187)	0.062 (175)	0.047 (116)	520.32 (1547)
7	647, 126, 250, 271, 277, 99, 126, 52, 50	0.09 (235)	0.03 (103)	4.95 (1361)	0.094 (235)	0.047 (103)	667.344 (1769)
8	661, 135, 284, 242, 261, 101, 116, 44, 50	0.05 (222)	0.06 (134)	14.50 (1336)	0.125 (222)	0.047 (134)	805.391 (1826)
9	675, 130, 282, 263, 269, 99, 126, 44, 50	0.11 (212)	0.03 (134)	5.23 (1457)	0.188 (212)	0.031 (134)	1112.438 (1903)
10	714, 140, 308, 266, 272, 109, 125, 38, 50	0.03 (185)	0.05 (179)	19.25 (1507)	0.141 (185)	0.078 (179)	1066.422 (1939)
11	765, 163, 304, 298, 335, 129, 139, 67, 50	0.14 (274)	0.03 (167)	19.19 (1640)	0.078 (274)	0.062 (167)	1113.391 (2130)

Table 6.6: Performance comparison between CP and ILP implementations

Serial	Experiments (C, Lec, Lab, T, I, P, A, G, R)	Time (second)	ILP (optimal solution)	CP (feasible solution)
1	279, 56, 105, 118, 104, 45, 42, 17, 20	32.203	799	623
2	340, 66, 136, 138, 147, 51, 65, 31, 20	56.735	941	727
3	392, 83, 157, 152, 162, 61, 64, 37, 20	310.515	1117	853
4	444, 89, 192, 163, 159, 64, 67, 28, 20	311.891	1250	978
5	504, 104, 201, 199, 202, 80, 84, 38, 20	361.047	1382	1036
6	543, 108, 235, 200, 213, 77, 103, 33, 50	520.32	1547	1133
7	647, 126, 250, 271, 277, 99, 126, 52, 50	667.344	1769	1371
8	661, 135, 284, 242, 261, 101, 116, 44, 50	805.391	1826	1376
9	675, 130, 282, 263, 269, 99, 126, 44, 50	1112.438	1903	1467
10	714, 140, 308, 266, 272, 109, 125, 38, 50	1066.422	1939	1459
11	765, 163, 304, 298, 335, 129, 139, 67, 50	1113.391	2130	1579

Table 6.7: Performance comparison between CP and ILP implementations of phase-4

6.2.3 Phase-4 Heuristic

As mentioned in chapter 4, phase-4 assigns labs and tutorials to days and time-slots. The heuristic we use for searching the solution space tries to find better solutions earlier. Before deciding on a suitable variable ordering, we experiment with different ordering and examine the solution time and objective values.

The indices of the three dimensional decision variable (x_{cdt}) used in phase-4 model has the following meanings.

d stands for days. Days are ordered based on the number of available time-slots in a single day.

t refers to time-slots. Time-slots within a day are ordered according to the number of classrooms available during that period.

c represents instruction-units (labs and tutorials), which are ordered on the basis of their duration.

We use the following terms to describe our experimental results.

Resource: A resource is a single day, time-slot or instruction-unit.

Resource set: A resource set is a set of similar resources. Phase-4 deals with the 3 disjoint resource sets, one for the instruction-units, one for the days, and another for the time-slots.

Resource set selection order: Resource set selection order refers to the ordering of the three sets. One element of each sets are picked in the sequence as the resource sets are ordered.

Intra-set resource order: Intra-set ordering refers to the ordering of elements within a resource set.

We use ↑ and ↓ to indicate increasing and decreasing order respectively. Absence of any of these signs indicates the following ordering that we use as default.

For days: *increasing* number of available time-slots in a single days.

For time-slots: *increasing* number of classrooms during the period. within a day.

For instruction-units: *decreasing* duration.

We experiment using the timetabling problem instances of the University of Lethbridge (see section 6.1).

Table 6.8 presents results of different ordering of resource sets and intra-set resource ordering. Here, we don't differentiate between labs and tutorials. That means at the time of selecting an index corresponding to an instruction-unit, it does not really matter whether the index corresponds to a lab or tutorial. Each row of table 6.8 represents the result of experiment with a distinct resource set order and intra-set resource order. For each experiment, we put a 50 seconds time-limit on the search procedure to find solution.

For instance, if we look at the results on the 3rd row, we see that the first solution is obtained in 1.00 second with objective value 199. The rightmost column indicates that no other solution is found within the given 50 seconds time limit. The leftmost column shows the resource set selection order. For this experiment, first an instruction-unit is picked up, then a day is picked up, and at last a time-slot within the selected day is chosen. The second column from the left displays the intra-set ordering of resources, which is default for all variables in this experiment we are discussing now. That means, for selecting an instruction-unit, the one with the highest duration is picked; for selecting a day, the one with the minimum number of time-slots is chosen; and while picking a time-slot among all the time-slots within the selected day, the time-slot when the minimum number of rooms available, is selected.

Again, in the experiment where the first solution with objective value 201 is obtained in 2.19 seconds, the second column from the left indicates that only the time-slots are ordered here according to increasing number of classrooms available during that periods.

From the results of table 6.8 we come to a conclusion that *the best resource set selection order is* $\{c, d, t\}$, i.e., instruction-units, days, time-slots.

Keeping this resource set selection order {c, d, t}, we do more experiments to determine the best intra-set resource ordering for all the three sets of resources and come up with results shown in table 6.9. In this experiment we schedule first the labs and then the tutorials. The second column from the left displays the intra-set resource ordering while assigning the labs. The third column from the left shows the intra-set ordering for the tutorials. In the first row, the objective value is the highest (215), as well as the time required (0.97 seconds) is the minimum. This represents the best ordering of intra-set resource ordering. As we see, for both labs and tutorials, the resources in all the three resource sets are sorted in decreasing order. Hence, we conclude that the best intra-set resource order is $\{d\downarrow, c\downarrow, t\downarrow\}$.

Again, keeping the resource set selection order {c, d, t}, we experiment more by first scheduling tutorials and then labs. Table 6.10 shows the experimental results. Here the second row contains the best result, where the first solution with objective value 205 is obtained in 0.91 seconds. However, this is worse than the best result (objective value 215 in 0.97 seconds) we achieve in our previous evaluation.

Keeping the resource set selection order {c, d, t}, we further experiment without differentiating labs and tutorials. Table 6.11 shows the results we get. Here the objective value and time needed to find the first solution is still worse than the best result (objective 215 in 0.97 seconds) achieved so far.

