Defining, Modeling, and Solving a Real University Course Timetabling Problem

by

Shoshana Hahn-Goldberg

A thesis submitted in conformity with the requirements

for the degree of Masters of Applied Science

Graduate Department of Mechanical and Industrial Engineering

University of Toronto

Abstract

Defining, Modeling, and Solving a Real University Course Timetabling Problem

Shoshana Hahn-Goldberg
Master of Applied Science
Graduate Department of Mechanical and Industrial Engineering
University of Toronto
2007

The central thesis of this dissertation is that the problem definition stage of solving real world problems should be directly studied and that problems must be studied in their entirety to create useful solutions. The problem definition stage has been identified as important yet is not directly studied in operations research (OR). This work is an introduction of such research to OR. Timetabling has received much attention from researchers and this work is a continuation of such effort.

We conduct an analysis of the timetabling problem at the Faculty of Applied Science and Engineering at the University of Toronto (APSC), showing how difficult it would be to create a definition for a mathematical model. We also create evaluation criteria for APSC and show that quality measures in an objective function cannot accurately represent the desired metrics. Finally, we apply mathematical programming and decomposition techniques to some benchmark timetabling problems.

Acknowledgments

This work would not have been possible without the help of many people, notably:

- My husband for his love and support. Also, thank you for your help with the programming involved in this work.
- My parents for always being there for me for whatever I needed.
- My wonderful babysitters for giving me the time I needed to finish this research.
- Professor Beck for his insight and direction. Also, thank you for your help with the editing.
- Professor Frances for arranging this opportunity for me and for his encouragement and assistance.
- Leslie Beckskei and Sandy Walker for their availability and invaluable knowledge.

I would also like to thank Ontario Graduate Scholarship for their financial support.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The central thesis of this dissertation is that the problem definition stage of solving real world problems should be directly studied and that the problems must be studied in their entirety in order to create a solution that is truly useful in the real world. This is shown through the use of university course timetabling problems and the Faculty of Applied Science and Engineering at the University of Toronto's timetable in particular. The problem definition stage has been identified as important and is studied in depth in both software engineering and enterprise modeling, yet this is not the case in the area of Operations Research (OR). This work is an introduction of such research into the OR field. University course timetabling has received much attention from researchers in both the OR and Artificial Intelligence (AI) fields and this work is a continuation of such effort. In particular, in this dissertation:

- We conduct a thorough analysis of the university course timetabling problem at the Faculty of Applied Science and Engineering at the University of Toronto (APSC) as an example of a real world problem in operations research.

- We create detailed evaluation criteria for the APSC timetable.

- Motivated by the constraint structure of university course timetabling problems, we apply constraint programming (CP), Integer Programming (IP), and decomposition techniques to a benchmark university course timetabling problem found in the literature. The problem has a similar structure to the APSC problem.

1.1 Background Information

The real world is full of complex situations, addressed by a wide variety of research domains; from zoology to electricity to operations research, the focus of this thesis. In operations research, problems can be anything from an optimization of a factory [79, 80], a schedule for a job shop [81, 82], a transportation routing problem [83, 84], or even a university timetable [11, 66]. The approaches used to solve these problems can range from integer [66, 84], mixed-integer [67, 82], or constraint programming [63, 68] to simulation [79] and heuristics such as tabu search [13] and other local search techniques [5, 83].

In order to model a problem or put a problem into a language that a given approach can solve, the problem definition must be formulated; i.e. one must decide what information

to include when creating a model to solve the problem. It is also important to be able to evaluate the model created by a given problem definition in the real world. Therefore, one must decide how to evaluate a solution when defining the problem. A certain set of evaluation criteria can dictate what information is important to include in the problem definition. Once the evaluation criteria are known and the problem definition is made, a model can be created to solve the problem. We can then take the solution from that model and see how it fits back into the actual system, the real world situation.

This thesis looks at the timetabling problem at the Faculty of Applied Science and Engineering at the University of Toronto (APSC) as an example of a real world problem. As with many real life problems, the university course timetabling problem can be messy and complicated. Solving the timetabling problem at APSC involves many people communicating to try to achieve a timetable that meets some set of requirements and goals. The literature on automated timetabling takes a given timetabling problem and reduces it to a mathematical definition, which can then be solved. In reality, the timetabling process is long and consists of many stages before that of actually placing courses into timeslots. To automate the university course timetabling process, one must consider this entire process, and not just the scheduling part of it.

1.2 Objectives

This research investigates the process of solving a real world problem. This process can be broken down into three steps after which a solution can be generated. The first two steps are both parts of the problem formulation stage [85].

1. Analyzing the process currently in place for solving the problem. This includes a detailed study of the system, data collection, and identification of problematic areas, as well as, system constraints, restrictions, and objectives.

2. Determining evaluation criteria and creating a problem definition. This is the construction of an abstraction of the problem that can be mathematically modeled.

3. Developing a model to solve the problem. It is important to remember that solutions obtained from the model are solutions to the model and not necessarily solutions to the real-world problem. How well the solution to the model fits in the real world is dependent on the specific problem definition.

The objectives of this research are as follows:

- Introduce the step of creating a problem definition as something that should be directly studied in operations research.

- Use the timetabling problem of the Faculty of Applied Science and Engineering at the University of Toronto (APSC) to investigate the process of solving a real world problem.

- Make contributions to each of the three steps of solving a real world problem outlined above.

    1) Create a detailed process definition for the timetabling problem at APSC. This description can be used to analyze and improve the process. Areas where automation can be helpful are identified.
    2) Determine evaluation criteria for APSC. These evaluation criteria create objectives when timetabling. Together with the process description, the evaluation criteria can be used to create a problem definition.
    3) Create models to solve a university course timetabling problem. Investigate the use of constraint programming for solving a university course timetabling problem.

1.3 Thesis Organization

The Thesis is organized as follows:

Chapter 2 introduces the problem definition concept to the operations research field. It does so through the use of an example, which shows how the concept of a problem definition is part of the operations research problem solving process, yet it is not directly studied in the research.

Chapter 3 is a literature review. It looks at the process of creating a problem definition in the areas of operations research, model-based diagnosis, bond graphs, software engineering, and enterprise modeling. Both defining the problem and validating the model will be discussed. The step of creating the problem definition is studied directly in many fields. However, in the field of operations research, although it is present, it is not directly studied.

Chapter 4 provides a detailed description of the process used to solve the university course timetabling problem at APSC. It also outlines the goals and constraints of the problem and highlights areas where automation would be helpful. The entire timetabling process can be thought of as the largest possible problem definition because everything is included. In order to apply automated solutions to the problem one would most likely require an abstraction of such a large definition.

Chapter 5 details the evaluation criteria for the timetabling problem at APSC. It details the process involved in creating the evaluation criteria along with a set of database queries that calculate several quality metrics based on a set of data provided by APSC. Creating evaluation criteria is an important part of creating a problem definition. It must be clear how solutions generated from a model based on a particular problem definition will be validated and evaluated back in the real world.

Chapter 6 acts as an introduction to the next stage in the problem solving process as well as an introduction to the next chapter of the thesis. Once the problem definition is

created, a mathematical model can be created to generate an actual timetable. The next chapter of the thesis focuses on the modeling step of problem solving. This chapter is a literature review of research existing on automated university course timetabling.

Chapter 7 provides a description of six models created to solve a university course timetabling problem. It also describes the experiments run on these models.

Chapter 8 provides the conclusions and highlights the major contributions of the research. It concludes with suggestions for future research, building from the thesis.

1.3 Summary of Contributions

This thesis looks at problem solving in the domain of operations research through the example of university course timetabling. The three steps of solving a real world problem are outlined and contributions are made to each. The three steps in solving a real world problem are (1) to analyze the problem, (2) to develop a problem definition and evaluation criteria, and (3) to model and solve the problem. The contributions made by this thesis follow the problem solving process.

- First, to address the step of analyzing the problem domain, the problem of timetabling at APSC is analyzed. A detailed process description is made and problem areas, specifically ones where automation may be helpful, are highlighted. Solutions are suggested for all problematic areas. The main contribution to this area of problem solving is in taking a real world problem and going in detail over the process of how the problem is solved. By looking at the timetabling problem at APSC, we show that real world problems are much more complicated than what typically appears in a mathematical model or in a typical research paper on timetabling. The complexity of the APSC problem emphasizes how difficult, if not impossible, it is to come up with a definition of an optimization problem that could be used to define a mathematical model. The complexity of the APSC problem is also motivation for research into how to define a problem, a problem-solving step that is not directly studied in the operations research domain.

- Second, to address the step of creating a problem definition, evaluation criteria are created for APSC. These evaluation criteria are implemented in the form of a Microsoft Access database that can score the quality of a timetable created through their current process. Creating a complete problem definition is a vital step in the problem solving process. The creation of evaluation criteria is the creation of a part of a problem definition. The evaluation criteria are complex and may be difficult to incorporate into a traditional optimization function. Since that is the case, it is necessary to evaluate how a solution works. In the APSC case, a detailed set of evaluation criteria is useful and necessary if one is to find an automated timetabling solution.

- Thirdly, to address the step of modeling and solving the problem, six mathematical programming models are created to solve a university course timetabling problem of similar style to that of the APSC, which was studied in the first part of the thesis. These six models experiment particularly with Constraint Programming (CP) and decomposition techniques. These are ideas that have not been explored as of yet in the automated timetabling research. The value of CP and decomposition in the automated timetabling domain is studied and compared to the popular solution techniques of Integer Programming (IP) and local search.

Chapter 2

The Problem Definition Stage in Operations Research

2.1 Introduction

In operations research, real world problems are often solved by analyzing and modeling them so that they can be solved by some sort of mathematical program or heuristic. These problems can be anything from an optimization of a plant, a schedule for a job shop, a transportation routing problem, or even a university timetable. The programs used to solve these problems can range from integer, mixed-integer, or constraint programming to simulation and also heuristics such as tabu search and other local search techniques. In order to model a problem or put a problem into a language that a program can solve, the problem definition must be formulated. This step of formulating the problem is called the problem definition stage.

2.2 The Problem Definition Stage: A Definition

A problem definition is created by picking the information from the problem domain that should be represented in the model. The real world situation, no doubt, contains many details. Including every detail in the representation could result in a problem that may not be solvable by any program or may take an unrealistically long time to solve. The goal is to choose a level of detail that produces a model that accurately represents the problem and is not too complex that it is overly difficult or costly to model and solve. This desired level of abstraction is often reached using simplifying assumptions.

A specific combination of assumptions creates a specific problem definition. These definitions, although they describe the same real world problem are not mathematically equivalent. Each problem definition will create a different mathematical model and will result in a different solution to the problem. This solution must be checked to ensure that it works back in the real world. It is also valuable to learn what implications choosing one problem definition over another have on the solution and whether one is in fact better than another. One can imagine that there are sets of assumptions that will create a solution that is unusable as well as sets of assumptions that do not simplify the problem enough to make any difference.

In the following section, we illustrate the concept of a problem definition stage and how different problem definitions create different solutions using a simple example.

2.3 An Example: LP Relaxation

The above concept can be illustrated using the example of an integer program (IP). An integer program is one where the variables that are being solved for are restricted to integer values. IP's are very common because in reality many decisions are discrete, yes or no, decisions. IPs can be very difficult to solve since there is no generic and computationally effective algorithm for solving them. A classic assumption to simplify an IP is to relax the integrality constraint and formulate the IP as a linear program (LP). The IP is one problem definition and the LP is another problem definition for the same real world problem. The advantage of having this simpler problem definition is that the LP can be solved easily using well-known algorithms such as the simplex method. The resultant solution from such a relaxation may be an integer solution, in which case the simplifying assumption still had a solution that could be used to solve the problem in the real world. The LP could also result in a solution with fractional values that may or may not be usable.

2.3.1 Problem Definitions

The following example, which will further illustrate the problem definition concept, is taken from J. E. Beasley's OR-Notes [59]. It is a capital budgeting problem where a company has to choose from four possible projects. They will each run for three years and are subject to the following data:

Figure 2.1. Data for the capital budgeting problem example.

		Capital requirements (in millions of dollars)		
Project	**Return (millions)**	**Year 1**	**Year 2**	**Year 3**
1	0.2	0.5	0.3	0.2
2	0.3	1.0	0.8	0.2
3	0.5	1.5	1.5	0.3
4	0.1	0.1	0.4	0.1
Available Capital (millions)		3.1	2.5	0.4

The company's goal is to decide which projects to take on in order to maximize the return. The formulated IP is as follows:

Figure 2.2. IP formulation for the capital budgeting problem example.

The variables are:

x_j = 1 if we decide to do project j (j = 1, 2, 3, 4)

= 0 otherwise.

The objective function is:

Maximize $0.2x_1 + 0.3x_2 + 0.5x_3 + 0.1x_4$

It is subject to constraints of availability of funds each year:

$0.5x_1 + 1.0x_2 + 1.5x_3 + 0.1x_4 = 3.1$ (year 1)

$0.3x_1 + 0.8x_2 + 1.5x_3 + 0.4x_4 = 2.5$ (year 2)

$0.2x_1 + 0.2x_2 + 0.3x_3 + 0.1x_4 = 0.4$ (year 3)

And the integer constraint:

$x_j = 0$ or 1, j = 1, 2, 3, 4.

The above problem can be defined as is, where the variables must be integer, because that is how the problem exists in the real world. The company cannot decide to do a fraction of a project; it is a go/no-go decision. However, this problem can be defined in another way, which is simpler and therefore may be easier to solve. It can be defined in such a way that the variables do not have to be integer. The model resulting from such a problem definition is the same as above with the integer constraint replaced with the following continuous constraints:

$x_j = 1$, j = 1, 2, 3, 4

$x_j = 0$, j = 1, 2, 3, 4.

2.3.2 Evaluating Different Problem Definitions

The next step is to see if this second problem definition results in a solution that is useful in the real world. As it turns out, in this case the solution is not naturally integer, so the solution is different from that of the IP definition of the problem. The solution of the IP was: $x_1 = 0$, $x_2 = 0$, $x_3 = 1$, and $x_4 = 1$ for a return of 0.6. The solution of the LP was: $x_1 = 0$, $x_2 = 0.5$, $x_3 = 1$, and $x_4 = 0$. Sometimes, the LP can be useful even if it is not naturally integer by rounding the values to the nearest integer and getting a feasible solution that way. Here, if

we round up and accept projects 2 and 3, the available capital constraint for year 3 is violated. This is an infeasible solution, but it may still be usable, if the company is willing to adjust their allocation of capital. We can also round down and only accept project 3. We then get a feasible solution with an objective function value of 0.5. This solution, although feasible, is not optimal. It has an objective function value that is less than the optimal value of 0.6.

It is evident from this small example that different problem definitions can result in different solutions to the same problem. In the example given, an LP relaxation is used as a simplified problem definition of a problem that required integer values and it resulted in a solution that was not feasible in the real world.

In general, an LP relaxation could result in a very usable solution if it is naturally integer. There has been research done and it is known that certain types of problems are naturally integer, such as network optimization problems [60]. When a problem can be formulated as a network problem, the LP relaxation will result in the optimal solution because the problem is naturally integer. However, if the problem is not one that is naturally integer, it may or may not result in a usable solution. It may result in a somewhat usable solution if rounding up or down can give an idea of what the optimal values are. It may also result in a solution that is useless, with the rounded LP solution being very far away from the optimal integer solution.

2.4 Conclusion

The problem definition stage consists of choosing what information to include in a model of a real situation. It is deciding what simplifying assumptions to make. It is settling on a level of abstraction that accurately represents the domain, but is not too complex that it is too hard to solve in a reasonable amount of time. There is more than one problem definition for each real world problem and, as shown above, a different problem definition can result in a different solution to the same problem. Each problem definition may be useful in a different way for solving a given problem. Some problem definitions may result a solution that is usable or preferable in the real world while some will result in a solution that is completely useless. Different problem definitions can be analyzed to see what implications a given definition has on the resulting solution in comparison to another. The problem definitions can also be evaluated to see if and how their respective solutions are usable in the real world.

As we will see in the next chapter, the problem definition stage is studied in several research domains, but has been overlooked in the Operations Research literature. This is surprising because it seems to be a fundamental concept when applying optimization techniques to the real world. Formulating the problem definition is, in fact, deciding what information should be represented in a given abstraction, thereby driving the information engineering aspects of real world applications. The problem definition stage exists in optimization problems, but its direct study as well as what implications different problem definitions have on the resulting solution is missing from the literature.

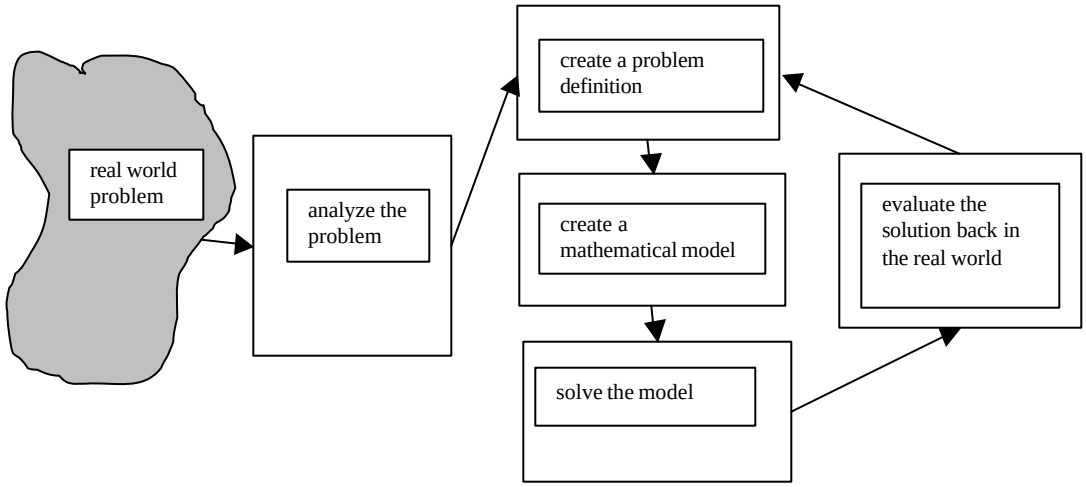
Chapter 3

The Problem Definition Stage - Literature Review

3.1 Introduction

Solving a real-world problem such as a university timetable, diagnosing faults in a machine, or creating a knowledge-based decision support system is often quite difficult. This is due to the complex nature of many real-world situations. They include many details and are often extremely complex, so much so that if all the details were to be taken into account, the problem would not be solvable. When solving a real-world problem, or any other problem for that matter, there are many stages that must be traversed. First is the problem formulation stage. This first stage can be broken down into two parts [85]. The first part is to conduct a detailed study of the system currently in place, including identification of issues, constraints, restrictions, and goals. For the second part, the problem definition must be formulated, in what we will be referring to as the "problem definition" stage. As we described in the previous chapter, this is the act of describing the real-world domain by deciding what information to include in a mathematical model. After the problem formulation stage there is a modeling and solution stage, where a problem definition is put into a language that the program or heuristic can use. Finally, the mathematical program, computer program, or heuristic is used to find a solution. That solution can then be evaluated in the real world. If the solution cannot be used, the problem definition may need to be restructured and the process will continue from there. The following is a diagram of the problem solving process.