All these experiments drive us to develop the heuristic (see chapter 4) for solving the phase-4 problem.

Resource set order	Intra-set resource order	1st solution		Next or optimal
		obj	time	
d, c, t	all	none	N/A	N/A
d, t, c	all	none	N/A	N/A
c, d, t	all	**199**	**1.00**	none
d, c, t	none	none	N/A	N/A
d, t, c	none	none	N/A	N/A
c, d, t	none	195	3.06	none
c, t, d	none	173	3.50	none
t, d, c	none	none	N/A	N/A
t, c, d	none	none	N/A	N/A
d, c, t	c	none	N/A	N/A
d, t, c	c	none	N/A	N/A
c, d, t	c	195	2.41	none
c, t, d	c	191	2.49	none
t, d, c	c	none	N/A	N/A
t, c, d	c	none	N/A	N/A
d, c, t	t	none	N/A	N/A
d, t, c	t	none	N/A	N/A
c, d, t	t	**201**	**2.19**	none
d, c, t	d	none	N/A	N/A
d, t, c	d	none	N/A	N/A
c, d, t	d	195	2.24	none
c, d, t	d	191	2.28	none
t, d, c	d	none	N/A	N/A
t, c, d	d	none	N/A	N/A
d, c, t	c, d	none	N/A	N/A
d, t, c	c, d	none	N/A	N/A
c, d, t	c, d	189	3.95	none
c, d, t	c, d	173	3.28	none
t, d, c	c, d	none	N/A	N/A
t, c, d	c, d	none	N/A	N/A
d, c, t	c, t	none	N/A	N/A
d, t, c	c, t	none	N/A	N/A
c, d, t	c, t	199	1.10	none
d, c, t	d, t	none	N/A	N/A
d, t, c	d, t	none	N/A	N/A
c, d, t	d, t	199	2.88	none

Table 6.8: Resource set ordering without differentiating labs and tutorials

Resource set order	Intra-set resource order		1st solution		Next or optimal
	labs	tutorials	Obj	time	
c, d, t	$d\downarrow, c\downarrow, t\downarrow$	$d\downarrow, c\downarrow, t\downarrow$	**215**	**0.97**	none
c, d, t	$d\downarrow, c\uparrow, t\uparrow$	$d\downarrow, c\downarrow, t\downarrow$	207	0.97	none
c, d, t	$d\downarrow, c\uparrow, t\uparrow$	$d\downarrow, c\downarrow, t\uparrow$	183	1.00	185 (75.25 sec)
c, d, t	$d\uparrow, c\downarrow, t\downarrow$	$d\downarrow, c\downarrow, t\downarrow$	215	1.02	none
c, d, t	$d\uparrow, c\uparrow, t\downarrow$	$d\downarrow, c\downarrow, t\downarrow$	213	0.97	none
c, d, t	$d\uparrow, c\uparrow, t\uparrow$	$d\downarrow, c\downarrow, t\downarrow$	207	1.03	none

Table 6.9: Intra-set resource ordering: first assigning labs and then tutorials

Resource set order	Intra-set resource order		1st solution		Next or optimal
	labs	tutorials	Obj	time	
c, d, t	$c\downarrow, d\uparrow, t\uparrow$	$c\downarrow, d\downarrow, t\downarrow$	201	1.02	none
c, d, t	$c\downarrow, d\downarrow, t\downarrow$	$c\downarrow, d\downarrow, t\downarrow$	**205**	**0.91**	none
c, d, t	$c\downarrow, d\uparrow, t\downarrow$	$c\downarrow, d\downarrow, t\downarrow$	205	0.99	none

Table 6.10: Intra-set resource ordering: first assigning tutorials and then labs

Resource set order	Intra-set resource order	1st Solution		Next or optimal
		obj	time	
c, d, t	$c\downarrow, d\uparrow, t\uparrow$	201	0.89	none
c, d, t	$c\downarrow, d\downarrow, t\downarrow$	**213**	**0.98**	none

Table 6.11: Intra-set resource ordering: assigning labs and tutorials together

6.3 Summary

We experiment with our multi-phase implementation using the real world timetabling problem instance at the University of Lethbridge. Further experimentations are carried out using generated problem instances. The comparison between decomposed and combined phase-1 implementation presented in section 6.2.1 shows that decomposition of the phase-1 problem finds the optimal solution taking less time than combined phase-1. Further, phase-1 and phase-2 gives the optimal solutions quickly for all the problem instances experimented with. As the ILP implementations of phase-3 and phase-4 take comparatively longer time to find the optimal solution, we apply CP to find a feasible solution within a given time bound. Besides, we apply heuristics to Phase-3 and phase-4 so that better (near to the optimal) solutions are found early.

Chapter 7

Conclusion and Future Directions

7.1 Concluding Remarks

Until recently many researchers have worked on university level course timetabling. Most of the works are on course scheduling based on students' demand. Courses are offered prior to start of an academic semester. Students then choose their desired courses. Typically, students are given a deadline for registering and dropping courses. After this deadline, if it is found that the number of students registered for a certain course is below a given threshold, the course may be dropped. Clearly, such a timetable is much affected by the students' demand. Unfortunately sufficient importance is not given on the availability and preferences of the faculty members. Moreover, dropping a course may cause last minute changes in the schedule. This may affect schedule of the faculty members. Moreover, such changes may also propagate to the schedule of the other courses offered in the semester. Therefore, scheduling of faculty members also demands much importance. The multi-phase approach presented in this thesis takes into account this issue. At the beginning of the timetabling procedure the courses to be offered may be determined based on the program requirement and expertise of the available faculty members. Students' demand may also be considered looking at the schedule of earlier semesters and finding a trend on the students' selection of courses. Thus we may reduce the probability of last minute changes in the schedule.

Splitting the entire problem into multiple subproblems, and solving each of them in separate phases yields a number of advantages. Usually the size of course timetabling problem instances at university level is quite large in terms of concerned resources and participants as well as number and types of constraints involved. Solving the entire problem as a whole

may be difficult. Exploiting the problem structure, our multi-phase approach functionally decomposes the entire problem into smaller subproblems with reduced complexity, which are easier to solve. We formulate mathematical models for all the subproblems, which are then translated into ILP and CP models. These ILP and CP models are completely independent of the other parts of our software implementation.

Secondly, depending on the type and structure of the subproblems different solution techniques as appropriate may be applied to solve them. In our implementation we use ILP for solving phase-1 and phase-2 subproblems. For phase-3 and phase-4 we applied CP and ILP separately and compared their performance. As we see the requirement of consecutive time-slots for some labs and tutorials make the phase-4 subproblem computationally hard. In such circumstances finding the optimal solution may be time consuming. So, we may apply constraint programming techniques to find a feasible solution within a given time limit.

Thirdly, our multi-phase implementation keeps the entire timetabling procedure transparent to the users. After a certain phase is solved, this intermediate solution may be examined and modified by the user before proceeding to solve the next phase. Thus, the multi-phase approach allows better utilization of users' expertise. Enhanced user involvement at the subproblem level results in a timetable more acceptable to the parties involved.

Finally, new constraints or last minute changes may be incorporated easily by working on only the concerned phase. In such cases, only the concerned phase and phases following it needs to be re-executed, rather than computing the entire solution starting from the scratch.

A significant contribution of this thesis is the software implementation of our multi-phase approach. Modular implementation conforming software engineering principles keeps our timetabling implementation well manageable and scalable. All architectural and computational complexities are kept hidden from the end-user behind a carefully de-

signed graphical user interface (GUI). This GUI facilitates ease in users' interaction with our timetabling implementation. The GUI presents the generated schedule to the user in a way that the user can easily interpret the solution. The flexible GUI also allows the users to customize constraints(see chapter 5) as needed.

Our timetabling implementation incorporates the flexibility to generate new schedule based on a previously saved schedule. An earlier schedule may be loaded, modified, and provided as the basis of the expected new schedule. This feature may be particularly useful when the timetable of one semester does not differ much from that of another semester. As a further example of flexibility the implementation allows pre-allocation of resources.With this feature our implementation can easily handle the issues with interdepartmental critical courses, which need to be scheduled in different times. Such courses may be centrally identified and assigned to days and time-slots before the academic departments start making their schedule. Each department then may make these prior assignments of the critical courses they offer. Keeping these assignments, the new schedule may be generated for the department.

7.2 Future Work

For further research on this work we would like to give the following suggestions.

- The current timetabling implementation takes into account courses having duration of 1 or 2 time-slots. Future work may schedule courses of longer duration.
- The assignment of classrooms to the courses may be included by incorporating additional phase(s).
- Some form of backtracking among the phases may improve the quality of the solutions.

- Finding better heuristics for subproblems may also result in better solutions.
- Automated handling of critical courses and courses across the departments may be useful. Currently the user can resolve such issues by assigning concerned resources (courses, instructors, etc.) in advance.
- More flexibility may be incorporated by allowing the user to choose solvers from among a set of solvers. Currently the ILOG's CPLEX is used for solving the integer linear programming and constraint programming models.
- Our timetabling implementation is somewhat specific to the timetabling problem at the University of Lethbridge. A more flexible implementation may be very useful if other type of institutions can use it after making necessary customizations.
- Our current timetabling implementation uses simple Microsoft Excel Spreadsheets as the sources of input data. The use of a fully featured database may be useful to store and manage timetables of different semesters.
- Our current implementation is a desktop application. It may be extended to client-server application with or without web interface to make it usable over networks.
- Elaborate experimental evaluations may discover more ways of improving the solution quality, flexibility, and usability.

Bibliography

[1] Alan K. Mackworth, J. A. Mulder, and W. S. Havens. *Hierarchical Arc Consistency: Exploiting Structured Domains in Constraint Satisfaction Problems*. Computational Intelligence, 1(3):118126, 1985.

[2] Aldy Gunawan, Kien Ming Ng and Kim Leng Poh. *A Mathematical Programming Model for A Timetabling Problem*. World Congress in Computer Science, Computer Engineering and Applied Computing, pp. 42-47, 2006.

[3] Andrea Schaef. *A Survey of Automated Timetabling*. Artificial Intelligence Review 13(2), pp. 87-127, 1999.

[4] A. Wren. *Scheduling, Timetabling and Rostering - A Special Relationship?* In the Practice and Theory of Automated Timetabling, ed. E.K. Burke and P. Ross, pp. 46-75, Springer-Verlag, 1996.

[5] C. Bessière and J. C. Régin. *Arc Consistency for General Consistenct Networks: Preliminary Results*. In proceedings of 15th International Joint Conference on Artificial Intelligence (IJCAI-97), pages 398-404, Nagoya, Japan, 1997.

[6] C. Bessière and J. C. Régin. *Refining Basic Constraint Propagation Algorithm*. In proceedings of International Joint Conference on Artificial Intelligence (IJCAI'01), pages 309-315, Seattle WA, 2001.

[7] D. Werra. *An Introduction to Timetabling*. European Journal of Operations Research 19 (1985), 151-162.

[8] E. Burke, J. Kingston, K. Jackson, R. Weare. *Automated University Timetabling: The State of the Art*. The Computer Journal 40 (9) 565-571, 1997.

[9] E.C. Freuder and R.J. Wallace. *Partial Constraint Satisfaction*. Artificial Intelligence, 58: 21-70, 1992.

[10] Edmund K. Burke and Sanja Petrovic. *Recent research directions in autmated timetabling*. Eurpean Journal of Operations Research. 140 (2002) 266-280.

[11] Gary M. Thompson. *Using Information on Unconstrained Student Demand to Improve University Course Scheduling*. Journal Of Operations Management 23 (2005) 197-208.

[12] Gautam Appa and et al. *Integrating Constraint and Integer Programming for the Orthogonal Latin Squares Problem*. P. Van Hentenryck (Ed): CP 2002, LNCS 2470, pp. 17-32, 2002.

[13] Hadrein Cambazard and et al. *Interactively Solving School Timetabling Problems Using Extensions of Constraint Programming*. E. Burke and M. Trick (Eds.): PATAT 2004, LNCS 3616, pp. 190-207, 2005.

[14] H. Paul Willium. *Model Building in Mathematical Programming*, 4th Edition. John Wiley and Sons Ltd., England, 2001.

[15] ILOG SA. ILOG OPLStudio 3.7 Reference Manual, 2003.

[16] Irvin J. Lustig and Jean-FranCois Puget. *Program Does Not Equal to Program: Constraint Programming and Its Relationship to Mathematical Programming*. INTERFACES 31: 6, pp. 29-53, 2001.

[17] Matthew L. Ginsberg. *Dynamic Backtracking*. Journal of Artificial Intelligence Research, pages 23-46, 1993.

[18] Michael R. Gary and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman, SanFrancisco, CA, 1979.