Figure 3.1. The Problem Solving Process.



Including all possible information during the problem definition stage is often impossible because it would result in a problem that is not solvable in a reasonable amount of time. For example, in university timetabling problems information such as certain preferences for rooms or information having to do with the distance that students have to travel between consecutive classes is often omitted. This is because including such information as constraints would make the problem too complicated to solve. As well, there is the overhead of representing and maintaining all the information that is included in the problem definition. Therefore, it is preferred to represent as little as possible, especially because the data might change or it may not be completely accurate to begin with. The goal is to pick the level of detail that produces a model that represents the domain at the right level of abstraction. The right level of abstraction means that the model is solvable in a reasonable amount of time and that the solution can be used in the real world.

It is important to be able to evaluate the model created by a given problem definition back in the real world. Does the model accurately represent the domain? Is the level of detail sufficient? Is the problem solvable in a reasonable amount of time and can that solution be used? Often, models have objective functions, some sort of cost function, or a set of optimization criteria. However, those optimization criteria are unique to a given model and they measure how well the model does at meeting certain calculated measures and not necessarily how well the solution to the model works in the real world. Measures to represent the quality of a solution in the real world need to be defined so that models can be evaluated. Such measures will be referred to as evaluation criteria. These measures are reflective of some aspect of real world quality, yet they may not be represented well in a mathematical model. Evaluation criteria can be used to compare models to see what effect choosing one problem definition over another has; how does a different problem definition impact the usefulness or quality of a solution.

When solving an optimization problem, the problem definition stage is a necessary step. However, in operations research, the problem definition stage is not formalized. In the following sections, we look at the problem definition stage in the areas of model-based diagnosis, software engineering, enterprise modeling, and operations research. Both defining the problem and validating the model will be discussed. We will also look at one technique that exists for modeling real world systems, namely bond graphs.

3.2 Model-Based Diagnosis

Model-based diagnosis uses mathematical models of machines and systems in order to diagnose faults. This requires having an accurate model of the machine or system. Model-based diagnosis takes a model of the correctly behaving machine and makes a diagnosis by looking at the abnormal observations given and producing hypotheses as to the faults that lead to abnormal behaviour. In the design of a domain model, one must choose a level of detail that represents the object accurately, yet is not too difficult to represent and has an acceptable computational complexity [51]. There is a point where adding more detail does not result in enough new accurate hypotheses that it justifies the increased complexity. In the field of model-based diagnosis, a lot is written about how to simplify problems so they result

in models that accurately represent the domain and that the solutions work well in the real world. There is research done on how to represent the complex information in such a way that the resulting model is not too complex. In other words, there is a lot written on how to create problem definitions.

One way to represent complex information is to use qualitative reasoning [28, 30, 32]. The way that a system is represented can affect the accuracy of the model and the claim is that a qualitative representation will result in a more accurate model. In a paper by Museros & Escrig a method is given to represent shapes qualitatively [28]. Numerical methods often use piecewise interpolation, which is sometimes too much of a simplification. Often numerical methods require the use of data sets and if those sets are inadequate, the model may not be able to reproduce system dynamics. Guglielmann & Ironi explain how using data models for fuzzy systems can result in unstable models if inadequate data is available, but using qualitative reasoning allows the structure and the parameters to be modeled separately and therefore minimizes the effect of inadequate data [32]. Keppens & Shen look at using Bayesian networks to accurately represent a domain [31]. This is useful because most definitions assume that system behavior is deterministic as a way to simplify the problem. Assuming that a system behaves deterministically works fine for most physical systems where the knowledge of how they work is quite complete, however, sometimes the assumption is too much of a simplification. Bayesian networks include many more options for why a certain scenario has come to be using probabilities. Alonso et al. use machine learning techniques incorporated with typical consistency-based diagnosis in order to represent the system in such a way that the reason for faults can be uncovered [33].

Complex information can still be used without over-complicating the model if it is modeled separately from the actual domain model used for diagnosis. One option is to combine heuristic and model-based diagnosis [51]. Heuristic diagnosis uses rules to diagnose. It is fast but may leave things out. Model-based diagnosis can take a lot of time, but is more accurate. Andersson uses two theories [51]: the object theory contains the model of the domain in its correct state and the meta-theory contains the heuristic rules as well as the complex information. The complex information is kept separate so it does not increase the level of detail involved in the actual model, but the information can still be used when refuting hypotheses during fault diagnosis. This way, one can run a simpler model and it will most likely be faster and require less memory. However, the complex information can be used to check the solution resulting from the simpler model and if necessary, a more complex model can be run.

Another way that definitions used for model-based diagnosis are simplified is by using the single fault assumption [47]. This assumption either requires the problem be defined in such a way that every possible fault combination is modeled separately as a component that may fail or it restricts the solver to problems of simple, routine diagnosis [52]. There are papers that try to find other ways to represent multiple fault types without having to explicitly write out every possible fault combination. For example, Nyberg represents system fault modes as a vector of component fault modes [48].

In model-based diagnosis there are many assumptions that are made to link the model with the actual problem, to deal with issues of complexity, and to be able to find solutions in a reasonable amount of time [52]. Assuming that domain models are accurate and complete and that the design of the model is correct are assumptions that have to do with linking the model to the real-world problem [52]. The single fault assumption is an assumption that deals with complexity. Being able to model using a hierarchal structure, so that more detailed levels of abstraction need only be explored for certain components, and assuming that probabilities exist, in regards to why faults occur are assumptions that reduce search time [52]. Including certain assumptions as opposed to others results in a different problem definition and therefore a different model.

In the model-based diagnosis research, little is written on evaluation criteria and comparing how different problem definitions play out in the real world. The models are validated either through an example [31, 48, 51] or using made up test data that is supposed to represent reality [33]. There are some cases where the model is tested in the real world [28] or used [30].

The choice of which assumptions to include and at what level of abstraction to design is the problem definition stage. This stage is obviously present in model-based diagnosis. Although the stage exists and there is quite a bit written on assumptions that may be necessary, there does not appear to be research comparing different problem definitions against each other.

3.3 Software Engineering

When creating software, one needs models of required data, information and control flow, and behavior [40]. Creating these models requires an understanding of what is required by the system. These requirements are received from clients and potential users of the system and are generally given in vague terms that need to be turned into specifications. Specifications are abstractions of real or envisioned situations that are normally quite complex, resulting in specifications that are incomplete and exist at many levels of detail [40]. Deciding which requirements to turn into specifications and at what level of detail is called the requirements definition stage of software engineering. This task is, in effect, the same as the problem definition stage discussed earlier.

Nuseibeh & Easterbrook describe requirements engineering as a way to anchor development activities to a real-world problem so the appropriateness of a solution can be analyzed [42]. They describe the act of creating requirements as a construction of abstract descriptions that are amenable to interpretation. They go through the many different modeling tasks of requirements modeling:

- Enterprise modeling - to capture the purpose of a system where high-level goals are repeatedly refined.
- Data modeling - where decisions are made as to what information to represent and how the information held corresponds to the real world phenomena being represented.

- Domain modeling - where a model of the domain provides an abstract description of the world in which an envisioned system will operate.
- Modeling non-functional requirements.

Software architecture is the high level abstraction of a software system that provides a way to document boundaries and constrain which parts rely on other parts [43]. Each architecture style makes assumptions. For example, publish-subscribe assumes that event delivery is reliable, centralized routing is sufficient, and that a common vocabulary makes sense [43]. Each style is appropriate for certain purposes due to its particular simplifying assumptions. By choosing to define the problem in a particular way, one effects the requirements definition as well. Garlan mentions that an area of future research is how software architectural choices effect the prioritization and evolution of requirements [43].

Software engineering is one area where there has been a lot of work done on process models. The idea of a problem definition stage as well as validation of the definition in the real world is quite well-established. Following detailed processes to formulate the problem definition is beneficial to the field of software engineering. It helps ensure successful projects by being organized early on in the process and testing that the system works. This is necessary for the large and difficult projects encountered in the software engineering field, but it may be beneficial in other fields as well. Perhaps following a detailed procedure in operations research, when formulating the problem, would help in solving difficult, real-world optimization problems.

In software engineering, there is the idea of process simulation that tries to evaluate the performance of a given model [53, 54]. There are many developed process models that have detailed descriptions of how and when to collect requirements and how to analyze those requirements in order to decide which are the most important and how they should be included in the requirements definition. The process models also include testing. System tests, unit tests, and acceptance tests are just a few of the tests whose purpose is to check how the system works in the real world. The process models formalize the requirements definition, or in other words, the problem definition stage, in the context of the whole software development process. For example, two such process models are the classic waterfall model and the newer extreme programming model.

The waterfall method follows a detailed process that begins with a feasibility study of the whole project. This is followed by requirements analysis and specification of the requirements for the entire system, a design and specification stage, coding and module testing, integration and system testing, and finally, delivery and maintenance [53]. Extreme programming, on the other hand, is an incremental and iterative process. The development team produces “stories”; descriptions of interactions with the system that they release to the customer every two weeks. Software functionality is built so that the story can be achieved. There is a planning act at the start of the project that looks at the project as a whole, but it is quite vague. The detailed planning, called iteration planning, occurs every two weeks and focuses on the requirements for the next software release date. The customer, at each planning session, provides the requirements, sets priorities, and defines tests that the

software must pass at the next release date [58]. This way, the requirements are being continuously defined and the system is being continuously validated.

In software engineering, not only is there research on process models, there has also been work done on which processes are applicable to which types of projects. For example, a less risky project would benefit from a process model like extreme programming, which involves a lot of working prototypes early in the process and getting requirements from the users and testing system functionality at the same time [53]. A riskier project, however, would benefit more from a structured process such as the classic waterfall method [53]. It would be useful, when tackling a real world optimization problem, to know what sorts of simplifying assumptions are useful given the nature of the problem. It would be nice to have processes in place to follow in order to create problem definitions that lead to solvable models with solutions that work in the real world.

3.4 Enterprise Modeling

Enterprise modeling is another area where the problem definition stage is established. Enterprises need to be agile and integrated across their functions in order to stay competitive. Enterprise models enable this by promoting better design and analysis of enterprise practices [55]. A deductive enterprise model (DEM) that is given a proper model of the enterprise can answer common sense questions about the enterprise and thereby reduce management information system backlog [55]. The model needs to accurately represent the enterprise for a DEM to work. In a paper by Gruninger & Fox, the authors want to create formal representations of the knowledge found in enterprise engineering perspectives using ontologies [57]. An ontology identifies objects in a domain along with the properties of those objects and the relations between them. A micro theory defines the set of axioms to represent the constraints on the ontology. In enterprise modeling, ontologies have to represent concepts such as activity, time, and resource.

In enterprise modeling, the idea of having different problem definitions as well as comparing and evaluating them is strongly present. Fox & Gruninger reason about alternative designs for an enterprise in [55]. Different sets of constraints need to be considered. One must see if a process can be performed differently or if a constraint can be relaxed to improve performance. The impact of changes has to be known for all parts of the enterprise. How quality is affected by a relaxation is considered as well. Such information would be very useful in real world optimization problems.

Enterprise models are evaluated and validated through the use of competency questions, the set of queries that the enterprise model can answer [57]. The competency questions represent tasks that an ontology can represent and solve. The ontology represents tasks and their solutions using a set of axioms[57]. If the set of queries that a DEM can answer can be reduced to the competency questions, then the DEM is known to be sufficient. Also, the competency questions can be used to determine if a DEM is precise enough or if it allows for abstractions [55].

There is more than one way to represent knowledge and each way has different computational complexity when answering a specific set of competency questions [55]. The idea of having different ways to represent the same knowledge is the problem definition stage and the competency questions are a method to compare and evaluate given definitions and models. It is obvious that these ideas are considered important in the field of enterprise engineering.

3.5 Operations Research

In the field of operations research, there are many tasks that require creating a model of a real-world domain in order to solve a real-world problem. Tasks such as creating a timetable for a university or analyzing a queuing system are but two in a list of many. As mentioned earlier, when representing a real world problem, simplifying assumptions are almost always necessary. In operations research, the problem definition stage, where simplifications are made, exists, but the direct study of this stage is, for the most part, absent from the literature.

When analyzing queuing models, it is often desirable to represent arrivals and service times in the system using pre-defined probability distributions or models. In a paper by Brandao & Porta Nova, an Auto Regressive Integrated Moving Average (ARIMA) model, a model used to forecast a time series, is used to represent a queue with utilization greater than or equal to one [37]. This representation is simpler than using all of the actual data, yet still realistic, and it is therefore easier to analyze. Graphs are provided showing how close the distribution is to reality, but the model does not represent the system exactly and what implication this has in terms of results is not discussed.

In the area of automated timetabling, the problems are known to be NP-hard and therefore heuristic searches are often used to find solutions. When dealing with heuristic solutions, there is the idea of relaxing what would be hard constraints in order to increase the flexibility for moving around the search space [13]. This is a way of simplifying the problem by changing its definition. Such relaxations of hard constraints may result in a solution that, although it may have been found faster and it appears to be better as far as the cost function is concerned, may not be useful in the real world. This may be because the combination of constraints that are violated makes for a worse solution in the real world than a different solution, with more constraint violations. In a paper by Cambazard et al., a solution method for creating a timetable starts with a model of the problem with all constraints being hard; this is at the lowest level of abstraction [3]. If that model is found to be over-constrained, a search is done in the space of possible relaxations, in other words the space of different restricted problem definitions, to find a definition that both accurately represents the real problem and is solvable. A more common phenomenon is for a human scheduler to attempt to change the problem. For example, the person responsible for taking reservations at a restaurant will negotiate with the clients to change their requested reservation time to one where he has a solution, thereby changing the problem.

Another way to alter the problem definition is to change the structure of the problem. In a paper by Aubin & Ferland there is discussion on whether or not it is more computationally cost effective to model a problem of generating a timetable and assigning

students to sections as two sub-problems or one large problem [14]. A question that comes to mind, but is not addressed in the paper is: Which problem definition makes for a better solution in the real world? Altering the priority of objectives can also change a problem definition by changing which objective comes first [22]. Modeling a problem with a different objective as its first priority may result in a different solution. In a paper by Wright, a timetable with a two-week cycle is simplified so that it can be represented by a one-week cycle that is repeated [25]. The simplification of the problem results in a different problem definition, yet the implications of this are not discussed. The problem is easier to model and solve as a one-week cycle, but we do not know if it works as well in practice as the two-week definition.

The stage of simplifying a problem is present because it is necessary in order to find solutions to almost every problem. Little is written, in the operations research domain, about what one simplification does in comparison to another and what implication each simplification has to fitting the solution into the real world. There are some papers that do discuss how to check for model quality. They look at which factors are important when creating a cost function and how important each constraint violation is in the real world [17, 25]. A paper by Carter describes the timetabling procedure at the University of Waterloo as well as the cost function [17]. It describes how the costs for rooms are calculated by taking into account factors such as distance, size, and equipment. Wright describes a method of searching for a timetable at a high school [25]. It includes the full calculations for the cost function in the appendix. Although cost functions can help guide the solution to be of good quality, what factors to include and how to include them can be very difficult. Often, in timetabling, as will be shown in Chapters 4 and 5, human judgment has to be used to make difficult trade-offs between several goals and simply looking at the results of a cost function will not accurately describe the quality of the schedule. Nonetheless, measures of quality are important because they can be used to compare different problem definitions.

There are some cases where a comparison of problem definitions is hinted at. Muller & Rudova test a timetable for a university using real data [4]. A problem definition is given along with several small changes that can be made. For example, the same definition can be used in one case where students are not moved between sections and in another case where they are moved. Also, in one definition the priority can be to satisfy faculty preferences while in another, the priority can be to satisfy student preferences. The paper compares the results for each of the cases listed above in terms of how long it took to solve the problem, if a feasible solution was reached, and to what degree soft constraints were violated. It does not look at how well solutions from each definition worked in the real world.

There are many papers that show a model being implemented in a real situation, such as an automated timetable being used in a school [11, 21, 24, 25, 26, 27]. More commonly, though, a solution is simply tested with real world data [1, 6, 10, 13, 14, 15, 18, 19, 22] or with made-up data that is meant to be a good representation of the real world [5, 12, 20]. These methods work to show that a given solution can work in the real world.

What is missing is showing why certain assumptions were made and how they affect the solution; showing how one problem definition creates a solution that works differently

than that of another problem definition. In other words, there should be evaluation criteria that are created. These evaluation criteria can then dictate what information is necessary to include in a problem definition and what information is not. There should be a step added to the process of solving real world problems in OR, which occurs after analyzing the problem, but before creating a model. One should sit down with the clients and clarify what is a solution, what form of solution is desired, and what, if any, tests can be done to ensure that the solution will work in the real world. In Chapter 5, we make a step towards this goal by creating evaluation criteria for a real world problem.

3.6 Bond Graphs

This section is different from the previous sections because it focuses on a specific technique for system modeling. Bond graphs are a modeling tool in that they are a flexible way to model a system that includes some sort of interaction between components. Systems interact by storing, transporting, or dissipating energy among subsystems. Bond graphs apply to the problem definition stage because they can represent a domain in such a way that a program or solver can use to find a solution. Below is a picture of a simple single degree of freedom mass-spring-damper system and its corresponding bond graph taken from [61]:

Figure 3.2a. A mass-spring damper system

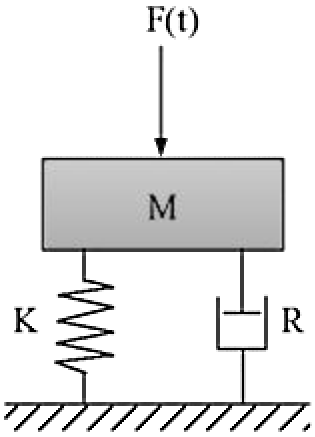
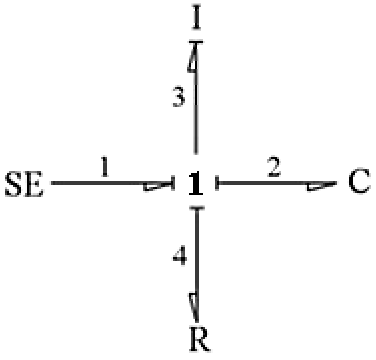


Figure 3.2b. The bond graph representing the system in figure 3.1a.



In this example, a one port resistor (R), capacitor (C), and inductor (I), along with an effort source (SE) describe the system. These elements are connected by a 1-junction, which means that there is equality of flows and the effort sums to zero.