[19] Minhaz Zibran. *A Multi-phase Approach to Automated Course Timetabling*. Poster Session, CMS-MITACS Joint Conference, University of Manitoba, Winnipeg, Canada, May 31-June 03, 2007.

[20] Mini Goyal. *Graph Coloring in Sparse Derivative Matrix Computation*, M.Sc. thesis, Department of Mathematics and Computer Science, University of Lethbridge, Canada 2005.

[21] M. W. Carter. *A survey of Practical Applications of Examination Timetabling Algorithms*. Operations Research 34, 193-202, 1986.

[22] M. W. Carter and G. Laporte. *Recent Developments in Practical Course Timetabling*. In E. Burke and M. Carter (eds.), Practice and Theory of Automated Timetabling II, Lecture Notes in Computer Science, Springer-Verlag New York, 1408, pp. 3-19, 1998.

[23] Pascal Van Hentenryck and Vijay Saraswat. *Constraint Programming: Strategic Directions*. Constraints: An International Journal, 2, 7-33 (1997). Kluwer Academic Publishers, The Netherlands.

[24] Pascal Van Hentenryck and et al. *Constraint Programming in OPL*. International Conference on Principles and Practice of Declarative Programming (PPDP-99), Paris, France.

[25] Pascal Van Hentenryck. *A Preview of OPL*. Department of Computing Science and Engineering, UCL, Belgium.

[26] Philppe Galinier and Jin-Kao Hao. *Tabu Search for Maximal Constraint Satisfaction Problems*. In proceedings of 3rd International Conference on Principles and Practices of Constraint Programming, pages 196-208. Springer-Verlag LNCS 1330, 1997.

[27] Prakash Ojha, Abigail Walker, and Jennifer Wanner. *Empirical Study of Course Scheduling Methods*. 9th ACM International Student Research Contest, 2001.

[28] Rina Dechter and Daniel Frost. *Backjump-based Backtracking for Constraint Satisfaction Problems*. Artificial Intelligence, 136 (2): 147-188, 2002.

[29] R. Mohr and T. C. Henderson. *Arc and Path Consistency Revised*. Artificial Intelligence, 28 (2): 225-233, 1986.

[30] Robert J. Vanderbei. *Linear Programming Foundations and Extensions*, 2nd Edition. Springer, 2001.

[31] Roman Barták. *Constraint Programming – What is behind?* J. Figwer (editor) Proceedings of the Workshop on Constraint Programming in Decision and Control, June 1999, Poland.

[32] Roman Barták. *Dynamic Constraint Models for Planning and Scheduling Problems*. In New Trends in Constraints, LNAI 1865, pp. 237-255, Springer, 2000.

[33] Roman Barták, Tomas Müller, and Hana Rudova. *Minimal Perterbation Problem – A Formal View*. Neural Network World (2003), vol. 13, no. 5, p. 501-511.

[34] Roman Barták, Tomas Müller, and Hana Rudova. *A New Approach to Modelling and Solving Minimal Perturbation Problems*. Recent Advances in Constraints, pages 233-249. Springer Verlag LNAI 3010, 2004.

[35] S. Abdennadher and M. Marte. *University Timetabling Using Constraint Handling Rules*. Journal of Applied Artificial Intelligence, Special Issue on Constraint Handling Rules, 1999.

[36] S.A. MirHassani. *A Computational Approach to Enhancing Course Timetabling with Integer Programming*. 2005 Elsevier Inc. doi: 10.1016/j.amc.2005.07.039.

[37] Shahadat Hossain and Minhaz F. Zibran. *A Multi-phase Approach to the University Course Timetabling Problem*. 6th Cologne Twente Workshop on Graphs and Combinatorial Optimization, pp. 73-76, University of Twente, Enschede, The Netherlands, 29-31 May, 2007.

[38] Steven Minton, Mark D. Johnston, Andrew B. Philips, and Philip Laird. *Minimizing Conflicts: A Heuristic Repair Method for Constraint Satisfaction and Scheduling Problems*. Artificial Intelligence, 58: 161-205, 1992.

[39] T. B. Cooper and J. H. Kingston. *The Complexity of Timetable Construction Problems*. In the Practice and Theory of Automated Timetabling, ed. E. K. Burke and P. Ross, pp. 283-295, Springer-Verlag, 1996.

[40] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*, 2nd Edition. MIT Press, 2002.

[41] Timothy Anton Redl. *A Study of University Timetabling that Blends Graph Coloring with the Satisfaction of Various Essential and Preferential Conditions*. Ph.D. thesis, Rice University, Houston, Texas 2004.

[42] Tomas Müller. *Constraint Based Timetabling*, Ph.D. thesis, Faculty of Mathematics and Physics, Charles University of Prague, Prague 2005.

[43] Vahid Lotfi and Robert Cerveney. *A Final Exam Scheduling Package*. J. Opl Res. Soc. Vol. 42, No. 3, pp. 205 - 216, 1991

[44] Waldemar kocjan. *Dynamic Scheduling: State of the Art Report*. Technical Report T2002:28, SICS, 2002.

[45] Wayne L. Winston. *Operations Research Applications and Algorithms*, 3rd Edition. Duxbury Press, Belmont, California, USA, 1994.

[46] Zbigniew Michalewicz and David B. Fogel. *How to Solve It: Modern Heuristics*. Springer, 2000.