Bond graphs can be used for many levels of abstraction when representing a problem, so they are adaptable to different problem definitions. Bond graphs are used in model-based diagnosis because they can represent a hybrid system [35, 36]. In papers by Narasimhan, et al.[35] and Karsai, et al. [36] the advantages of being able to represent a hybrid system is explained. A hybrid system is usually represented either as a discrete or continuous one, depending on the problem. Either way, it is a simplification that often results in inaccurate models of the domain. Discrete changes are not handled well by continuous models and discrete models can cause loss of information that is necessary for fault isolation and control. An enhanced form of bond graph, with switched junctions to represent the discrete change of modes can be used to represent a hybrid system.

In a paper by Bos & Tiernego bond graphs are used to represent an industrial manipulator (figure 3.3a) with three rigid bodies. We see word bond graphs as the highest level of abstraction, where components and their connections are almost everything that is represented (figure 3.3b). The typical bond graph comes next and it expands on each of the connections shown in the word bond graph by including the type of energy connection between components (figure 3.3c). Finally, there are causal bond graphs, which expand on the connections shown in the typical bond graph and include detailed information on how components interact as well as direction of energy flow [29]. This lowest level of abstraction can be used for simulation.

Figure 3.3a. An industrial manipulator with three rigid bodies [29]

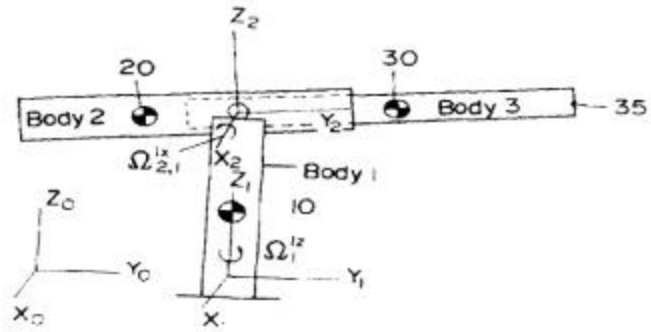


Figure 3.3b. A word bond graph of the industrial manipulator [29]

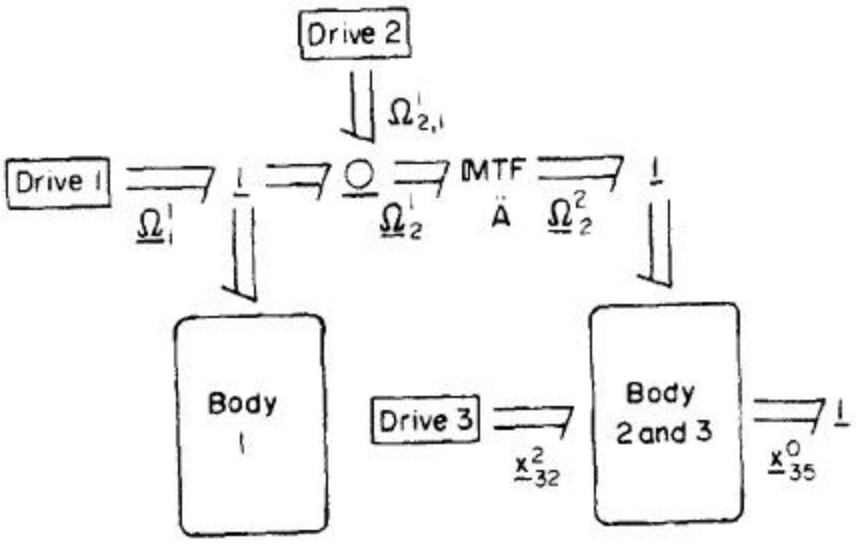
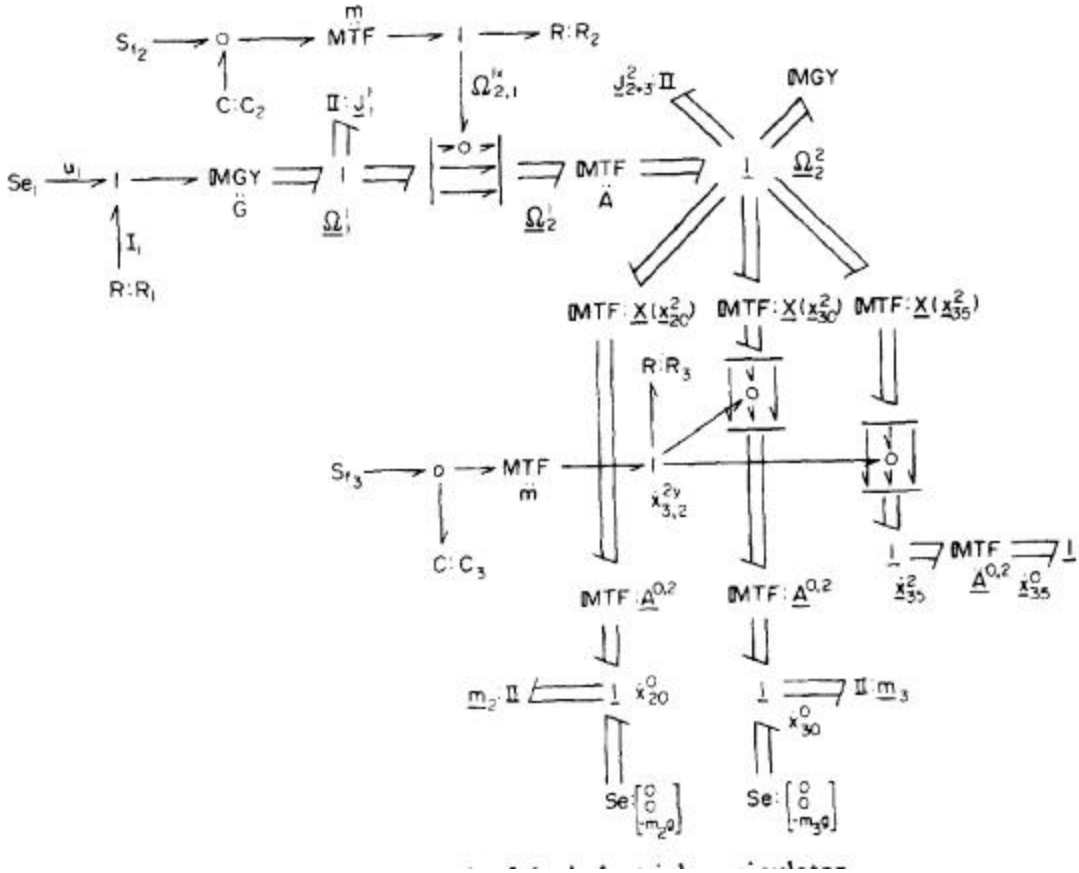


Figure 3.3c. A bond graph of the industrial manipulator [29]



It is possible to build a bond graph through a heuristic search such as the genetic algorithm [44]. Rosenberg, et al. show one way to find the best bond graph representation [44]. They start with a high level of abstraction and build to lower levels with more detail as they explore the space of possible bond graph representations. They also shown how, at the same level of detail, there can be different problem definitions and hence a different bond graphs, such as a bushy structure as opposed to a long chainlike structure. Different structures come with possible advantages. For example, a bushy structure may make it easier to exploit the subsystems because one may be able to make use of the chunks. The specific uses of a given structure are mentioned in the paper as an area for future work.

Bond graph models can be used at many abstraction levels and therefore for many different problem definitions. Here, as before, there is no use of evaluation criteria and the validation that exists is showing that a model works through the use of an example [29, 35, 36, 44]. Perhaps a technique, similar to bond graphs in that they can represent information at several levels of abstraction, would be useful to have for optimization problems.

3.7 Conclusion

Real-world design typically begins with initial requirements that are vague and incomplete and that must be transformed into specified ones where a solution can be found to satisfy them [46]. Each problem can be defined as a set of requirements that refer to functional and other elements in a domain [46]. In all fields, where a definition of a domain is required, whether for a problem in optimization, diagnosis, software design, or enterprise modeling, assumptions are made for many reasons. These assumptions, to enable exploration of a space of possible solutions in a reasonable amount of time, are what define the problem in a particular way and may result in a solution that is quite different than one that would have resulted had the problem been defined differently.

By looking at the literature, it has been shown that software engineering and enterprise modeling are two areas that actively research the problem definition stage as well as validating different problem definitions in the real world. Model-based diagnosis has a problem definition stage, although it is not discussed as openly as in software engineering and enterprise modeling. In the literature on operations research, although a problem definition stage is not directly studied, we know that such a stage is necessary when it comes to creating a model to solve a real-world problem. For example, the literature on automated timetabling takes a given timetabling problem and reduces it to a mathematical definition, which can then be solved. In reality, the timetabling process is long and complicated, as will be shown in Chapter 4. Authors, therefore, had to create a problem definition in order to solve the timetabling problems, but the process of creating those definitions as well as its implications on a solution are not discussed. It seems that research on this topic of defining the problem and seeing how different definitions can have different results in the real world would be beneficial to optimization and the field of operations research as a whole.

It seems like it would be helpful, when tackling a real world optimization problem, to know what sorts of simplifying assumptions are useful given the nature of the problem, as in model-based diagnosis. It would be nice to have processes in place to follow in order to

create problem definitions that lead to solvable models with solutions that work in the real world, as in software engineering. It would also be nice to have a problem definition that could be used to analyze what would happen to the problem if certain constraints were relaxed and what effect it would have on the quality of the solution, as in enterprise modeling. It also seems that a technique, similar to bond graphs in that they can represent information at several levels of abstraction, would be useful to have for optimization problems.

All of this would be nice, but currently does not exist. One way to start would be to add a step into the OR problem solving process. The step would be after analyzing the problem, formally deciding what a solution would look like and what tests can be done to be sure that the solution will be useful in the real world. This thesis makes contributions towards these goals. Firstly, in Chapter 4, a real problem is taken and its solution process is studied in detail to show how real world problems are much more complicated than what typically appears in a mathematical model. Secondly, the problem definition stage for that real world problem is addressed by creating evaluation criteria, which we will see in Chapter 5. Finally, in Chapter 7, a mathematical model is developed for a problem similar to the real world problem that is studied.

Chapter 4

Timetabling at the Faculty of Applied Science and Engineering, University of Toronto

4.1 Introduction

As with many real life problems, the university course timetabling problem can be messy and complicated. Solving the university course timetabling problem involves many people communicating to try to achieve a timetable that meets a set of requirements and goals. As explained in Chapter 3, the literature on automated timetabling often takes a given timetabling problem and reduces it to a mathematical definition, which can then be solved. In reality, there is a lot more to a real world timetabling problem than what is represented in such a definition. The timetabling process is long and consists of many stages before that of actually placing courses into timeslots. The first stage of solving a problem in OR involves a detailed study of the system, identifying specific problems, system constraints, and objective functions.

This chapter looks, in detail, at the timetabling problem at the faculty of applied science and engineering at the University of Toronto (APSC). The process described is the one that took place in order to create the timetable for the 2006-2007 school year. This process shows how real world problems are actually much more complicated than what appears in a mathematical model. As well, a detailed analysis of a given problem is a step towards creating a problem definition. It allows one to identify all of the process issues, constraints, restrictions, and goals, thereby providing a base of information that may be included in a problem definition.

The undergraduate program at APSC consists of four years of study. There are 4000 students, over 1200 of which are first years. There are seven departments and nine degree programs totaling 79 POSts[1]. There are 219 faculty members, 12 buildings, and 80 lab rooms that are managed internally. The faculty uses a software scheduling package that is part of the Syllabus Plus suite of scheduling products. In particular the software Course Planner (CP) is used to schedule, identify issues, and support decisions. CP is a software package that uses several heuristics when scheduling. 75% of timetables are delivered to the individual student conflict-free, based on program structure. In the following sections, we

[1] POSt stands for "Program Of Study". It refers to a student group studying in the same program; students in the same department, same year, and same option. For example, fourth year, engineering science, manufacturing option is a POSt, as is first year civil engineering.

describe the goals that the timetable tries to achieve, the constraints involved, and the strategy, the process, used when creating the timetable. We then outline some problematic areas existing in the current process and highlight the areas where IT could be helpful. Identifying areas where IT could be helpful should make the problem definition problem easier.

4.2 Goals

The overall goal of the timetable is to provide students with a schedule that is not only conflict-free, but is of good quality as well. The following is a list of goals in terms of the students, the faculty, and the use of resources. There are both timetable goals and process goals. Timetabling goals refer to aspects that should be present in the resultant timetable and process goals refer to the process of creating the timetable.

4.2.1 Timetable Goals

Students:

- Conflict-free schedules for years one to three and fourth year core courses
    - Next best is to minimize conflict and limit it to tutorials or ends of labs
    - For fourth year, try to minimize conflicts among the most popular courses
- Deliver required courses to the students
- Try to provide each student with a 9-5 schedule with a lunch break of one hour between eleven and one
    - next best is 9-6 and then 9-7
    - lunch break between 11 and 2
- Minimize gaps in a given day
    - Existing gaps should be meaningful (i.e. not too long)
- There should be some study time
- All Programs of Study (POSts) should have equal access to resources (e.g. labs)

Faculty:

- Conflict-free
- Meet staff criteria, such as staff availability and course delivery requirements
- Ideally, professors should have one day for research that is free from teaching

Resources:

- Better utilize labs and rooms

- Room usage: A minimum goal is to fill rooms 50% of the time
    - To fill a room means to have a scheduled event taking place in the room

4.2.2 Process Goals

- To serve the client - the students and the departments - through the counselors and the faculty

- Try to get at much course data as possible confirmed, as early as possible
    - The curriculum committee should meet earlier than they do so that course delivery can be known early on and scheduled. Ideally, the curriculum committee would be working on a schedule where they are a full academic cycle ahead. For example, curriculum for the 08/09 school year would be mostly determined in 06/07.

- Minimize the transfer of information between the counselors, the representatives from the departments who work together with the director of scheduling to create the timetable

- Give all departments equal time with the director of scheduling to work on creating their schedules

- Improve communication between faculty and department and director of scheduling

Quality is a subjective measure and in such a large organization, where there are many courses that are taken by more than one student group, constraints can make it very difficult to create a satisfactory timetable that is of good quality for all the student groups. For example, in the third and fourth year of the electrical and computer engineering program, there is a flexible curriculum. This means that students get to choose all their courses from a fairly large list of options. This makes for over a hundred possible combinations of courses. When scheduling, it can be impossible to make all the courses in the list conflict-free.

Providing schedules of good quality is the secondary objective when scheduling, after obeying all of the necessary constraints such as faculty and room availability. Assessing what makes a schedule one of good quality is therefore an important step of creating a timetable. The process we undertook to assess what makes a schedule good will be described in detail in Chapter 5.

4.3 Constraints

In the timetabling domain, there are two types of constraints. Hard constraints are constraints that cannot be violated because if they were, the schedule would be infeasible. Soft constraints, otherwise known as preferences, are there to make the timetable as good as possible. Fewer soft constraint violations mean that the schedule is better. In addition, in the University of Toronto example, there are certain situations that arise, due to the nature of the program, that seriously constrain the schedule. Although these are constraints in a slightly different meaning, they will be referred to as constraining factors and they will be listed in this section as well.

4.3.1 Hard Constraints

The hard constraints are the constraints that cannot be violated. Of course, there are exceptions to every rule and if violating a hard constraint will make the quality of the schedule much better, perhaps it will be done.[2] If a hard constraint is to be violated it must be sanctioned by the director of scheduling, the counselor of the affected department, the head of the affected department, and any affected faculty members. An example of when this might be done is the case where there are only supposed to be two sections of a tutorial, but for one group of students it makes their schedule bad if they go to either of those sections. It will be discussed and analyzed to see if it is worth the money and if there are enough resources to add a third section for that group of students. Another case where hard constraints may be violated is the case of the flexible curriculum of fourth year. There is a hard constraint to make a conflict-free schedule for all students, but when there are so many course options, it may be impossible to have every combination be conflict-free, especially when the courses are scheduled before the students' final choices are known. In this case, conflicts are minimized and any conflict is sanctioned by the department.

The following is a list of hard constraints in no particular order:

- No conflicts[3] for students
- No conflicts for staff
- No double booking of rooms
- Faculty availability – All professors should be available when they need to teach
- No courses on the weekend
- All classes must fit into the capacity of the assigned room
- Delivery requirements must be met
    - Number of lectures, labs, and tutorials
- Special room needs such as electronic classrooms

[2] This requirement of human judgment to make tradeoffs between satisfying a constraint and ensuring good quality is one of the things that makes the definition of a mathematical timetabling model so difficult.
[3] A conflict is when a student or faculty member is scheduled, in their timetable, to be in more than one event at the same time.

- Pre set rooms (and lab rooms)
- Same-timed activities – Same-timed activities refer to events that must be scheduled at the same time. One example is a tutorial of a particular course with several sections, where all the sections should have their tutorial at the same time. Another example is two courses for the same student group which both have three hour labs on alternating weeks. Those labs should be at the same time, just on different weeks.

4.3.2 Soft Constraints

Some of the soft constraints are built into the software as preferences and the user can choose what priority to give each constraint and whether or not to include it at all. The remainder of the soft constraints/preferences are kept in mind by the director of scheduling when creating the timetables. The following is the list of preferences. Preferences existing in the software will be denoted by (S).

- Avoid overtime for staff and locations. Overtime is requiring staff to work, or rooms to be in use, later than the standard end of the day or earlier than the standard start of the day.
- Avoid conflicts that are considered by the department to be acceptable. For example, a department may allow for there to be conflicts between tutorials, but they would still want them to be avoided, if possible.
- Try to put activities in rooms that are as close in size as possible to the number of students attending the activity (S).
- Load balancing (S)
    - Try to use all available resources evenly.
- Preferred starts – Try to assign activities to their preferred times (S).
- Preferred usage – Try to assign activities to their preferred rooms (S).
- Primary suitability – The most suitable or primary resource should be used. For example, a lecture should be scheduled in a lecture room and a tutorial, in a tutorial room (S).
- Usage spread
    - When there is more than one section of a course, the sections should be assigned as even as possible. Ideally there should be the same number of students in every section.

4.3.3 Constraining Factors

There are six issues that seriously constrain the schedule. Courses that are effected by these constraints must be scheduled in specific spots, thereby requiring that the timetable is built around them.

1. Over-taxed locations – There are certain rooms, in particular lab rooms, that are almost always in use. They need to be used by many groups of students for many courses and therefore constrain the timetable a lot. In 2006-2007, as well, all of the

larger lecture rooms are over-taxed because of the double cohort[4] being in the upper years. Usually, the upper years do not require larger rooms, but this year, because of the double cohort, they do.

2. Shared courses – By nature of the program, there are many courses that are shared by several groups of students. It may be an elective for one group and core for another and/or it may be shared across departments and years. This causes all the timetables that are involved to be constrained by each other.

3. Common sense rules – There are certain common sense rules that the scheduling department tries to follow while scheduling. One such rule is not to schedule tutorials in the morning, before lectures. The rule exists because it is known that few students will attend the tutorial.

4. External faculty – Faculty availability can be very constraining. This is especially true for external faculty who are constrained by courses that they teach outside of the faculty and other external responsibilities. External faculty refers to faculty members that are based in other departments at the university as well as faculty members whose main place of work is not the university.

5. Team teaching – Team teaching is a new phenomenon where more than one teacher teaches the same course. This causes the course to be constrained by all the teachers' schedules and all the teachers' schedules to be constrained by whatever else is constraining that course.

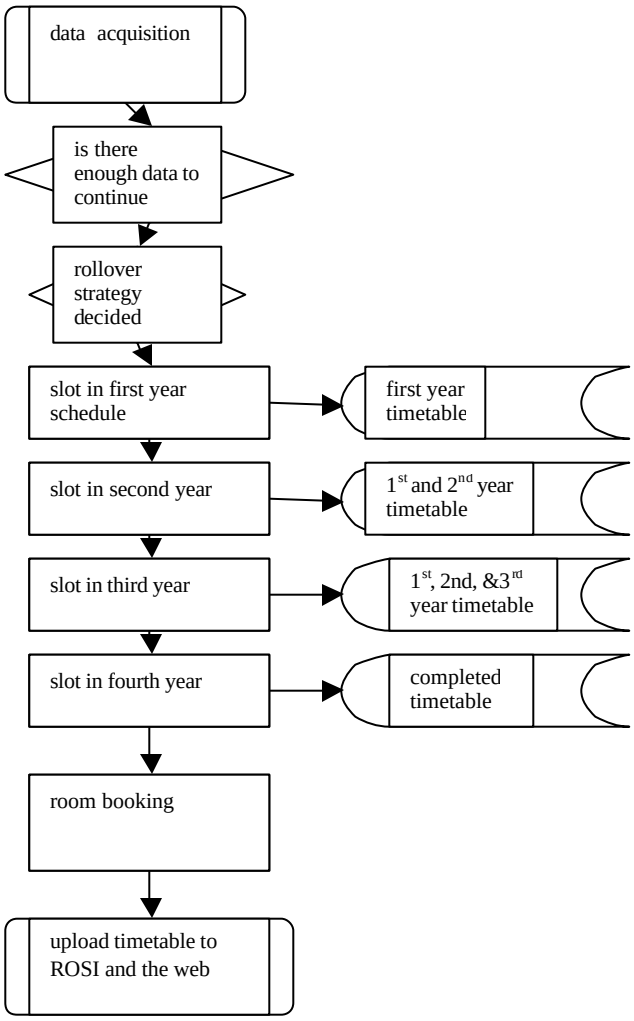
6. Specific courses – There are specific courses that require a lot of resources and therefore highly constrain the schedule. This applies, mostly, to first year. The first year program has one core course that all 1200 students must take at the same time. It uses up 30 rooms and many faculty members. No other first year courses can be scheduled at that time.

4.4 Strategy

There is no written protocol that is followed when creating the timetable. This is because every year is unique and different than the previous one. There is, however, a general strategy that is used. The basic steps that make up the scheduling process are the same each year. First is data acquisition. Second is deciding on the rollover strategy. The rollover strategy is deciding what part of the previous year's schedule is kept and rolled over for the following year. After the rollover strategy is determined, each year's timetable is scheduled, one at a time, starting with the first year program and finishing off with the fourth year. The following is a workflow diagram of the scheduling process:

[4] In Ontario, the class that started high school in 2000 was the first class to graduate without Ontario Academic Credit (OAC), a previously required fifth year of high school. This resulted in two cohorts of students graduating high school, and therefore starting university, at the same time, in 2004. This group of students is referred to as the double cohort.

Figure 4.1. A workflow diagram of the scheduling process.



The scheduling process really begins before the data acquisition stage, with the creation of the curriculum and calendar. However, this part of the process is not discussed here. In the following sections, each step in the above scheduling process will be looked at in more detail.

4.4.1 Data Acquisition

Accurate data is essential when creating a timetable. Without it, constraints cannot be formulated properly and conflicts will undoubtedly arise. Data is necessary for identifying the potential problem areas in a schedule so they can be kept in mind while scheduling. At APSC there is a lot of data that must be collected before the scheduling process can begin. The following is a list of data that must be collected:

- Course information
  - What is and is not being offered
  - What is different from last year

- Program/POSt relationships
  - Which courses are core or elective
  - Which courses are shared
  - How are they shared

- Staffing info
  - Who is teaching
  - What is their availability
  - What are their other responsibilities/availabilities
  - Which courses are not yet staffed, so late changes can be anticipated

It is important to know which faculty members are teaching which courses. It is also important to know which courses have faculty assignments that are still to be announced (TBA), so that those courses can be scheduled with the knowledge that their assignments may have to change later on in the process.

- Resource requirements
  - Special rooms
  - Smart/electronic classrooms
  - Labs
  - Large tutorials

- Course choices
  - Possibly from a pre-registration process (Course and Option Selection (COS) is often used for third year students)

- Activities
  - Sections per course
  - Planned sizes
  - Enrollment limits
  - Labs and tutorials
  - Delivery requirements such as same time vs. sequential scheduling

- Resource data
  - New or refurbished space
  - Space lost
  - Updated/lost equipment

- Student data
  - Returning students through COS
  - PREP for new students on the basis of previous years' selections and curriculum changes. PREP is the term used to refer to estimated data. In this

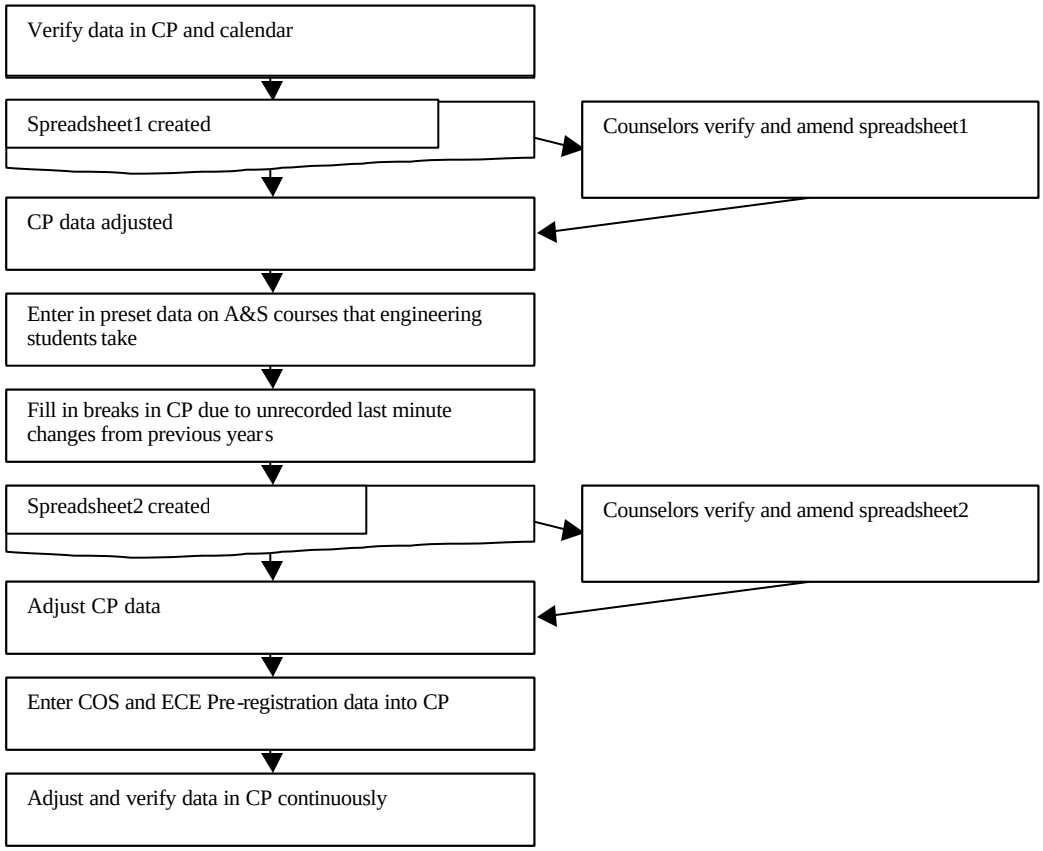
case, PREP contains estimated planned sizes for a lot of the courses, as well as the number of first year students. The course choices are also estimated using the previous year's data.

- Allowances for retakes, etc.

- Anticipated changes and trends in the student population

The following is a workflow diagram of the data acquisition process:

Figure 4.2. Workflow diagram of the data acquisition process.



CP refers to Course Planner, the software used by APSC and A&S refers to the Faculty of Arts and Science. Spreadsheet1 contains data on staffing assignments, staffing constraints, term offerings, and planned sizes. Spreadsheet2 contains data on course delivery, class patterns, number of sections, and best room suitabilities. COS and ECE (Electrical and Computer Engineering) pre-registration are online forms that the students fill out in early March. They indicate which courses and/or options they are planning to register in for the coming year. For 2006-2007, APSC received 80% response back for the choice of options of students entering third year and 50% of course selections for students entering fourth year.

Verifying the CP and calendar data is a two-person process. It is a crosscheck of the curriculum change form, which contains all changes to the curriculum for the upcoming

year, the CP database, with the data from the previous year, and the calendar for the upcoming year. This is a manual process.

The process of collecting data through the spreadsheets takes a long time: a couple of months. It involves the counselors verifying and filling out what is missing from the data by checking with the professors teaching each course. This is a manual process, which means there can be errors and the data must be checked over carefully before it is put into CP.

Curriculum changes come in throughout the process and when they do, the data in CP must be adjusted and verified. Also, if scheduling has already begun, the schedule may need to be adjusted as well. When changes come in after the schedule is already posted, and there are always changes, three systems must be updated; CP, ROSI, and the room reservation system (RRS). ROSI is the student web service. Along with many other things, it stores students' schedules and it is the tool through which students enroll in their courses.

4.4.2 Rollover Strategy

The rollover strategy determines which courses are kept from the previous year. Rolling over some courses makes the scheduling task a little easier by providing a starting point. It provides something other than constraints to work from.

Curriculum changes affect the rollover strategy greatly. If the courses offered are different than the previous years, then the schedule cannot be kept. All curriculum changes need to be evaluated for the impact on the rest of the schedule. A feasibility study should be done to see if existing resources are sufficient to accommodate the changes. There is often not enough time to evaluate every change properly because the curriculum committee works on a very tight timeline. As a result, many changes go through when they probably shouldn't and it is only once scheduling has begun that it becomes clear that the changes require too many resources. These changes often have a negative impact on the rest of the timetable. Along with curriculum changes, staffing changes can affect the rollover strategy as well. Each faculty member comes with his or her own availability. A timeslot that worked well for one faculty member may not work at all for another. Analysis must be done to see if a curriculum or staffing change affects other courses, to the extent that they too cannot be rolled over.

When deciding what to rollover, the scheduling office analyses the timetable from the previous year. They ask themselves: How was the quality? Were we satisfied? The determination of whether or not a timetable was of good quality is currently done very informally. In an open discussion, each department has input as to how they would like their schedule dealt with. There are no concrete measures currently used to evaluate the quality of a schedule. The scheduling office prefers not to completely erase the previous year's schedule. If something is kept as an initial template, the scheduling office can then make incremental changes as needed. Of course, this can only be done if there are no major curriculum changes and if the schedule from the previous year was satisfactory.

It is preferred to have some rolled over courses because it decreases the amount of time required to create a schedule. For example, the fourth year curriculum is very flexible and changes have been made in the last few years by almost all the departments. Because of this, it is impossible to keep the same timetable from year to year. It is quite likely that scheduling from scratch would result in a better timetable because of these large changes and due to the fact that the existing student choice data is out of date and hence, practically useless. The scheduling office still rolls over some basic, core courses, which are the same as the previous year. It is too labor intensive and the timelines from faculty committees do not provide adequate response time to start from scratch.

4.4.3 Slot in First Year

There are certain things that are common to the scheduling of each year. For each year, the director of scheduling together with the counselors look at what changes affect that particular year. The scheduling for each year is done separately for each department. The director of scheduling meets with each counselor individually and they schedule that department's courses. There are many shared courses, but each course is owned by one department and that department can choose to place their course where they would like, assuming there is room for it, there are no faculty conflicts, and that it does not prohibit another engineering student set, that is supposed to have access to that course, from accessing the course. It is the director of scheduling's responsibility to ensure that all of the departments are being treated fairly as far as access to rooms and times.

Before the scheduling department can begin scheduling they must check that all the courses are in the database in the system and check the delivery patterns. Delivery patterns refer to the number and length of course activities. For example one course may have a delivery pattern of three one-hour long lectures and a two-hour lab. Another course may have a delivery pattern of a single one-hour lecture, a two-hour lecture, and a ninety-minute tutorial.

For each step in the scheduling process, there are many iterations. At each step, the settings of preferences and constraints are redone and reworked many times to see if compromises can be made in order to get a better quality schedule for the students. There are many meetings with each departmental counselor. Often, at a meeting the scheduling director along with the counselor will come up with a solution to a scheduling problem, but the change cannot be made until the counselor checks if the faculty member assigned to the course is willing to accept the change.

Once the timetable for a given year and department is completed, the scheduling department checks if everything was scheduled. They also check how it got scheduled; are there classes in the evenings and what is the spread like. If they are not happy with what the schedule looks like, they will often reschedule by unscheduling and rescheduling classes. First this is done automatically, using the software. Automatically means letting the software use the embedded heuristics to schedule any selected group of classes. The results are then evaluated, preferences are tweaked and rescheduling can occur. Later, manual adjustments are made. This step is invoked if most of the courses were scheduled well except for one or

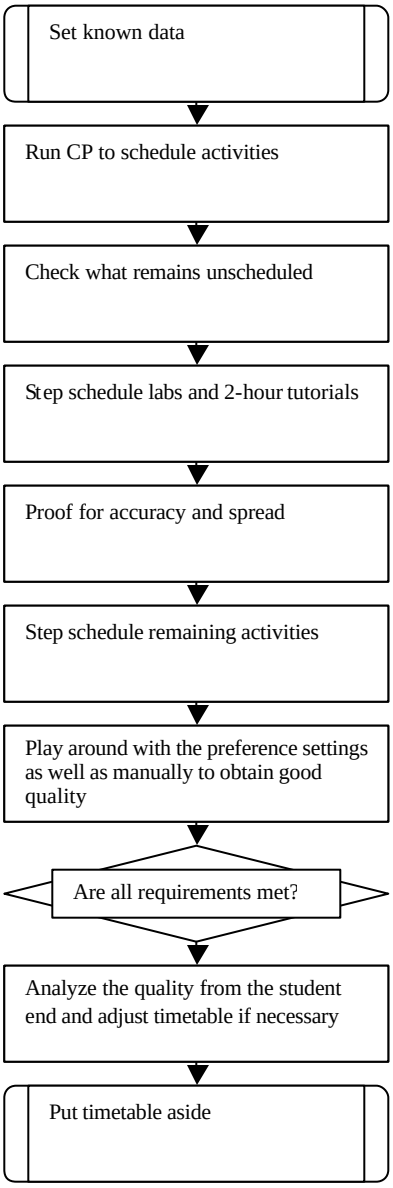
two. Those could be manually moved in the schedule, rather than auto-scheduling everything. While scheduling, CP will indicate whether an activity can be put in a given slot and if not, why; when it conflicts and with what student set or if any other hard constraints are being violated. Sometimes these constraints can be overridden.

Once all departments have been scheduled for a given year, the schedule for that year is then analyzed from the student perspective and signed-off. Changes are often made to a signed off schedule if it causes an upper year schedule to be of bad quality, if changes are made to originally poor data, or as a response to unanticipated factors.

At the end of scheduling a given year, the existing timetables are signed off temporarily. At the end of the process, when all years are signed off, departmental counselors ensure that quality criteria are met, everything is scheduled, and that program constraints are accommodated.

At the APSC, first year is scheduled first for several reasons. (1) First year is given priority because the scheduling department wants to ensure the best quality possible for the new students as a sort of welcome to the faculty. Because the scheduling department wants the first years to have the best quality, they don't want to run the risk of not scheduling the first years first and leaving them unprotected, having the first year schedule be impacted negatively. (2) First year uses over 25% of the resources. (3) First year has a lot of large classes and there are a limited number of adequately sized rooms. In fact, there are half as many rooms that can accommodate first year sized classes than can accommodate third and fourth year classes. Below is a workflow diagram of slotting in the first year timetable:

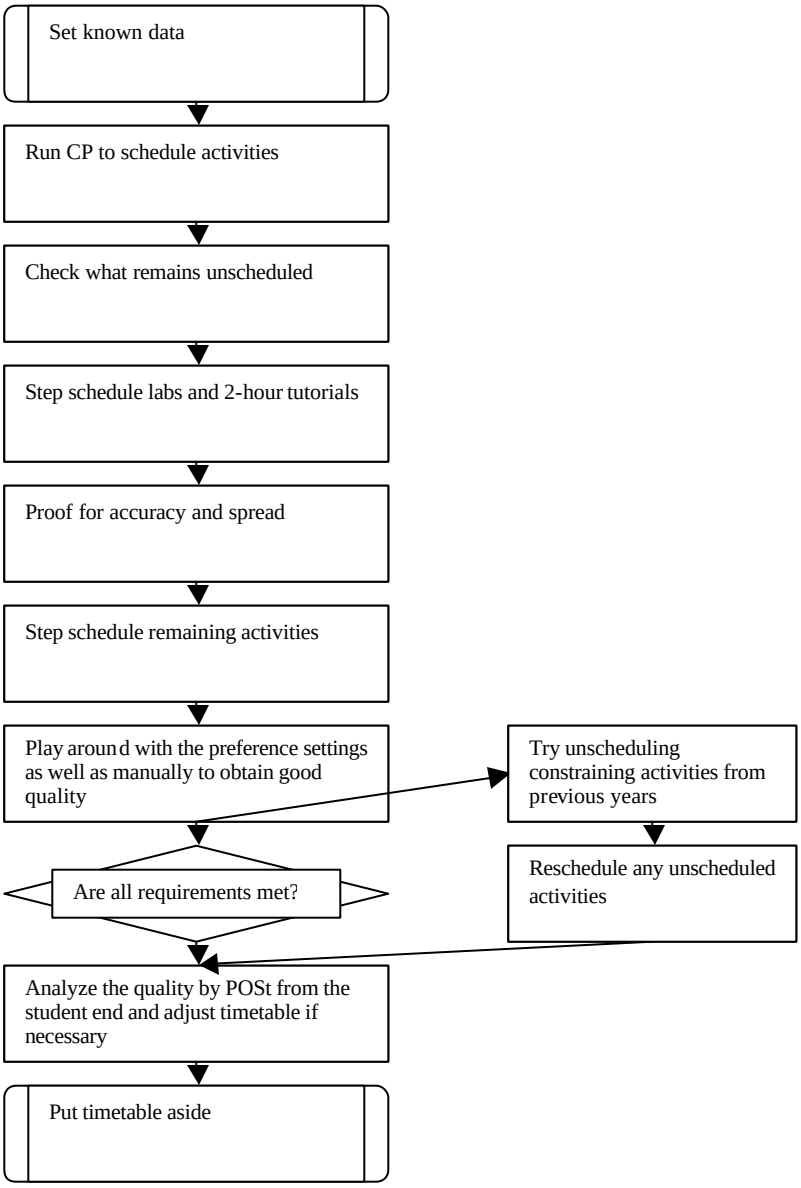
Figure 4.3. A workflow diagram of slotting in the first year timetable



4.4.4 Slot in Second Year

Below is the workflow diagram for scheduling courses of second, third, and fourth year. This process is repeated separately for each department and year of study.