Appendix-A

OPL Implementation of ILP and CP Models

Model for Combined Phase-1

```
/* Assigns Instructors to Courses */

int nCourse = ...; //number of instruction-units
range Courses 0..nCourse-1;
int+ nInstructor = ...; //number of instructors
range Instructors 0..nInstructor-1;

//maximum number of instruction-units each instructor teaches
int+ instCourseLimit[Instructors] = ...;

// each instructor's preference on each instruction-unit;
// if instCoursePref[i, c] = p then preference of instruction-unit c of instructor i is p.
int+ instCoursePref[Instructors, Courses] = ...;

{int+} instGroup[0..2] = ...;

int+ nCourseGroup = ...;
range Groups 0..nCourseGroup-1;

{int} courseGroup[Groups, 0..2] = ...;

//decision variable; x[i,c] = 1 means instructor i is assigned to instruction-unit c.
var int x[Instructors, Courses] in 0..1;

maximize
   sum(i in Instructors, c in Courses) x[i, c]*instCoursePref[i, c]

subject to{
    //number of instruction-units assigned to an instructor must not exceed the limit
   forall(i in Instructors) sum(c in Courses) x[i, c] <= instCourseLimit[i];

    // each instruction-units must be assigned to exactly 1 instructor
   forall(c in Courses) sum(i in Instructors) x[i, c] = 1;

    //professors will not be assigned to tutorials
   forall(i in instGroup[0])
      forall(g in Groups)
         forall(c in courseGroup[g, 1]) x[i,c] = 0;

    //professors will not be assigned to labs
   forall(i in instGroup[0])
      forall(g in Groups)
         forall(c in courseGroup[g, 2]) x[i,c] = 0;
```

```
    //academic assistants will not be assigned to lectures
   forall(i in instGroup[1])
      forall(g in Groups)
         forall(c in courseGroup[g, 0]) x[i,c] = 0;

    //grad students will not be assigned to lectures
   forall(i in instGroup[2])
      forall(g in Groups)
         forall(c in courseGroup[g, 0]) x[i,c] = 0;

    //grad students will not be assigned to tutorials
   forall(i in instGroup[2])
      forall(g in Groups)
         forall(c in courseGroup[g, 1]) x[i,c] = 0;
};

search{
    // in the order of prof, academic assistants, grad students
   forall(ig in 0..2 ordered by increasing ig)
        // in the order of lecture, tutorials, labs
      forall(cg in 0..2 ordered by increasing cg)
         forall(g in Groups)
            forall(c in courseGroup[g,cg])
               forall(i in instGroup[ig] ordered by decreasing instCourseLimit[i])
                  try
                     x[i,c] = 1 | x[i,c] = 0
                  endtry;
};
```

Model for Phase-1a

```
/* Assigns Professors to Lectures */

int nCourse = ...; //number of instruction-units
range Courses 0..nCourse-1;
int+ nInstructor = ...; //number of instructors
range Instructors 0..nInstructor-1;

//number of instruction-units each instructor teaches
int+ instCourseLimit[Instructors] = ...;

//each instructor's preference on each instruction-unit;
//if instCoursePref[i, c]=p then preference of instruction-units c of instructor i is p.
int+ instCoursePref[Instructors, Courses] = ...;

{int+} instGroup[0..2] = ...;
{int+} profs = instGroup[0];
int+ nCourseGroup = ...;
range Groups 0..nCourseGroup-1;
{int} courseGroup[Groups, 0..2] = ...;
{int+} lectures = union(g in Groups) courseGroup[g,0];

//decision variable; x[i,c] = 1 means professor i is assigned to lecture c.
var int x[profs, lectures] in 0..1;

maximize
   sum(c in lectures, i in profs) x[i, c]*instCoursePref[i, c]

subject to{
    //number of lectures assigned to a professor must not exceed the limit
   forall(i in profs) sum(c in lectures) x[i, c] <= instCourseLimit[i];

   // each lecture must be assigned to exactly 1 professor
   forall(c in lectures) sum(i in profs) x[i, c] = 1;
};

search{
   forall(i in profs ordered by decreasing instCourseLimit[i])
      forall(c in lectures)
                  try
                     x[i,c] = 1 | x[i,c] = 0
                  endtry;

};
```

Model for Phase-1b

```
/* Assigns labs and tutorials to academic assistants and grad students */

int nCourse = ...; //number of instruction-units
range AllCourses 0..nCourse-1;
int+ nInstructor = ...; //number of instructors
range AllInstructors 0..nInstructor-1;

//maximum number of instruction-units each instructor teaches
int+ instCourseLimit[AllInstructors] = ...;

// each instructor's preference on each instruction-unit;
// if instCoursePref[i, c] = p then
// preference of instruction-unit c of instructor i is p.
int+ instCoursePref[AllInstructors, AllCourses] = ...;

{int+} instGroup[0..2] = ...;
{int+} instructors = instGroup[1] union instGroup[2];
int+ nCourseGroup = ...;
range Groups 0..nCourseGroup-1;
{int} courseGroup[Groups, 0..2] = ...;
{int+} labs = union(g in Groups) courseGroup[g, 2];
{int+} tutorials = union(g in Groups) courseGroup[g, 1];
{int+} courses = labs union tutorials;

//decision variable; x[i,c] = 1 means instructor i is assigned to instruction-unit c.
var int x[instructors, courses] in 0..1;

maximize
   sum(i in instructors, c in courses) x[i, c]*instCoursePref[i, c]

subject to{
    //number of instruction-units assigned to an instructor must not exceed the limit
   forall(i in instructors) sum(c in courses) x[i, c] = instCourseLimit[i];

    // each instruction-unit must be assigned to exactly 1 instructor
   forall(c in courses) sum(i in instructors) x[i, c] = 1;

    //grad students will not be assigned to tutorials
   forall(i in instGroup[2])
      forall(g in Groups)
         forall(c in courseGroup[g, 1]) x[i,c] = 0;
};
```

```
search{
   forall(i in instructors ordered by decreasing instCourseLimit[i])
      forall(c in courses)
                  try
                     x[i,c] = 1 | x[i,c] = 0
                  endtry;

};
```

Model for Phase-2

```
/* Assigns Lectures to Day-Sequences */

//day-sequence 0 refers to TR
//day-Sequence 1 refers to MWF

int+ nInstructor = ...; //also used in phase1
range Instructors 0..nInstructor-1; //also used in phase1
{int+} instGroup[0..2] = ...;
int+ nCourse = ...; //also used in phase1
range Courses 0..nCourse-1; //also used in phase1
int+ nDaySeq = ...;
range DaySeq 0..nDaySeq-1; // 0 means TR, 1 means MWF
int+ dayTimeSlot[DaySeq] = ...;

// 3 means MWF, 2 means TR, 1 means no preference
int+ instDaySeqPref[Instructors] = ...;

int+ nRoom = ...; //number of classrooms
int+ crsAssi[Courses] = ...;
int+ nCourseGroup = ...;
range Groups 0..nCourseGroup-1;
{int} courseGroup[Groups, 0..2] = ...;
{int+} lectures = union(g in Groups) courseGroup[g,0];

// x[c, d] = if lecture c is assigned to daySeq d
var int x[DaySeq,lectures] in 0..1;

maximize
  sum(c in lectures, d in DaySeq)
         x[d,c]
         *
         case{
            instDaySeqPref[crsAssi[c]] = 1 -> 2; // no preference
            d = 0 & instDaySeqPref[crsAssi[c]] = 2 -> 3; // TR satisfied
            d = 1 & instDaySeqPref[crsAssi[c]] = 3 -> 3; // MWF satisfied
            1 // preference violated
            }

subject to{

    // number of lectures assigned to a day sequence
    // must not exceed the available timeslot
  forall(d in DaySeq) sum(c in lectures) x[d, c] <= dayTimeSlot[d]*nRoom;
```

```
    // each lecture is assigned to exactly 1 day sequence
  forall(c in lectures) sum(d in DaySeq) x[d, c] = 1;
};

search{
      forall(c in lectures: instDaySeqPref[crsAssi[c]] > 1)
         forall(d in DaySeq: instDaySeqPref[crsAssi[c]]-d = 2)
            try
               x[d,c] = 1 | x[d,c] = 0
            endtry;

      forall(c in lectures: instDaySeqPref[crsAssi[c]] = 1)
         forall(d in DaySeq)
            try
               x[d,c] = 1 | x[d,c] = 0
            endtry;
};
```