Figure 4.4. A workflow diagram of slotting in an upper year timetable.



It should be noted that certain courses that were supposed to be rolled over, might have been affected by the first year schedule. This is true for the third and fourth year schedules as well. Third year may have been effected by first and second year and fourth year may have been affected by all three previous year. Often, while scheduling an upper year, changes are made to the earlier year's schedules.

4.4.5 Slot in Third Year

For some departments, third and fourth year are scheduled together. For the rest, third year is done before fourth year. In 2006-2007, many third and fourth year student sets resulted in too many combinations, so student sets could not be considered. A student set is students that share the same timetable. In the case of third and fourth year, a student set is students who chose the exact same courses. When there are so many possible combinations of courses, and equally many student sets, it is impossible to create a conflict-free schedule for every student set. For the 2006-2007 school year, there were 540 possible combinations of courses that a student could choose in the mechanical engineering program and there were nine possible combinations in the mineral program, a program with only twelve students. Therefore, the scheduling office did not take into account the student sets for many third and fourth year programs. Instead, they tried to minimize conflicts between courses that seemed to go together. In 2007-2008, they plan to group courses according to streams. A stream refers to an area of specialization within a program. For example, the industrial engineering courses can be grouped into three streams; operations research, human factors, and information engineering. By grouping courses into streams, courses that apply to a particular stream, can be conflict-free.

4.4.6 Slot in Fourth Year

Fourth year is done last for several reasons. (1) There is the most flexibility in the fourth year schedule. The number of options results in a large number of possible combinations. It is impossible for all the combinations to have conflict-free schedules. Some of the combinations have only one or two students in them. Because of this, difficult trade-offs have to be made. Since many conflicts would be sanctioned for the fourth year program anyway, it makes sense to leave it to last so that it not constrain the other schedules. (2) The student choice data that exists is out of date, making the scheduling process even more difficult. As it gets later in the process, more and more current student choice data becomes available. (3) There are smaller lecture sizes and therefore the most flexibility in the number of rooms that can be used.

4.4.7 Room Booking

As a first step, before rooms can be assigned, each room has to be given a suitability designation; whether it is a departmental room, a lab room, a tutorial room, or a lecture room. The features that each room possesses must also be recorded.

Assigning rooms is a process that takes place both during and after the assignment of times described above. During that process, lab rooms are assigned. This is necessary

because there are a finite number of lab rooms for the entire faculty and there are no rooms that can be used in their place in the rest of the university. Also, activities with preset rooms are entered during the assignment of times. CP assigns rooms during this process as well, although during the time assignment exercise, all conflicts regarding the rooms assigned by CP are ignored. The rest of the rooms are assigned after times are assigned. If there are no rooms available at the assigned time in one of the engineering buildings, a room is found elsewhere in the university. This, of course, is not preferred.

Once the times are set, the list of conflicts regarding rooms is examined. Some room conflicts are sanctioned. For example, this may be the case if one course has two course codes. There are some courses that are taken by both graduate and undergraduate students. The course is the same, but the course code is different for the graduate students than for the undergraduate students. Therefore, both of the courses must be in the same room. Also, some lectures can be given in tutorial rooms and vice versa. At that point, the rooming for any activities that are still left without a room is done manually.

Just like the rest of the timetabling process, there are often unexpected challenges in room assignment. In 2006-2007, there were surprise room conflicts because the office of space management (OSM) allowed Arts and Science to use some of APSC's larger spaces, which the faculty needed, before APSC was done scheduling their rooms. The scheduling department then had to find new spaces for those classes. There were also several instances of rooms not being adequately maintained. For example, what was supposed to be an electronic lecture room may not have had adequate equipment, etc.

4.4.8 Upload Timetable to ROSI and the Web

Once the schedule is made it is uploaded to ROSI and the faculty website so that students can view it. The timetable needs to be on ROSI by the time students are able to register for their courses. This mainly applies to students in third and fourth year, who choose and enroll themselves in all their courses. It may also apply to a lower year student who has an Arts and Science (A&S) elective to enroll in. The scheduling office attempts to have the schedule online before the date when students enroll, so that students have a chance to look over the courses and get an idea for which combination of courses will work for them along with any A&S courses they are considering.

Uploading the timetable is not completely automated. CP and ROSI are not, by nature, compatible. Therefore, formats and such may need to be tweaked in order for the upload to go smoothly. The scheduling department does the tweaking manually. As well, the IT department of the registrar, does tweaking electronically.

After the timetable goes up, there are almost always more changes that need to be made. Mostly this is a result of poor communication of requirements on the part of the faculty. At the time of timetable creation, the data that was verified by the counselors is assumed to be correct, but once it is online, there are often complaints such as those from professors who realize that they want more or different rooms. Also, there are some courses

that do not have assigned staff at the time of the upload and once that staff is assigned, the assigned professor may have time constraints or other requirements that force a change in the schedule.

Other conflicts can result from planned sizes. At the time of upload, the scheduling department only has an estimate of course sizes. There are students who transfer between programs or options as well as students who did not fill in their COS form and therefore it was unknown which courses or options they were taking. Once the real course sizes are known, there may be rooming issues. Planned sizes are especially problematic for first year, since it is unknown who will accept or reject their offers of admissions at the time of the upload.

Issues such as those described above continue to occur until the last day to drop and add fall courses, in mid-September.

4.5 Problems in the Process

There are many areas of the process where there is a need for improvement. These problems range from technical issues such as there being too much data being entered manually, to communication issues, to political issues within the faculty. Some can benefit from an IT solution, and some cannot.

4.5.1 IT Solutions

There are several instances during the process where automation would be helpful. The obvious one is that of the creation of the timetable. Software is currently used, but that software requires a lot of interaction and in a way it is merely a database that holds data and notifies the user when conflicts exist, while the timetable is actually created manually. The CP software can schedule automatically, but from experience, the created schedules are often quite far from ideal. CP often has a lot of difficulty finding a timetable that doesn't violate constraints. CP does, after all, use heuristics to make its scheduling decisions, which may not be the best option. Using mathematical programming, a model could be created to solve the APSC timetabling problem. Such a model might not require as much interaction. It would take the data and create a timetable, which could then be modified by the user.

There are other areas, earlier in the APSC process that could also benefit from automation. The director of scheduling has identified these areas as well as the proposed solution. One such area is the step of verifying the CP and calendar data. This is currently a manual, two-person process involving cross-checking data from three different sources. If these data were connected electronically, a lot of time would be saved. Also, during the data acquisition phase, data is collected through spreadsheets. The process involves passing back and forth information that gets changed slightly each time. This process is currently done manually, creating many opportunities for miscommunication and errors. Errors include filling out forms incorrectly as well as missing information. A third area where automation would be helpful is that of updating the CP data after the spreadsheets are completed. This is done manually.

The proposed solution, from the director of scheduling, is to make the process of verifying, collecting, and updating data electronic. A database could be created from which the calendar data could be uploaded electronically to CP. Also, data collection could be done through online forms, where there could be input restrictions so that the counselors would not be allowed to fill out the forms incorrectly and blank slots would not be permitted. The data from these forms could then be uploaded electronically into CP. Such a solution would save a lot of time as well as prevent many errors.

Another area where an IT solution would be useful is that of the disconnect between the systems used for the schedule. When a change is made to the schedule, three systems must be updated: CP, ROSI, and the Room Reservation System (RRS). Often, there are different people updating the different systems and if it is not done simultaneously, someone may work on one of the systems assuming it is up to date when it is not. This can cause problems. It would be useful to connect the systems so that when one is updated, so are the others.

4.5.2 Non-IT Solutions

There are two reasons why an IT solution may not be possible: there is no IT solution that applies to the specific problem, or the IT solution that applies is not feasible.

The biggest issue existing in the current timetabling process is that of communication during the data acquisition phase. During this phase, the counselors are supposed to get all the requirements from the faculty in regards to their schedule preferences and necessities. Faculty are supposed to supply their departments with the delivery of the courses they will be teaching. Delivery refers to the number of sections the course should have and the number and length of all meetings of the course. Faculty members are also supposed to supply their rooming requirements. It is very common in the current process that faculty members do not supply much data during the data acquisition phase. In such cases, it is assumed that there are no strict constraints and that the delivery is the same as what is written in the calendar. It is also very common for such faculty members to come to the scheduling office with demands or complaints once the schedule is completed and uploaded. These demands range from wanting different rooms to wanting to change a one-hour lab to be a three-hour lab.

Although it may be possible to have an IT solution where faculty members could enter their data online, instead of going through the counselor, it is likely infeasible to expect "buy in" from all the faculty members. A more realistic solution would be to develop a written policy that includes a date by which the departments must have all their teaching assignments done, a date by which the faculty members must submit their scheduling data, and what data must be included. The scheduling office would then be required to approve any deviations from the faculty members' requests and there would be no changes made once the schedule is uploaded. A similar policy would be useful in regards to the development of curriculum. There should be no changes to curriculum made past a certain date. Implementing such a strict set of rules would not be a simple task. Ideally, the

curriculum committee would be a year ahead of where they are now. Adjusting to that timeline would take time and effort and although it would be nice for scheduling, it would mean that it would take a year longer for curriculum changes to take effect.

Another issue that can be resolved without an IT solution is that of scheduling without known class sizes for first year. Since the admission numbers are not known until after classes start, it is impossible to schedule the first year schedule with known class sizes. However, the later on in the summer the first year is scheduled, the more accurate the estimate of the class sizes. It would be a good idea to change the scheduling order and schedule first year last, after all the other years are completed. There were several reasons, listed earlier in the chapter for scheduling first year first. However, when the first year schedule has to be changed last minute due to unknown class sizes, it ends up being scheduled last anyway. The only difference is that time was wasted by scheduling it the first time. The scheduling department intends to try scheduling first year last in the upcoming year.

4.6 Conclusion

University course timetabling is not simply putting a bunch of constraints into a software program, pushing a button, and getting back a timetable. It is a long, laborious process that involves many people: the director of scheduling, the departmental counselors, the curriculum committees, the professors, and of course, the students. It is process that takes the entire year, starting with the creation of the next year's curriculum and calendar, continuing with collecting and verifying all data, and finally ends in the creation of a timetable. There are many surprises that come up and changes that must be made at the last minute, throughout the entire process.

To automate the university course timetabling process, one must consider this entire process, and not just the scheduling part of it. As shown above, there are many parts to the process and a lot of it is done manually, causing an already difficult job to become more tedious and to take longer than necessary and also creating more opportunities for errors and miscommunications. Automating parts of the process would provide the director of scheduling with more time to focus on creating a good quality timetable.

The complexity of this timetabling problem shows how difficult, if not impossible, it would be to create a definition of this problem that could be put into a mathematical model. Not only is there more than one issue to consider, the problem is extremely dynamic and is based on judgments as to what constraints can be relaxed as well as the data that has been gathered. It is clear that real problems, like the APSC problem are large and complex and we don't have a formal methodology for creating a problem definition, such as some that we saw in Chapter 3. As one suggested step toward such a methodology, in the next chapter, we will look at evaluation criteria created for the APSC problem. Having evaluation criteria is important as it enables you to look at how well the solution created from a problem definition fits back in the real world.

Chapter 5

Evaluation of the Timetable at the Faculty of Applied Science and Engineering at the University of Toronto

5.1 Introduction

The real-world evaluation of a solution to a model representing a real-world problem is an important step in an application. It allows one to discern what information to include in the problem definition as well as whether the problem definition was useful. An evaluation shows whether the problem definition resulted in a model whose solution could be used. As we saw in the previous chapter, the timetabling problem at APSC, like many real life problems, is messy and complicated. It involves many people communicating to try to achieve a timetable that meets a complex set of requirements and goals. It is important to keep in mind the criteria for evaluation of the timetable when one is creating the timetable. This chapter looks at the evaluation process developed for the faculty of applied science and engineering at the University of Toronto (APSC).

The creation of an evaluation system for APSC is a step toward formalizing evaluation as an integral part of the problem solving process. Before modeling, one should sit down with the client and define, independent of any eventual models, how to judge the quality of a solution.

The problem solving process, as described in Section 3.1, starts with the creation of a problem definition. In this case, the APSC's problem definition includes all the constraints and goals described in the previous chapter. Since the schedulers' main focus is to satisfy the hard constraints of the problem, it can be difficult to ensure the secondary objective: a good quality schedule from the student's perspective. They therefore wanted to develop a system for measuring the quality of a schedule so they could see how their solution to the timetabling problem worked in the real world, when the student's actually had to follow their schedules. In most of the timetabling literature, objectives such as the quality measures described in this chapter are put into an objective function of a given mathematical model [7, 13, 21, 25, 66]. The results of this analysis show that this is often unproductive, as it is extremely unlikely for such an objective function to accurately represent the quality criteria that it is supposed to represent. Quality is determined by using human judgment to make tradeoffs between satisfying constraints and obtaining quality on an issue-by-issue basis.

In this chapter, we present the motivation for creating an evaluation system for the timetabling problem at APSC. We then examine the process of determining which quality measures to include and provide a detailed description of the resulting quality metrics. The

metrics are followed by a description of the tool created in order to implement the evaluation system, directions for future work, and conclusions.

5.2 Motivation

There were several reasons that motivated the creation of an evaluation system for the APSC timetabling problem. The first motivator was the complexity of the problem's hard constraints. The timetable is put together almost completely manually and there are many complex constraints, as shown in the previous chapter. This results in most of the scheduler's energy being spent trying to create a schedule that does not violate any of the hard constraints and where all students have access to the courses in their curriculum. During the process, the scheduler does keep in mind the idea of creating a schedule that is of good quality for the students and whenever possible will adjust the timetable in order to achieve a good quality schedule. Nevertheless, there are many student sets with intertwining schedules to consider and balancing good quality across these student sets can be a nearly impossible task. The registrar's office at APSC thought that having a tool that could measure the quality of their schedule according to certain metrics would help them to see how good a job they were doing in creating a good quality schedule. It would also highlight the areas where they were not doing a great job. If this tool were to be used only on past schedules, the scheduler would know what to keep in mind the next time around. If the tool were to be used during the scheduling process, the scheduler would be able to address the problem areas.

Another reason that an evaluation system is useful is that it aids in the objectification of quality. Quality is subjective. What one person thinks of as a good quality schedule, another may think of as horrible. For example, some people may prefer to have a couple of days that are very full, even without a break, if it means they could have a day off, while other people would rather have the load spread evenly throughout the week. It is useful to have set quality measures by which to evaluate a timetable, as it makes sure that everyone is in agreement as to what should be motivating the scheduler's judgments when timetabling.

The third motivating factor for the APSC was the ability to compare timetables. They wanted to be able to compare quality in timetables across years and POSts. They were interested in seeing if they had improved over the years at creating a good quality schedule. As well, if they had gotten worse, in which aspects had the quality suffered. For example, over the years, more and more flexibility has been put into the curriculum. This makes it more difficult to create a conflict-free schedule due to the immense number of possible combinations of courses that students can take. As a consequence, as demonstrated below, the evaluation system shows an increase in the number of conflicts over the years. In this particular example, nothing can be done, but in other cases, the scheduler may be able to analyze why a certain area is getting worse and perhaps find a way to correct it. The registrar's office at APSC would also like to be able to use the evaluation tool to show the curriculum committee what effect the curriculum changes have had on the quality of the schedule. It is also useful to see if there are any POSts whose schedules are of considerably worse quality than the others. It may be a case that the curriculum does not allow for a better schedule, but it may also be the case that that particular POSt's schedule can be improved if

some of the other POSts compromise their quality, resulting in an overall better quality timetable.

The final motivating factor is to measure how well the timetable fits into the real world. This is of particular interest to my research as it is the connection to the problem definition phase described in Chapter 2. The combination of data, constraints, and quality measures is what defines the timetabling problem at APSC. They are what drive the scheduling process and ultimately the solution to the problem: a timetable. An evaluation system is a way to measure how well the solution works in the real world. It allows the registrar's office at APSC to see if their definition results in a timetable that can be used by the students. If it were to show that the quality of the schedule is so poor that the students cannot use it, APSC would have to change their definition by either adding or removing constraints or changing their definition of quality. It is a way of validating their problem definition and evaluating it in the real world.

5.3 The Quality Measures

The quality measures were developed through a series of meetings and interviews with the associate registrar and the director of scheduling at APSC. I developed a number of quality measures that, I believed, represented quality from the students' point of view and developed a prototype that displayed the results of the metrics in the form of bar graphs based on a preliminary set of timetable data. The associate registrar and the director of scheduling then reviewed the proposed metrics and suggested changes and several additional metrics. The resulting metrics are listed below, in no particular order:

1. Number of Conflicts
2. Days ending after 5pm.
3. Days without a lunch break
4. Student utilization
5. Days starting at 9am
6. Friday prayer break
7. Room utilization

Being that this is the first time that the associate registrar and the director of scheduling have had these metrics implemented in the database, obviously they will need to be refined. For example, in the first metric, conflicts between required courses are more serious than conflicts between electives, yet the database considers them to be the same. The same is true for the second metric, where ending the day at ten is much worse than ending at six.

In the following paragraphs, each of the metrics will be described in detail.

5.3.1 Number of Conflicts

This is a metric representing the number of conflicts a student has in their schedule. It was chosen because one of the main goals of the timetable at APSC is to create a conflict-free schedule for the students. Unfortunately, this is not always possible. A conflict is any time a

student has more than one activity scheduled. It is tabulated as the number of students with *x* activities in conflict. For example, if a student has three activities scheduled at the same time, they have three activities in conflict. If a student has two pairs of activities scheduled at the same time, they have two activities in conflict, counted twice. It is tabulated for all the students, grouped by semester, and for each semester separately, grouped by POSt.

5.3.2 Days ending after 5pm

This metric represents the number of times students have to stay late at school. It was chosen because a good quality schedule from the students' point of view would not require them to stay late. 5pm was chosen, as it is the standard time for the end of the school day. It is calculated as the number of students ending after 5pm *x* times in a week. For example, there may be 200 students that end after 5pm two times in a week and 500 students ending after 5pm, five times in a week. It is tabulated for all students, grouped by the semester, and for each semester separately, grouped by POSt.

5.3.3 Days without a lunch break

This metric represents the number of times students have no break between 11am and 1pm. It was chosen because a good quality schedule from the students' point of view would contain a break for lunch. 11am to 1pm is thought of as lunchtime, and was therefore chosen as the time boundary for the metric. It is calculated as the number of student-days in a week without a break between 11am and 1pm. For example, if there are 200 students without a lunch break twice a week and 100 students without a lunch break three times a week, then there are 700 student-days without a lunch break (i.e., 400 student days plus 300 student days). It is tabulated for all students, grouped by semester, and for each semester separately, grouped by POSt.

5.3.4 Student utilization

This metric represents the percentage of a student's school day spent in a scheduled activity. It was chosen because it is important for a student's time at school to be meaningful. A meaningful schedule is one where there are enough breaks to do some work and to get something to eat, yet not too many breaks. Student utilization on a given day is the number of hours where the student has an activity scheduled divided by the number of hours between the start of their first class and the end of their last class. The metric is calculated in several ways. Firstly, it is calculated as the average student utilization over the week. This is tabulated both for all students, grouped by semester, and for each semester, grouped by POSt. For example, the average utilization for first year civil engineering in the fall of 2004 may have been 75%. The metric is also calculated in terms of low, medium, and high utilization. Up to 40% utilization is considered to be low, from 40% up to 70% is considered to be medium, and 70% and higher is considered to be high utilization. There is a separate utilization level assigned to each student day. It is tabulated grouped by semester, and for each utilization category separately, it is grouped by POSt and semester. For example, for the student days utilization level graph, grouped by semester, there may be 100 student days with a low utilization, 500 with a medium utilization, and 1000 with a high utilization.

5.3.5 Days starting at 9am

This metric represents the number of times students have to start the school day early. It was chosen because a good quality schedule from the students' point of view would not require them to start early. 9am was chosen as it is, currently, the earliest time for the start of the school day. It is calculated as the number of students starting at 9am *x* times in a week. For example, there may be 200 students that start at 9am two times in a week and 500 students starting at 9am, five times in a week. It is tabulated for all students, grouped by the semester, and for each semester separately, grouped by POSt.

5.3.6 Friday prayer break

This metric represents the number of students that do not have a break between 12pm and 2pm on Fridays, a time when the Moslems are supposed to pray. There are enough Moslem students to make this metric important. The metric is calculated as the number of students that do not have a break in their schedule between 12pm and 2pm on Fridays. It is tabulated for all students, grouped by the semester, and for each semester separately, grouped by POSt. An interesting point to note is that there is no indication in the date of whether a student is Moslem or not. This metric counts all students regardless of their desire for a prayer break. As well, it does not appear that daylight savings time, which effects the Moslem prayer time, has been taken into account: yet another complication making timetabling challenging.

5.3.7 Room utilization

This metric represents the percentage of time that a room has an activity scheduled in it. This metric was chosen because the university has a requirement that all rooms must be used 50% of the time. It does not reflect quality from the students' point of view, but it is still useful information for the director of scheduling and is therefore included. It is calculated as the number of hours in a week where a room has an activity scheduled in it divided by 40, a standard week length. It is tabulated for each room, grouped by semester.

5.4 Evaluation System Setup

The evaluation system for APSC was created using a Microsoft Access database. The database is based on the *end* of semester data on student choices and course schedules. Therefore, any changes made to the schedule in the first few weeks of the semester and any students who dropped or added courses at the appropriate dates are included in the database. This is not the data available when the schedule is created, because the registrar's office at APSC does not have all the student choices and they do not know which students will change their choices once the semester has already begun. However, this data allows us to see how well the created schedule, made with many unknowns, works in the real world, once all the unknowns become known. The data currently in the database is from the 2004-2005 and 2005-2006 school years. It is the data from the end of the semesters, meaning that it contains changes that were made once classes had already begun.

The database is based on three tables:

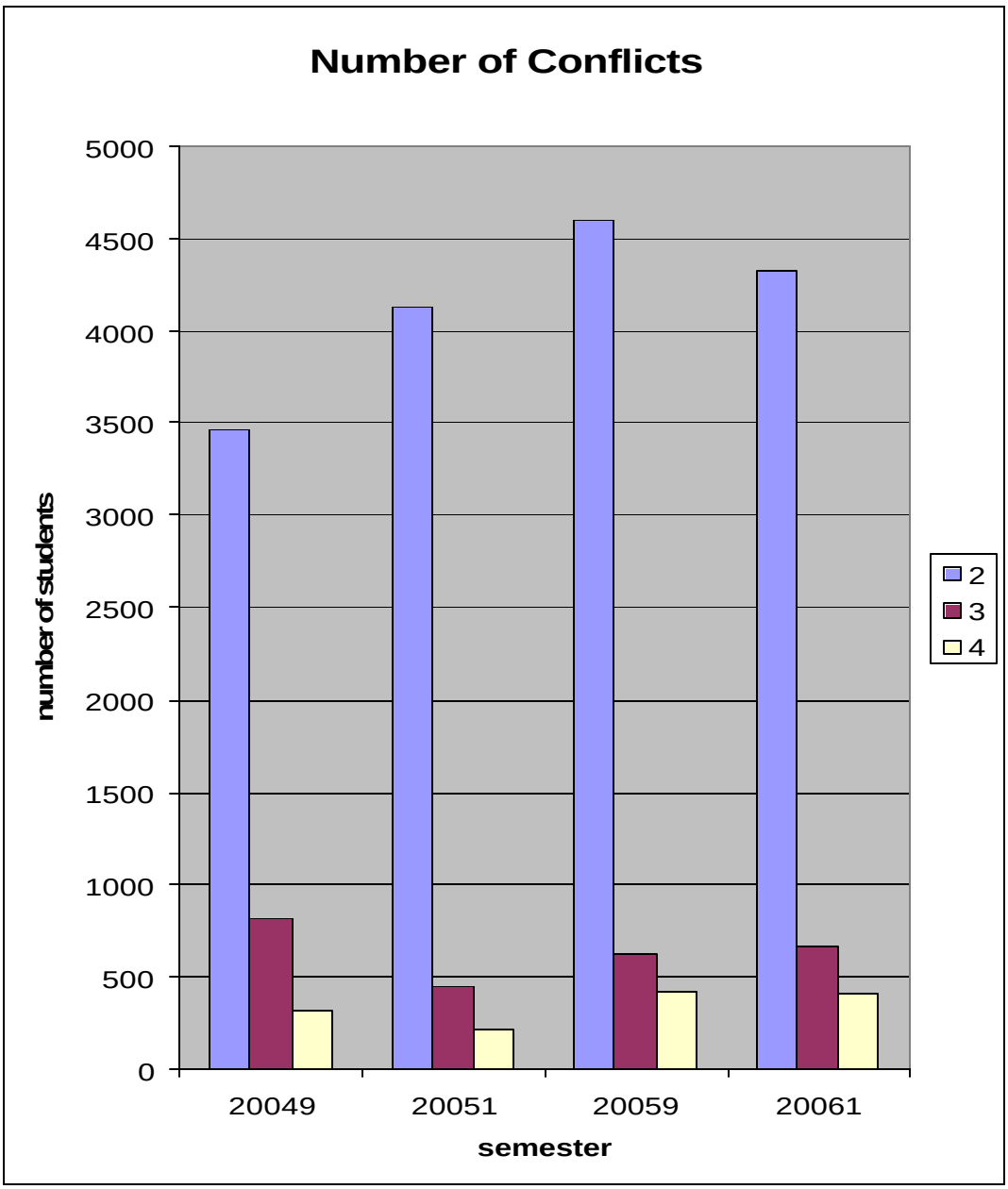
1. Student Choices
2. Course Schedules
3. Student Schedules

Student Choices contains data on each student. There is one row for each meeting of each course that each student is registered for. For each student, it includes the course code, the meeting code, and section number. Student Choices contains 86845 rows. Course Schedules contains data on each course. There is one row for each meeting of each course. For each course meeting, it contains the day, start time, end time, and location. Course Schedules contains 19294 rows. Student Schedules is a join on the first two tables; i.e. it combines the two tables into one large table. Student Schedules contains data for each student. There is one row for each meeting of each course that each student is registered for. For each student, there is the course code, meeting code, section number, day, start time, end time, and location. Student Schedules contains 308874 rows.

The metrics described in the above section are all calculated using SQL queries. Several of the metrics are calculated using a series of SQL queries. The code for all the queries can be found in Appendix A. The result of the queries are tabulated in the form of bar graphs and organized through the use of reports.

The bar graphs of the query results provide a visual display of the metric results. Below are several of the graphs to serve as examples. First is a graph of the first metric, number of conflicts, tabulated for all the students and grouped by semester.

Figure 5.1: bar graph of the results of the first quality metric, number of conflicts.

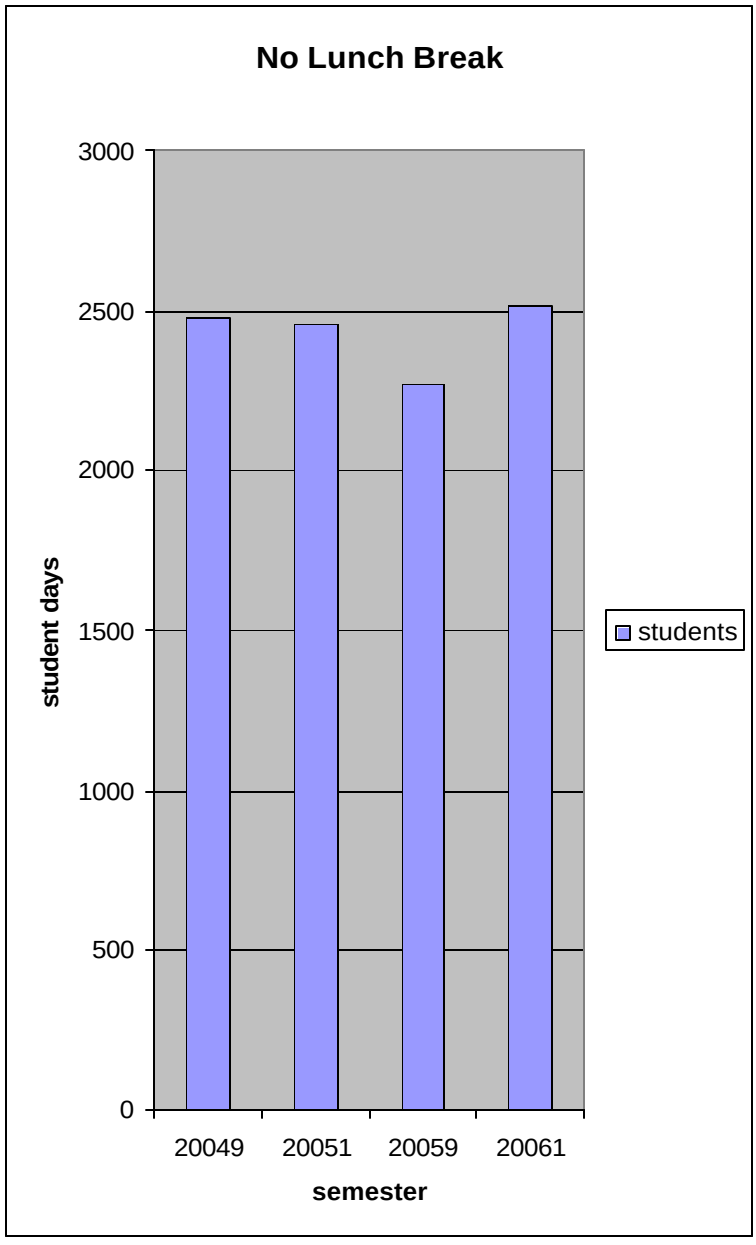


From this graph, one can see that the number of conflicts has risen by over 30% from 20049 to 20059, in the 2005-2006 school year. This rise can be attributed to the introduction of new, more flexible curriculums to the third and fourth year programs of several departments. Due to the large number of options available to the students, it became impossible to ensure that all possible combinations of courses be conflict-free. From the graph, the director of scheduling can see to what extent the curriculum changes have affected the schedule. Perhaps she can use this graph to illustrate this point to the curriculum committee and together they can decide what to do. They may choose to change the curriculum or they may choose to change the scheduling strategy. One way to do that is to

introduce several streams. The director of scheduling could then focus on keeping courses within the streams conflict-free and not have to worry about all the possible combinations of courses.

Next is one of the graphs of the third metric, No Lunch Breaks. It shows the number of student days during the week without a lunch break.

Figure 5.2: Bar graph of the results of the third quality metric, no lunch breaks.

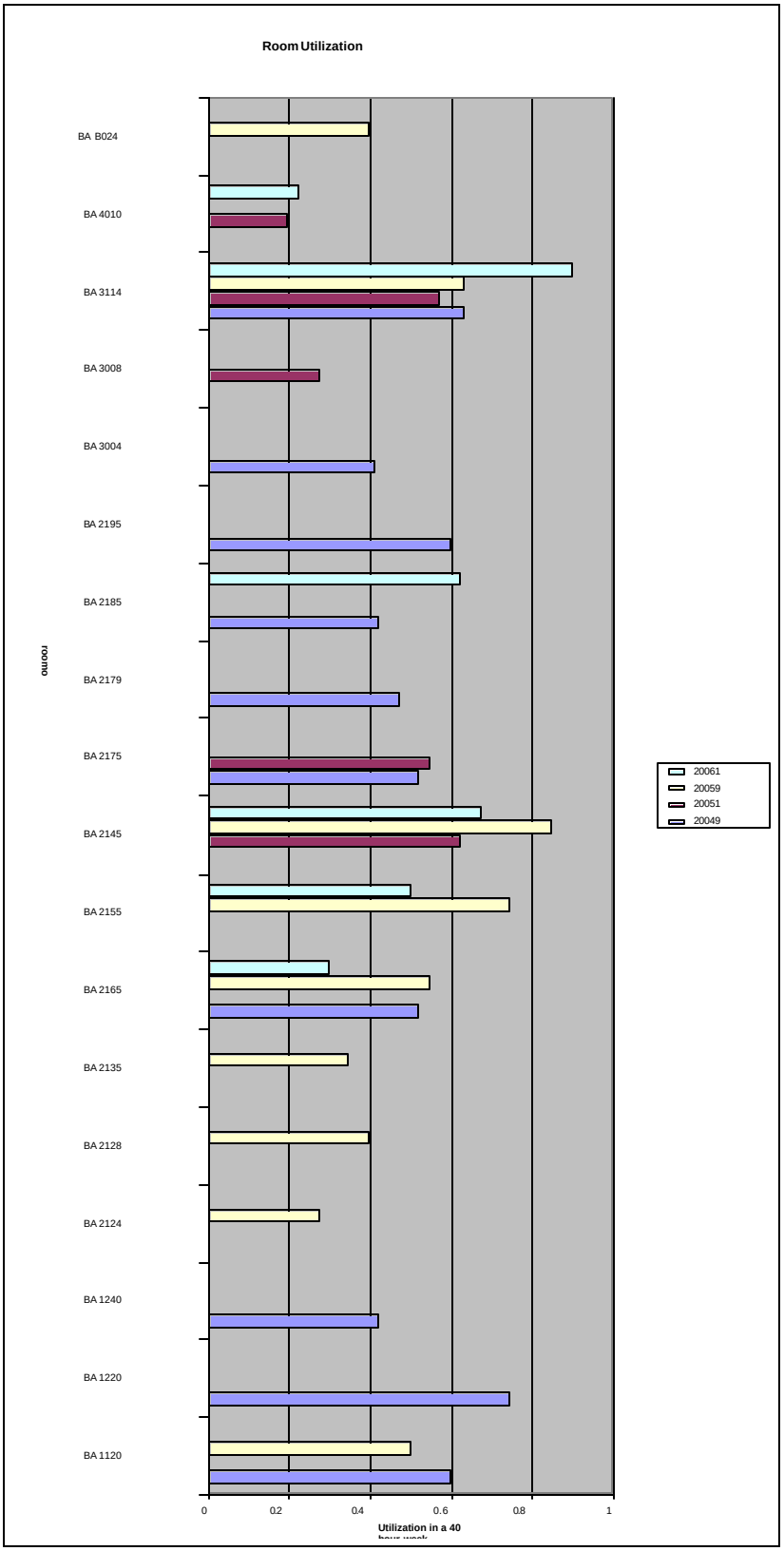


Here too, we can see that the number of students without a lunch break has slightly increased in the winter 2006 semester. Again, this can be attributed to the change in

curriculum. The registrar's office at APSC may decide that the increase is insignificant. If they do not think that the increase is insignificant, seeing that the curriculum changes have affected the quality of the schedule in several areas may make the faculty more likely to change something. Perhaps they will take a different approach when designing the curriculum and consider the effects on the timetable more seriously than they have previously. Perhaps, they will adjust their expectations for the quality of the timetable.

The next graph is a subset of the graph for the seventh metric, room utilization:

Figure 5.3: Bar graph of the seventh quality metric, room utilization.

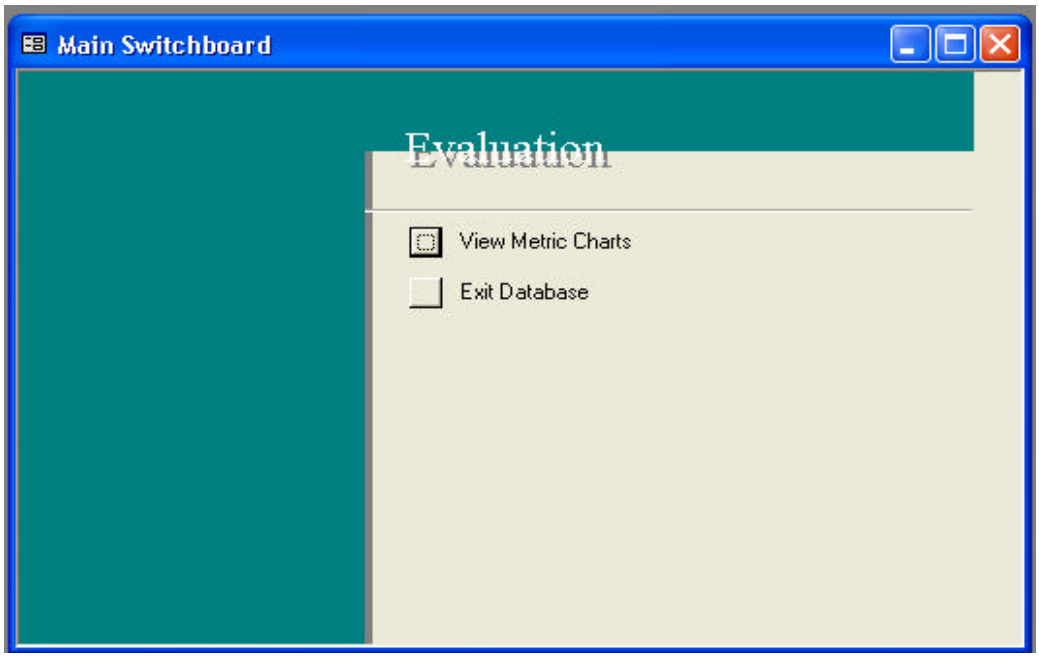


By boking at this graph, the director of scheduling can see which of the faculty's rooms are not used enough; i.e. they do not meet the university requirements. For example, GB 308, BA 2124, BA 2128, and BA 2135 are used less than 50% of the time. The registrar's office at APSC can then look at those rooms and try to discern why they are not used. In this case, the BA rooms are quite small, they can fit less than 40 students and all the rooms listed above are not electronic. The registrar's office can then decide if it is worthwhile to make the rooms electronic. The director can also look at the graph to see which rooms are over-utilized. For example, LM 217, ES 1050, and MC 402, which are not shown in the graph, are used more than 40 hours a week. The director of scheduler can now schedule these rooms first since they are under high demand and perhaps see if there is another room of the same type that can be used instead.