ILP Model for Phase-3

```
/* Assigns all Lectures of a certain day-sequence to time-slots */

int+ currentDaySeq = ...;
int+ nRoom = ...; //number of available classrooms
int+ nCourse = ...;
range AllCourses 0..nCourse-1;
int+ nInstructor = ...;
range Instructors 0..nInstructor-1;
int+ nCourseGroup = ...;
range CourseGroups 0..nCourseGroup-1;
{int+}courseGroup[CourseGroups, 0..2] = ...;
int+ crsAssi[AllCourses] = ...;

// preference on morning (2) or afternoon (3) shift or no preference (1)
int+ instShiftPref[Instructors] = ...;

{int+} instGroup[0..2] = ...;
int+ nDaySeq = ...;
range DaySeq 0..nDaySeq-1;

{int+} lecturesInDaySeq[DaySeq] = ...; // set of lectures in every day-sequence
int+ dayTimeSlot[DaySeq] = ...; //number of available slots in each day-sequence
int+ dayTimeMargin[DaySeq] = ...; //slot number after which afternoon shift begins
{int+} lecturesOf2day = lecturesInDaySeq[currentDaySeq];
{int+} lecGrpOf2Day[g in CourseGroups] = courseGroup[g,0] inter lecturesOf2day;

// range of available slots in current day-sequence
range TimeSlots 0..dayTimeSlot[currentDaySeq]-1;

// x[c,t] = 1 means lecture c is assigned to time-slot t
var int x[lecturesOf2day, TimeSlots] in 0..1;

maximize
    sum(c in lecturesOf2day, t in TimeSlots)
         x[c, t]
         *
         case{
         // no preference given
         instShiftPref[crsAssi[c]] = 1 -> 2;

         //morning preference fulfilled
         t <= dayTimeMargin[currentDaySeq]
         &
```

```
        instShiftPref[crsAssi[c]] < dayTimeMargin[currentDaySeq] -> 3;

        // afternoon preference fulfilled
        t > dayTimeMargin[currentDaySeq]
        &
        instShiftPref[crsAssi[c]] >= dayTimeMargin[currentDaySeq] -> 3;

        1 // preference violated
        }

subject to{
    //each lecture is assigned to exactly one time-slot
   forall(c in lecturesOf2day)
      sum(t in TimeSlots) x[c,t] = 1;

    // number of lectures assigned to any time-slot must not exceed
    // the number of available classrooms.
   forall(t in TimeSlots)
      sum(c in lecturesOf2day) x[c,t] <= nRoom;

    //at least 1 time-slot gap between Lectures of the same instructor
   forall(c1, c2 in lecturesOf2day: c1 <> c2 & crsAssi[c1] = crsAssi[c2])
         forall(t1, t2 in TimeSlots: abs(t1-t2) < 2)
            x[c1,t1] + x[c2, t2] <= 1;

    //lectures of same course (section) non-overlapped
   forall(g in CourseGroups, t in TimeSlots)
      forall(L1,L2 in lecGrpOf2Day[g]: L1<>L2)
            x[L1,t] + x[L2,t] <= 1;
};

search{
      forall(c in lecturesOf2day: instShiftPref[crsAssi[c]] > 1)
            forall(t in TimeSlots)
               try
                  x[c, t] = 1 | x[c, t] = 0
               endtry;

      forall(c in lecturesOf2day: instShiftPref[crsAssi[c]] = 1)
            forall(t in TimeSlots)
               try
                  x[c, t] = 1 | x[c, t] = 0
               endtry;
};
```

CP Model for Phase-3

```
/*  Assigns all Lectures of a certain day-sequence to time-slots */

int+ currentDaySeq = ...;
int+ nRoom = ...; //number of available classrooms
int+ nCourse = ...;
range AllCourses 0..nCourse-1;
int+ nInstructor = ...;
range Instructors 0..nInstructor-1;
int+ nCourseGroup = ...;
range CourseGroups 0..nCourseGroup-1;
{int+}courseGroup[CourseGroups, 0..2] = ...;
int+ crsAssi[AllCourses] = ...;

// preference on morning (2) or afternoon (3) shift or no preference (1)
int+ instShiftPref[Instructors] = ...;

{int+} instGroup[0..2] = ...;
int+ nDaySeq = ...;
range DaySeq 0..nDaySeq-1;
{int+} lecturesInDaySeq[DaySeq] = ...; // set of lectures in every day-sequence
int+ dayTimeSlot[DaySeq] = ...; //number of available slots in each day-sequence
int+ dayTimeMargin[DaySeq] = ...; //slot number after which afternoon shift begins
{int+} lecturesOf2day = lecturesInDaySeq[currentDaySeq];
{int+} lecGrpOf2Day[g in CourseGroups] = courseGroup[g,0] inter lecturesOf2day;

// range of available slots in current day-sequence
range TimeSlots 0..dayTimeSlot[currentDaySeq]-1;

// x[c,t] = 1 means lecture c is assigned to time-slot t
var int x[lecturesOf2day, TimeSlots] in 0..1;

maximize
     sum(c in lecturesOf2day, t in TimeSlots)
         x[c, t]
         *
         case{
         // no preference given
         instShiftPref[crsAssi[c]] = 1 -> 2;

         //morning preference fulfilled
         t <= dayTimeMargin[currentDaySeq]
         &
         instShiftPref[crsAssi[c]] < dayTimeMargin[currentDaySeq] -> 3;
```

```
        // afternoon preference fulfilled
        t > dayTimeMargin[currentDaySeq]
        &
        instShiftPref[crsAssi[c]] >= dayTimeMargin[currentDaySeq] -> 3;

        1 // preference violated
        }

subject to{
    // each lecture is assigned to exactly one time-slot
   forall(c in lecturesOf2day)
      sum(t in TimeSlots) x[c,t] = 1;

    // number of lectures assigned to any time-slot must not exceed
    // the number of available classrooms.
   forall(t in TimeSlots)
      sum(c in lecturesOf2day) x[c,t] <= nRoom;

    // at least 1 time-slot gap between Lectures of the same instructor
   forall(c1, c2 in lecturesOf2day: c1 <> c2 & crsAssi[c1] = crsAssi[c2])
         forall(t1, t2 in TimeSlots: abs(t1-t2) < 2)
            x[c1,t1]*x[c2, t2] = 0;

    // lectures of same course (section) non-overlapped
   forall(g in CourseGroups, t in TimeSlots)
      forall(L1,L2 in lecGrpOf2Day[g]: L1<>L2)
                x[L1,t]*x[L2,t] = 0;

};

search{
      forall(c in lecturesOf2day: instShiftPref[crsAssi[c]] > 1)
            forall(t in TimeSlots)
               try
                  x[c, t] = 1 | x[c, t] = 0
               endtry;

      forall(c in lecturesOf2day: instShiftPref[crsAssi[c]] = 1)
            forall(t in TimeSlots)
               try
                  x[c, t] = 1 | x[c, t] = 0
               endtry;
};
```