The remainder of the bar graphs can be found in Appendix B.

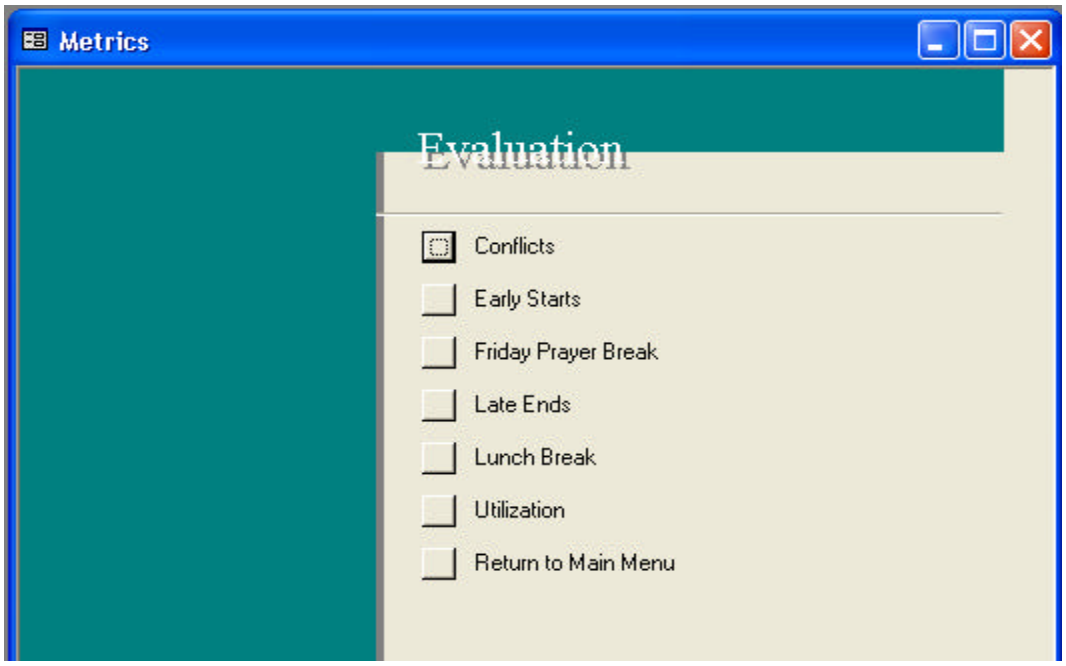
The user, through the use of switchboard menus, can easily access the reports containing the bar graphs. The switchboard menus are designed to be simple, clear, and easy to use. Below are snapshots of several switchboard menus.

Figure 5.4: The main switchboard menu.



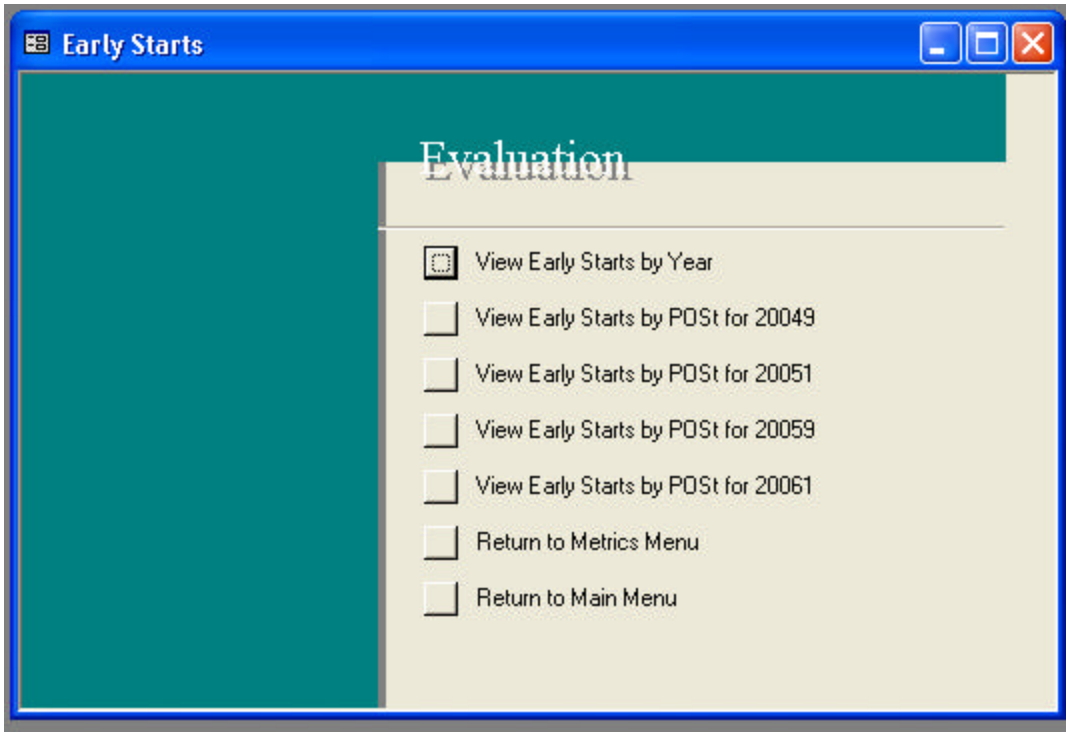
The main switchboard opens when the database is opened. The user has a simple choice of viewing the charts of the metric results or exiting the database.

Figure 5.5: The metric charts menu.



The metrics charts menu opens up when view metric charts is chosen on the main menu. The user can choose to view charts for any of the listed metrics or return to the main menu.

Figure 5.6: The Early starts menu.



The early starts menu opens up when early starts is chosen on the metric charts menu. The user can choose to view any of the charts on that metric or they can return to either the metric charts menu or the main menu.

5.5 Future Work

There are a couple of ways to extend on the work described in this chapter. From a research perspective, it would be useful to study quality metrics of all forms to see if there are metrics that could be placed in an objective function and that would accurately represent the desired qualitative effect.

Another way to extend the evaluation database is to add more, and more detailed, quality metrics. There are more complicated metrics, such as how courses are spread over the week and how students' breaks are spread over the course of a day. These metrics would provide more information to the schedulers.

A third way to extend on the evaluation database is to incorporate the database into the scheduling process at APSC. Microsoft Access can interface with Course Planner, the software used to create the timetable at APSC. The database could then be used to inform the scheduler as they make decisions throughout the scheduling process. For example, when the scheduler chooses to place a meeting for a course in a specific timeslot, it could show them how that changes the value of the quality metrics.

5.6 Conclusions

The registrar's office at APSC has decided to take on the evaluation database. They are pleased with the information it provides. They see themselves using it to evaluate their schedules as well as a tool to show the curriculum committee how the new curriculum has affected the schedule. They hope that it will provide enough proof to showcase the faculty's need of more resources. The registrar's office would like to change some of the existing metrics, such as changing metric number 5, days starting at 9am, to days starting at or before 9am. They would also like to add some additional metrics, such as calculating the number of hours of consecutive class, and drill down further on the existing metrics by analyzing them according to year and individual student sets. The IT department of the registrar's office will be taking over the database.

It has become evident that having a concrete way to measure the solution to a real world problem is very useful. It provides validation to the current solution method, and it also provides directions for improvement. Especially in this case, where the solution to a problem is judged based on quality, a subjective measure, concrete metrics provide objectivity as well. The evaluation tool also acts as a political tool because the director of scheduling can take the results to the board and show evidence that more resources are needed, perhaps changing the problem.

Another conclusion that has emerged through the creation of an evaluation system for APSC is that putting quality measures such as those described in Section 5.3 into an objective function of a mathematical model, as is done in many cases in the literature, is difficult and it would not accurately represent the desired quality metrics. For example, it is not enough to simply minimize the number of student days without a lunch break, as in quality metric number 3, because there are many cases where not having a break between 11am and 1pm is not a bad thing. A student may have a day where they start at 11am or where they end at 1pm. Both of these cases are considered to be a good quality day, even though there is no actual lunch break. Also, the metrics are misleading because they don't differentiate between levels of quality. For example, a day that ends at 10PM is considerably worse than one that ends at 6PM. The evaluation criteria must be finer to represent this. It is up to the APSC schedulers to decide where to draw the line so that this can be better represented. Furthermore, on top of single metrics being misleading, it is unclear how to balance all the metrics automatically. It is important to use the evaluation metrics as one of many tools to provide input into human decision making when making tradeoffs and developing a good quality schedule.

As far as the problem definition problem goes, this chapter has shown us that evaluation criteria are complex and may be difficult, if not impossible, to incorporate into a traditional optimization function. Since this is the case, it is important to evaluate how a given solution works back in the real world. In the case of a manufacturing plant, one might create a simulation. Here, a detailed set of evaluation criteria is useful and necessary if we are to continue on in attempting to find an automated timetabling solution.

In the next two chapters, we will look at the next step in the problem solving process, namely modeling and solving a timetabling problem. In Chapter 6, we look at existing literature on university course timetabling, and in Chapter 7, we experiment with a university timetabling problem, similar in part to the APSC problem that we found in the literature.

Chapter 6

Automated University Course Timetabling – Literature Review

6.1 Introduction

The timetabling problem involves scheduling lectures attended by both students and teachers into times and rooms. Typically a weekly timetable is created. Historically, people scheduled their timetables manually. This requires many hours of work and often the resulting timetable does not meet all the requirements. Because of this, a lot of research has been done in the area of automated timetabling and there are many different techniques used in the literature in order to solve the university course timetabling problem. The techniques span traditional operations research (OR) methods as well as, more recently, artificial intelligence (AI) methods have entered the picture.

Constraint programming (CP), a relatively new AI field, is becoming more common as a tool for solving timetabling problems, yet it has not been very successful. This is surprising because the timetabling problem seems to be well-suited to CP. This is due in part to the difficulty involved in defining the problem's constraints. CP provides flexibility when formulating the problem. Also, the constraints are often very hard to satisfy, making them suitable for CP propagation techniques. In Chapter 7, several models will be used to solve a university course timetabling problem. The main techniques used are CP, Integer programming (IP), and decomposition.

In this chapter, we describe the course timetabling problem. We then look at solution techniques and approaches found in the literature. Following that, we provide a brief presentation of CP and finish with some conclusions.

6.2 The Course Timetabling Problem

The course timetabling problem involves scheduling the delivery of courses into a specific number of rooms and timeslots. The delivery of a course usually involves lectures, tutorials, and occasionally labs. Courses can be mandatory or elective for students in a given program. Courses with common students conflict and shouldn't be scheduled in the same period. Room sizes and room availability also impact the schedule. The course timetabling problem is known to be NP-hard even in the simplest of cases [8].

6.2.1 Problem Formulation

The formulation of the course timetabling problem takes many forms. Each university or institution has its own unique structure and set of constraints. The following formulation is taken from [66].

There are i days, j time periods, k student groups (students with common courses), l faculty members, m courses, and n rooms. There are two basic variables. The first, x_{ijklmn} is 1 if course m, taught by teacher l, to the group of students k is scheduled on day i, in period j, and in room n. It is 0 otherwise. In [66], the above sets of variables are grouped in order to make modeling of constraints easier. An example of some groups are K_l - a group of students for which teacher l offers a course, L_i - a lecturer available on day i (this extends to L_{km} and L_{ki}), the same is done for M where courses are taught by a given teacher l, M_l, or for a certain k, M_k, and for I and J when a room n, I_n and J_n, or a teacher l, I_l and J_l, is available.

6.2.2 Constraints

Every university or institution has its own set of constraints. In timetabling problems these constraints are usually categorized into two groups. Hard constraints are ones that cannot be violated if a timetable is to be feasible. Soft constraints, sometimes referred to as preferences, are ones that the school does not want violated, but the timetable can exist with some violations. The best timetable will have the fewest soft constraint violations.

Hard Constraints

In [66] the hard constraints are as follows:

- There should be no conflicts. No teachers, student groups, or rooms should be assigned to more than one class at a time.
- The timetable should be complete. All the courses should be in the timetable in the correct number of periods.
- All pre-assignments of rooms or times should be honored.

Soft Constraints

Often, the soft constraints are represented in an objective function. This is the case in [66] where the soft constraints are represented as costs for each assignment. The soft constraints are the preferences for times and rooms. An assignment of the variable x that is less favorable will have a higher cost. The objective function will then be minimized. The result is that as many preferences as possible will be respected while not violating any hard constraints. The preferences are determined by looking at requests from teachers and student groups. The costs are also meant to minimize the number of room changes a student group has to make.

6.3 Solution Techniques

The timetabling problem is well-researched because it effects many institutions. There are many different techniques used in the literature to solve the problem. The techniques span traditional operations research (OR) methods as well as, more recently, artificial intelligence (AI) methods have entered the picture.

6.3.1 Operations Research

Timetabling has historically been considered an OR problem. The first research on automated timetabling began 30 to 40 years ago in this field. In this section, we look at some OR techniques that are used to solve course timetabling problems.

Graph Coloring

De Werra reduced a timetabling problem to a graph colouring problem [69]. Each lecture of a given course is assigned a vertex and a clique is made between the lectures for each course. Edges are introduced between cliques if two courses are conflicting, they share a teacher or students. The graph colouring technique is used in several papers in order to solve a part of the course timetabling problem. Some create the initial assignment of courses to times using graph colouring and then use a local search technique such as simulated annealing, discussed later in this section, to optimize the timetable [5]. Some use graph colouring to simplify the problem. For example, graph colouring may be used to divide the large problem into smaller ones and then other solution methods can be applied [11].

Integer and Mixed Integer Programming

Many papers use integer programming techniques to solve the course timetabling problem. One of the earliest mathematical programming timetabling papers uses layouts, a statement of the curriculum and its organization, to simplify planning and to suppress a certain amount of detail [67]. Other techniques used include Lagrangian relaxation to assign classes to rooms [17], or formulate the problem as a transportation problem [22]. Probably the most popular integer programming technique is to formulate the problem as a 0-1 optimization problem or an assignment problem [14, 21, 66].

Network Flow

Several authors suggest using a network model for the course timetabling problem. One example uses a network model with three levels [18]. The first level is the departmental level containing a vertex for each department. The second is the faculty level containing a vertex for every teacher and course combination. The third level is the room and time level with a vertex for every room and time combination. The network model can be solved in polynomial time, but it does not ensure that a teacher is not assigned to two courses at the same time. The paper, therefore, first solves the network model and if there are conflicts, it uses the solution as a starting point for a search to find a solution without conflicts.

6.3.2 Artificial Intelligence

Recently the field of AI began to tackle the timetabling problem with newer, promising heuristics.

Local Search Techniques

It is very common to find an approach that uses one method to find an initial solution and then follows up with a local search technique to optimize. Local search techniques are very popular because the course timetabling problem is known to be NP-hard. Therefore, local search heuristics can be used to search for good solutions as opposed to using a mathematical program to find an optimal solution, which may take too long to find. Local search techniques move from one solution to another by making a "move" to a neighbor of the current solution. Neighbors, as well as moves, may be different for different problem models. Solutions are compared based on an objective function that needs to be minimized or maximized.

The main local search techniques found in the automated timetabling literature are simulated annealing [5, 63] and tabu search [3, 7, 13, 16]. Simulated annealing uses a cooling rate to decide whether or not to accept the best neighboring move. Sometimes a move will be accepted even if it is not better than the current solution. This is done to prevent getting stuck in a local optimum. Kostuch uses two stages of simulated annealing once there is an initial timetable [5]. The first stage swaps already created timeslots and the second swaps individual events.

Tabu search appears to be the most popular local search for timetabling problems. Tabu search keeps a finite list of the most recent moves. While on the list, these moves cannot be reversed. This is the tabu search way of avoiding local optima. A common move is moving the timeslot of one lecture [13]. This way, neighboring solutions are identical except for the time of one lecture. Another common technique used in tabu search is relaxing hard constraints [13]. Often hard constraints are relaxed in order to give the method freedom while moving through the search space.

Other local search techniques found in the literature include genetic algorithms [65, 69] as well as other local search techniques using several different diversification methods, such as the one found in [25].

Logic Programming

There are some papers that use logic programming to solve timetabling problems [15, 20, 26]. Kang & White propose a logic programming method to the timetabling problem using PROLOG, a language that enables them to express constraints declaratively [20]. A heuristic reschedules conflicts by finding a so-called "equivalent" lecture and reassigning it to a different timeslot so that the conflicting lecture can be placed in its spot.

Constraint Logic Programming

A constraint logic programming (CLP) system generates values for variables and propagates through constraints so as to remove inconsistent values and shrink the search space. The

basic method is a backtrack search, but the constraints provide look-ahead capabilities. Gueret et al. model constraints using the built-in constraints provided by CHIP, a popular CLP language [65]. The paper compares four labeling strategies. Abennader & Marte use constraint handling rules to model a university timetabling problem [64]. They use a partial CSP (PCSP), where each constraint has a weight. Each value in a variable's domain has an assessment. Propagating through a soft constraint will change the assessment value while a hard constraint will remove values.

Constraint Programming

Constraint programming (CP), a relatively new AI field is becoming more prominent in the timetabling research [69]. Programming with constraints allows more flexibility when formulating the problem. This is important because the problems are usually complicated and unclear. CP has declarative constraints like CLP, but it is not as restrictive because constraints can be integrated into imperative languages like C++ and Java [70]. Sometimes CP is used to obtain an initial solution [63] or as part of the solution process [3]. Cambazard et al. create a system for over-constrained and dynamic problems [3]. At first, the problem is solved with all the constraints. If it is found to be over-constrained, the system searches in the space of possible relaxations. The details of CP and how it can be used to solve timetabling problems will be discussed further in section 6.4.

6.3.3 Other Methods

It is very common to see a combination of methods being used to solve the timetabling problem. For example, Cooper et al. combine several heuristics [19]. At its base, their solution method uses bipartite graph matching. The algorithm identifies groups of lectures that conflict. It then improves on the timetable by choosing from the possible assignments. Resources are assigned to lectures using a brute force algorithm or a beam search. More recent papers discuss the possibility of combining IP and CP [68]. Another technique used, although less prevalent in the literature, is goal programming. Shniederjans & Kim divide the constraints of the problem into three categories [24]. The first is a set of goals that ensure course offering requirements, the second is the set of faculty teaching load assignment goals, and the third is preference goal constraints. Each set of goals can have different weights/priorities.