ILP Model for Phase-4

```
/* Assigns labs and tutorials to time-slots */

int+ nInstructor = ...;
range AllInstructors 0..nInstructor-1;
int+ nCourse = ...;
range AllCourses 0..nCourse-1;
int+ courseDuration[AllCourses] = ...; // duration of each course
int+ crsAssi[AllCourses] = ...;
int+ instShiftPref[AllInstructors] = ...; //instructor-shift preference
int nRoom = ...;
int+ nDaySeq = ...;
int+ dayTimeSlot[0..nDaySeq-1] = ...;
int+ dayTimeMargin[0..nDaySeq-1] =...;
int+ nSlot = max(ds in 0..nDaySeq-1) dayTimeSlot[ds];
range Slots 0..nSlot-1;
int+ nDay = ...;
range Days 0..nDay-1;

//daySlotMargin[0] means TR and daySlotMargin[1] means MWF
int+ daySlotMar[d in Days]
    = case{d mod 2 = 0 -> dayTimeMargin[1]; dayTimeMargin[0]};

int+ nCourseGroup = ...;
range CourseGroups 0..nCourseGroup-1;
{int+} courseGroup[CourseGroups, 0..2] = ...; // all courses
int+ nSubGroup = ...;
range SubGroups 0..nSubGroup-1;
{int+} subGroup[SubGroups, 0..2] = ...; //all sub-groups
{int+} labsAndTut[s in SubGroups] = union(i in 1..2) subGroup[s,i];
{int+} tutorials = union(g in CourseGroups) courseGroup[g,1];
{int+} labs = union(g in CourseGroups) courseGroup[g,2];
{int+} Courses = tutorials union labs; //all labs and tutorials
{int+} instGroup[0..2] = ...;

//all academic assistants and grad students
{int+} Instructors = instGroup[1] union instGroup[2];

// schedule[d,t] contains the set of lectures assigned to time-slot t of day d
{int} schedule[Days, Slots] = ...;

int availSlotInDay[d in Days] = sum(t in Slots) (nRoom-card(schedule[d, t]));
int availRoomInSlot[d in Days, t in Slots] = nRoom-card(schedule[d,t]);
```

```
var int+ x[Courses, Days, Slots] in 0..1;

maximize
   sum(c in Courses, d in Days, t in Slots)
      x[c,d,t]
      *
      case{
        // no preference given
        instShiftPref[crsAssi[c]] = 1 -> 2;

        //morning preference fulfilled
        t <= daySlotMar[d] & instShiftPref[crsAssi[c]] < daySlotMar[d] -> 3;

        // afternoon preference fulfilled
        t > daySlotMar[d] & instShiftPref[crsAssi[c]] >= daySlotMar[d] -> 3;

        1} // preference violated

subject to{

    // each instruction-unit assigned to number of  slot of
    // exactly equal to its duration
   forall(c in Courses)
      sum(d in Days, t in Slots) x[c,d,t] = courseDuration[c];

    //number of instruction-units assigned to a slot in a day must not exceed the capacity
   forall(d in Days, t in Slots)
      sum(c in Courses) x[c,d,t] <= availRoomInSlot[d,t];

    //each instruction-unit will be assigned to consecutive slots of the same day
   forall(c in Courses: courseDuration[c] > 1)
      forall(d1, d2 in Days)
         forall(t1, t2 in Slots)
            2 - x[c,d1,t1] - x[c,d2,t2] + abs(d1-d2)
                <= 4*(2 - x[c,d1,t1] - x[c,d2,t2]) //same day
            &
            1 - x[c,d1,t1] - x[c,d2,t2] + abs(t1-t2)
                <= 7*(2 - x[c,d1,t1] - x[c,d2,t2]); //consecutive slots

    //at least 1 time-slot gap between instruction-units of the same instructor
   forall(c1, c2 in Courses: c1<>c2  & crsAssi[c1] = crsAssi[c2] )
         forall(d in Days & t1, t2 in Slots: abs(t1-t2) < 2 )
            x[c1,d,t1] + x[c2,d,t2] <= 1;
```

```
    //labs and tutorials of the same section of a course are non-overlapped
   forall(g in SubGroups, d in Days, t in Slots)
      forall(c1, c2 in labsAndTut[g]: c1<>c2)
            x[c1,d,t] + x[c2,d,t] <= 1;

    //lectures are not overlapped with labs and tutorials
    //of the same section of the same course.
   forall(g in SubGroups, d in Days, t in Slots)
      forall(Lec in subGroup[g,0]: Lec in schedule[d,t])
         forall(c in labsAndTut[g])
            x[c,d,t] = 0;
};

search{

    // first labs then tutorials

   forall(c in labs ordered by decreasing courseDuration[c])
      forall(d in Days ordered by decreasing availSlotInDay[d])
         forall(t in Slots ordered by decreasing (nRoom-card(schedule[d,t])))
            try
               x[c, d, t] = 1 | x[c,d,t] = 0
            endtry;

   forall(c in tutorials ordered by decreasing courseDuration[c])
      forall(d in Days ordered by decreasing availSlotInDay[d])
         forall(t in Slots ordered by decreasing (nRoom-card(schedule[d,t])))
            try
               x[c, d, t] = 1 | x[c,d,t] = 0
            endtry;
};
```

CP Model for Phase-4

```
/* Assigns labs and tutorials to time-slots */

int+ nInstructor = ...;
range AllInstructors 0..nInstructor-1;
int+ nCourse = ...;
range AllCourses 0..nCourse-1;
int+ courseDuration[AllCourses] = ...; // duration of each course
int+ crsAssi[AllCourses] = ...;
int+ instShiftPref[AllInstructors] = ...; //instructor-shift preference
int nRoom = ...;
int+ nDaySeq = ...;
int+ dayTimeSlot[0..nDaySeq-1] = ...;
int+ dayTimeMargin[0..nDaySeq-1] =...;
int+ nSlot = max(ds in 0..nDaySeq-1) dayTimeSlot[ds];
range Slots 0..nSlot-1;
int+ nDay = ...;
range Days 0..nDay-1;

//daySlotMargin[0] means TR and daySlotMargin[1] means MWF
int+ daySlotMar[d in Days]
     = case{d mod 2 = 0 -> dayTimeMargin[1]; dayTimeMargin[0]};

int+ nCourseGroup = ...;
range CourseGroups 0..nCourseGroup-1;
{int+} courseGroup[CourseGroups, 0..2] = ...; // all courses
int+ nSubGroup = ...;
range SubGroups 0..nSubGroup-1;
{int+} subGroup[SubGroups, 0..2] = ...; //all sub-groups
{int+} labsAndTut[s in SubGroups] = union(i in 1..2) subGroup[s,i];
{int+} tutorials = union(g in CourseGroups) courseGroup[g,1];
{int+} labs = union(g in CourseGroups) courseGroup[g,2];
{int+} Courses = tutorials union labs; //all labs and tutorials
{int+} instGroup[0..2] = ...;

//all academic assistants and grad students
{int+} Instructors = instGroup[1] union instGroup[2];

// schedule[d,t] contains the set of lectures assigned to time-slot t of day d
{int} schedule[Days, Slots] = ...;

int availSlotInDay[d in Days] = sum(t in Slots) (nRoom-card(schedule[d, t]));

var int+ x[Courses, Days, Slots] in 0..1;
```

```
maximize
   sum(c in Courses, d in Days, t in Slots)
      x[c,d,t]
      *
      case{
      // no preference given
      instShiftPref[crsAssi[c]] = 1 -> 2;

      //morning preference fulfilled
      t <= daySlotMar[d] & instShiftPref[crsAssi[c]] < daySlotMar[d] -> 3;

      // afternoon preference fulfilled
      t > daySlotMar[d] & instShiftPref[crsAssi[c]] >= daySlotMar[d] -> 3;

      1 // preference violated
      }

subject to{

    // each instruction-unit assigned to number of  slots
    // exactly equal to its duration
   forall(c in Courses)
      sum(d in Days, t in Slots) x[c,d,t] = courseDuration[c];

    //number of instruction-units assigned to a slot of a day must not exceed the capacity
   forall(d in Days, t in Slots)
      sum(c in Courses) x[c,d,t] <= nRoom-card(schedule[d,t]);

    //each instruction-unit will be assigned to consecutive timeslots of the same day
   forall(c in Courses: courseDuration[c] > 1)
      forall(d1, d2 in Days)
         forall(t1, t2 in Slots)
            x[c,d1,t1]*x[c,d2,t2] = 1 => d1 = d2 & abs(t1-t2) < 2;

    //at least 1 time-slot gap between instruction-units of the same instructor
   forall(c1, c2 in Courses: c1<>c2  & crsAssi[c1] = crsAssi[c2] )
         forall(d in Days & t1, t2 in Slots: abs(t1-t2) < 2 )
            x[c1,d,t1]*x[c2,d,t2] = 0;

    //labs and tutorials of the same section of a course are non-overlapped
   forall(g in SubGroups, d in Days, t in Slots)
      forall(c1, c2 in labsAndTut[g]: c1<>c2)
            x[c1,d,t]*x[c2,d,t] = 0;
```

```
    //lectures are not overlapped with labs and tutorials
    //of the same section of the same course
   forall(g in SubGroups, d in Days, t in Slots)
      forall(Lec in subGroup[g,0]: Lec in schedule[d,t])
         forall(c in labsAndTut[g])
            x[c,d,t] = 0;
};

search{
    // first labs then tutorials

   forall(c in labs ordered by decreasing courseDuration[c])
      forall(d in Days ordered by decreasing availSlotInDay[d])
         forall(t in Slots ordered by decreasing (nRoom-card(schedule[d,t])))
            try
               x[c, d, t] = 1 | x[c,d,t] = 0
            endtry;

   forall(c in tutorials ordered by decreasing courseDuration[c])
      forall(d in Days ordered by decreasing availSlotInDay[d])
         forall(t in Slots ordered by decreasing (nRoom-card(schedule[d,t])))
            try
               x[c, d, t] = 1 | x[c,d,t] = 0
            endtry;
};
```

Appendix-B

CD-ROM Content

This thesis includes a companion CD-ROM containing the electronic form of this thesis, the source codes of our timetabling implementation, the timetabling data generator, and the test data we use in our experiments. Given below is the list of directories and their contents.

Directory	Content
Thesis	Electronic version of this thesis in portable document format (pdf) and the LaTex files.
UofLTimetable	Final version of the timetabling implementation, with all modules kept in directory structure as required.
UofL_OPL\UofLPhase1	OPL implementation of combined phase-1 model.
UofL_OPL\UofLPhase1a	OPL implementation of phase-1a model.
UofL_OPL\UofLPhase1b	OPL implementation of phase-1b model.
UofL_OPL\UofLPhase2	OPL implementation of phase-2 model.
UofL_OPL\UofLPhase3_IP	OPL implementation of the ILP model for phase-3.
UofL_OPL\UofLPhase3_CP	OPL implementation of the CP model for phase-3.
UofL_OPL\UofLPhase4_IP	OPL implementation of the ILP model for phase-4.
UofL_OPL\UofLPhase4_CP	OPL implementation of the CP model for phase-3.
UofLC++	Source code of implementation of the C++ module.
UofLJava	Source code of Java implementation of the GUI.
SchedDataGen	Source code of Java implementation of the timetabling data generator.
dataConvert	Source code of the Java program that reads tabs and new lines separated timetabling data from text file and writes to MS Excell Spreadsheet.
ttData	Real world timetabling data (stored in text files) separated by tabs and new lines.
RandData	Pseudo-randomly generated timetabling data used in the experiments.
RealData	The real world data used in the experiment.

Table 7.1: Content of the companion CD-ROM