Another phenomenon seen in many timetabling papers is the use of an interactive system [6]. In such systems, there is a large manual part. It is more than the user being able to adjust the timetable at the end. Interactive systems are popular because the evaluation of timetable quality is complicated. It is often hard to describe to a computer what makes one timetable better than another, as was shown in Chapter 5.

6.4 Constraint Programming

CP is described as the study of computational systems based on constraints [70]. A constraint satisfaction problem (CSP) is a problem defined over finite domains, as is the case in timetabling problems. A CSP is a set of variables each with a domain and a set of constraints that restrict which values variables can take. A solution is an assignment to every

value with all constraints being satisfied. The search can be for any solution, all solutions, or for an optimal solution defined by an objective function.

6.4.1 Search and Heuristics

There are different methods for systematically searching through the search space. The classic is chronological backtracking (BT) that incrementally assigns values to variables and when a dead end is hit it goes back one step in the tree and assigns a different value to that variable. BT has three drawbacks [70]:

- Thrashing (repeated failure for the same reason)
- Redundant assignments because conflicting variables are not remembered
- Late detection of conflicts.

To overcome this, different search methods and heuristics have been developed for both going forward, deciding which variable to assign next and to what value, and going back after a dead end is hit. One method for going back after a dead end is backjumping (BJ) [72]. BJ looks at which variables the dead end variable had conflicts with and jumps back to the most recent one. It backtracks to the lowest level such that it can prove that it will not miss a solution. A classic going forward heuristic is fail-first, to pick the variable most likely to fail. More details about fail-first as well as methods for calculating which variable is most likely to fail first can be found in [73].

6.4.2 Propagation

Another technique used to minimize late detection of conflicts is propagation. As values are assigned to variables, the constraints are checked to see what values can be removed from the domains of the remaining unassigned variables. Standard CP solving is a combination of search with propagation at each node. A simple form is forward checking (FC). After a variable is assigned, FC checks the unassigned variables directly connected to the just-assigned variable and removes values that conflict with the just-made assignment.

An important concept in constraint propagation is local inconsistency, when a particular instance of a set of variables satisfies a set of constraints but cannot be extended to more variables and therefore cannot be part of a solution. Therefore, to prevent unnecessary backtracking, it is good to maintain consistency in a CSP [88]. Arc consistency is one type of local consistency. A constraint is arc consistent if for every variable in the constraint, for each of its values, there exists a value in the domain of all the other variables in the constraint such that the constraint is satisfied [88]. For global constraints, which will be described in the next paragraph, pruning the domains of the variables so that constraints remain arc consistent is referred to as generalized arc consistency (GAC). Generally, a compromise needs to be made between the level of consistency maintained (i.e. the amount of domain pruning) and the cost of performing that constraint propagation at every node in the search tree [88].

Global constraints are constraints over more than two variables. They usually have propagation algorithms developed specifically for them so that the propagation does more pruning or is less costly than if the same constraint was expressed using several smaller constraints [88]. They also make modeling a problem more natural. One example of a global constraint is the all-different constraint. An all-different constraint is a constraint over a set of variables that must all have different values. Without that global constraint, one would have to enumerate each pair of variables and constrain them to be different (i.e., with a binary not-equals constraint). Using the all-different constraint takes less effort and maintaining consistency on the all-different constraint removes more values from the domains of the variables than the alternative of writing out each pair of variables and requiring them to be different.

6.4.3 Modeling

Modeling can effect how well a problem can be solved [30]. Modeling techniques include using combined constraints and implied constraints as well as using different sets of variables. Combined constraints are made up of more than one constraint with the same scope. When local consistency is run on the constraint, it will only allow tuples that are allowed by both [71]. Implied constraints are constraints that are implied by the already existing constraints and they are therefore logically redundant. They can, however, reduce the required search effort [71]. Using a different set of variables can be useful because one set of variables may make modeling a constraint easier than another. Due to the interaction of search heuristics, algorithms and the model, it is hard to know which model is best [71].

6.4.4 Beyond the Basic CSP

Two extensions to the basic CSP are applicable to timetabling problems. The first is when there is an objective function. These problems, Constraint Satisfaction Optimization Problems (CSOPs), look for an optimal solution, which is one that minimizes or maximizes a given objective function. In timetabling problems, this is usually minimizing the soft constraint violations or maximizing the student and staff preferences. The second CSP extension is the partial CSP, PSCP. This is used when a problem is over-constrained, when there is no solution that satisfies all the constraints. This often occurs in timetabling problems. For example, in a paper by Cangalovic & Schreuder a tabu search is done in the space of possible relaxations [3]. Not all the constraints can be kept as hard constraints.

CSOP

The most widely used method for dealing with CSOPs is branch and bound [70]. A heuristic function is used to estimate the best complete solution from the partial solution that exists so far. This estimate is used as a bound on that section of the tree. If it is not as good as a solution that already exists that section of the tree need not be explored.

PCSP

PCSPs are a method of targeting over-constrained problems [70]. Like a CSOP, a numerical value is given to each assignment or partial solution. The value is, in effect, a rating of how well it solves the problem, knowing that some constraints are not satisfied. A PCSP is very

similar to a CSOP, except all the constraints do not need to be satisfied. This is achieved by allowing more variable assignments to be considered acceptable. A particular constraint is weakened by enlarging the domains of the variables effected by the constraint. Many of the standard algorithms, such as backjumping, arc-consistency, and branch and bound, can be extended to work for a PCSP [70].

6.5 Conclusions

The work discussed above motivates the work in Chapter 7. Since CP seems to be applicable to the timetabling problem, it is surprising that it is not more present in the research. We, therefore, implement several models, using CP, as a first step to understanding if CP can be successful in timetabling. We look at CP on its own, CP in contrast with an IP model, and CP in combination with IP using decomposition.

Chapter 7

Investigating Decomposition and Constraint Programming for Timetabling Problems

7.1 Introduction

The timetabling problem seems to be well-suited to constraint programming (CP). This is due in part to the difficulty involved in defining the problem's constraints as well as the difficulty involved in satisfying the constraints. CP provides flexibility when formulating the problem, which is helpful for modeling. Also, the difficult constraints seem to be suitable for CP propagation techniques. However, most of the successful timetabling work appears to use some form of local search as was shown in Chapter 6. This chapter is an investigation of CP to evaluate if it can be successful in timetabling. We look at CP on its own, CP using decomposition, and CP in combination with IP using decomposition. The use of CP decomposition has never been used for solving timetabling problems. Recently, research has been done combining CP and IP [91], but it is a novel approach in timetabling.

The timetabling problem involves scheduling lectures (i.e., meetings of students and teachers) into times and rooms. Typically, a weekly timetable is created. Historically, people scheduled their timetables manually. This requires many hours of work and often the resulting timetable does not meet all the requirements. Because of this, a lot of research has been done in the area of automated timetabling as was seen in Chapter 6.

This chapter discusses several models designed for solving the course timetabling problems of the 2003 international competition of the Metaheuristics Network. We created six models for this problem. There are three monolithic models (a CP model, a CP scheduling model, and an Integer Programming (IP) model) and three decomposition models (a CP/CP model, a CP Scheduling/CP model, and an IP/CP model).

In the following section, we will describe the specific problem instances used to test the CP model. The next section discusses the models. Then, we will describe the experiments and results. We will provide a discussion of the results as well as compare our result to those of the metaheuristics competition entries. Finally, we will conclude and discuss future work.

7.2 Problem Definition

The problem used for the experiments in this paper is taken from [75]. Ben Paechter designed the course-timetabling problem for the Metaheuristics Network, who used twenty instances of the problem for an international competition in 2003. The problem consists of a set of events to be scheduled in 45 timeslots; nine periods on each of the five weekdays. There is a set of rooms with features and a size, a set of events that require specific room features, a set of students, each of whom attend a number of events, and a set of features that are characteristics of rooms and requirements of events. A feasible timetable is one in which all events have been assigned to a time and a room so that all the hard constraints are satisfied.

The hard constraints are as follows:

1. No conflicts for students - A student conflict is any time a student is scheduled to be attending more than one event at a time.
2. The room assigned to an event must be large enough to hold all students attending that event and it must possess all the features required by that event.
3. No conflicts for rooms – A room conflict is any time a room is scheduled to have more than one event at a time.

The competition gave a penalty of one point for each soft constraint violation. The soft constraints are as follows:

4. A student has a class in the last slot of the day.
5. A student has to attend more than two events consecutively.
6. A student has a single class on a day.

The competition instances contained 350 to 440 events taken by around 200 students. There were 10 or 11 rooms containing 10 or 11 features.

7.3 Models

We created six models to solve problems of the above form using the ILOG Optimization Suite. The models are as follows:

- CP 1 is a monolithic model that uses constraint programming alone. It uses ILOG Solver 6.2, a constraint-based optimization engine.
- CP Scheduling 1 is a monolithic model that uses constraint programming as well, only it models the problem as a scheduling problem. It uses ILOG Scheduler as well as ILOG Solver.
- IP 1 is a monolithic model using integer programming (IP).
- CP 2 is a decomposition model. It contains two sub-models, both of which use Solver. The two sub-models work together in a way that resembles decomposition.

- CP Scheduling 2 is also a decomposition model that works in the same way as CP Model 2; only the first sub-model uses Scheduler and Solver while the second sub-model uses Solver alone.
- IP 2 is another decomposition model; only this one has one model that uses integer programming and one that uses constraint programming.

The following table summarizes the models:

Table 1. Model Descriptions.

Model	**Type**	**Master Problem**	**Sub Problem**
CP 1	Monolithic	CP using ILOG Solver	N/a
CP Scheduling 1	Monolithic	CP as a scheduling problem using ILOG Scheduler	N/a
IP 1	Monolithic	MIP using ILOG Cplex	N/a
CP 2	Decomposition	CP using ILOG Solver	CP using ILOG Solver
CP Scheduling 2	Decomposition	CP as a scheduling problem using ILOG Scheduler	CP using ILOG Solver
IP 2	Decomposition	MIP using ILOG Cplex	CP using ILOG Solver

Data

The data used in the models is as follows:

- D: set of days – the five weekdays.
- P: set of periods – nine periods on each day.
- T: set of times – day and period combinations. For example, the third period on the second day is time 11.
- TS: set of timeslots – time and room combinations. For example, the first time in the fourth room is timeslot 3.
- S: set of students.
- R: set of rooms.
- E: set of events – the events in the timetable.
- E_s: set of events that student s attends, s ? S.
- $size_r$ – room r's capacity.
- $features_r$ – an array of the features that the room r possesses.
- $size_e$ – the number of students attending event e.
- $features_e$ – an array of the features required by event e.
- $attending_e$ – an array of the students attending event e.

- *LastPeriod* - an array of auxiliary variables. They are binary variables, where $LastPeriod_e$ takes the value 1 if event e is scheduled to be in the last period of a day and zero otherwise.
- $Schedule_s$ – It is in the form of a 2D variable array of days by periods. It represents a student's schedule. An entry in the schedule will take the value 1 if the student has an event in that day and period combination, and zero otherwise.
- *EventDay* - It is an array of auxiliary variables indexed from zero to |E|-1. The values are the day index, from zero through four, on which the event takes place. In the IP models, *EventDay* is a 2D array of days by events. The variables take the value 1 if an event is on a given day and zero otherwise.
- *StudentEventDays* - It is a separate array of auxiliary variables for each student that is the same as *EventDay* except that it contains only the events that the student is taking.
- *EventTimes* - It is an array of auxiliary variables indexed from zero to |E|-1. The values are the time index, from zero through 44, during which the event takes place. In the IP models, *EventTimes* is a 2D array of times by events. The variables take the value 1 if an event is at a given time and zero otherwise.
- *StudentEventTimes* - It is a separate array of auxiliary variables for each student that is the same as *EventTimes* except that it contains only the events that the student is taking.
- $Events_t$ – It is an array containing the indices of the events scheduled at time t, where t ? T.

7.4 The Monolithic Models.

The first three models are monolithic models. They are each a single model designed to solve the problem described in section 7.2.

7.4.1 CP 1

CP 1 is a monolithic CP model.

Decision Variables

For each event, e ? E, we have one decision variable.

- ts_e - It takes a value from 0 through |TS|-1. It represents the timeslot that the event is scheduled in. Recall that a timeslot value corresponds to a time and room.

Set Up

The model is set up using three variable arrays that are linked to each other. The first is a 3D array of days by periods by rooms.

$$\forall d \in D, \forall p \in P, \forall r \in R, EventsAssignments[d][p][r]$$

The domain of each variable is the set of events and the value is the event assigned to day d, period p, and room r. The second array is a 2D array of times by rooms. Each time period variable has an array of rooms equal to the array of rooms for the corresponding day and period combination.

$$\forall d \in D, \forall p \in P, EventTimePeriod[t] = EventAssignments[d][p], \text{ t = d*|P| + p – 1.}$$

Here, too, the domain is the set of events and the value is the event assigned to the specific time period t, and room r. The third array is an array of timeslots.

$$\forall t \in T, \forall r \in R, Timeslot[ts] = EventTimePeriod[t][r], \text{ ts = t*|R| + r -1.}$$

Timeslot[*ts*] is the variable representing the event that is scheduled in timeslot ts. This value is also represented in the 2D array where *EventTimePeriod*[*t*][*r*] is the event that is scheduled at time t and in room r, and in the 3D array where *EventsAssignments*[*d*][*p*][*r*] is the event scheduled on day d, at period p, and in room r. The Timeslot variables are linked to the decision variables in such a way that *Timeslot*[*ts*] has the value e, while ts_e has the value ts.

$$\forall e \in E, \exists ts \in TS, Timeslot[ts] = e \leftrightarrow ts_e = ts$$

Since there may not be as many events as there are timeslots, dummy events are created so that there is the same number of events as there are timeslots. There are no students attending the dummy events and they require no features. There are, therefore, no constraint violations for dummy events because all the soft constraints involve the students and there are no students attending any dummy events.

The following modifications were made to the model in order to increase efficiency:

- Requiring a dummy event to be scheduled before any dummy event with a higher index enforced an order. This is possible since dummy events are identical and it doesn't matter what order they are in.
- Symmetry was reduced using lexicographic ordering constraints. For one array of variables to be lexicographically ordered before another means that the first non-zero entry in the array must be less than the first non-zero entry in the other array. The days in the model are symmetric, meaning that they can be interchanged without affecting the quality of the schedule. Therefore, lexicographically ordering constraints were imposed on the days. This ensures that the same combination of events will not be tried for each of the five days, but rather, only once [86].

$$\forall d \in D, day[d] <_{lex} day[d+1]$$

day[*d*] refers to the full assignment of day d, meaning all the events assigned to a particular day.

Constraints

There are three hard constraints and three soft constraints that are all common to every model. The constraints were described in section 7.2.

Hard Constraints

(1) There can be no conflicts for students. This constraint is modeled as an all-different constraint on the start times on each student's events.

$$\forall s \in S, alldifferent(EventTimes_s) \tag{1}$$

(2) Size and feature requirements must be respected. For each event, every room is checked, first for size and then for features. If a room is not large enough for the event or if it does not contain all the required features, any timeslot representing that room is removed from the domain of ts_e using a not-equals constraint.

$$\begin{aligned} &\forall e \in E, \forall r \in R, \ \ if\ (size_r < size_e) \\ &\vee (\exists f \in\ feature_e \not\subset\ features_r) \\ &then(\forall t \in T, ts_e != t+ \mid R \mid \cdot r) \end{aligned} \tag{2}$$

(3) There can be no conflicts for rooms. This constraint states that no room can be used by more than one event at a time. In CP 1 it is represented by an all-different constraint on the array of timeslots. This means that each event has a different time and room combination, ensuring that no room will have more than one event at any time.

$$alldifferent(Timeslot) \tag{3}$$

Soft Constraints

All the soft constraints are formulated in such a way that they output a variable or expression with the number of points or constraint violations. They are included in the objective, which the model is instructed to minimize. For each soft constraint, the number of constraint violations will be referred to as Points (x), where x is the constraint number.

(4) There should be no events in the last period of a day. In CP 1, the constraint is formulated as follows.

$$Point\ s(4) = \sum_{e \in E} size_e \cdot LastPeriod_e \tag{4}$$

(5) No student should have more than two consecutive events. To find the number of times there is a student with more than two classes in a row, each student's schedule is stepped through one day at a time using a three period window, and the number of times that a student has three events in a row is counted and added to the total number of *Points*(5), violations of the fifth constraint.

Two global cardinality constraints are used. The first global cardinality constraint counts the number of events in each three period span. The values are stored in a variable array called *Count_Events*, one for each student.

$$\forall s \in S, \forall d \in D, \forall p \rightarrow p+2 \in P, gcc(Count\_Events, 1, Schedule_s[d][p])^5 \quad (5)$$

$Count\_Events_s[i]$, where i goes from 0 through 3, is the number of times that student s has i events in a three-period span. The second global cardinality constraint extracts the number of times that the student had three events in a row from $Count\_Events_s$. The result, $Count\_Three_s$ contains the total number of times that the student had three events in a row.

$$\forall s \in S, gcc(Count\_Three_s, 3, Count\_Events_s) \quad (6)$$

The final step is to add the count for each student to the total number of constraint violations.

$$Point\ s(5) = \sum_{s \in S} Count\_Three_s \quad (7)$$

(6) No student should have a single event on a day. For each student, their events are looked at one by one to see how many are on each day. This is done using two global cardinality constraints. The first global cardinality constraint counts the number of events on each day. The values are stored in a variable array called *EventDayCount* , one for each student.

$$\forall s \in S, gcc(EventDayCount, [0,1,...,|D|], StudentEventDays_s) \quad (8)$$

$EventDayCount[i]$, where i goes from zero through |D|-1, contains the number of events that the student has on day i. The second global cardinality constraint goes through *EventDayCount* and counts how many variables took the value 1. The result, $Count\_One_s$ contains the number of times that student had a single event on a day.

$$\forall s \in S, gcc(Count\_One_s, 1, EventDayCount) \quad (9)$$

The final step is to add the count for each student to the total number of constraint violations.

$$Point\ s(6) = \sum_{s \in S} Count\_One_s \quad (10)$$

Objective function

The objective function is then to minimize all of the soft constraint violations.

---

[5] gcc(a,b,c) is the standard way of writing a global cardinality constraint. The constraint takes an array of variables, c, and it counts how many times each of the values in a given range, b, appears. It stores the count of how many times each of the values appear in an array, a.

$$Minimize\ (Points(4) + Points(5) + Points(6)) \tag{11}$$

7.4.2 CP Scheduling 1

CP Scheduling 1 is a monolithic CP model using scheduling constructs such as activities and resources.

Decision Variables

For each event, e ? E, we have two decision variables.

- t_e - It takes a value from 0 through |T|-1. It represents the time that the event is scheduled in.
- r_e – It takes a value from 0 through |R|-1. It represents the room that the event is scheduled in.

Set Up

The model is set up as a scheduling problem. The events are activities that have duration of one time unit. The students and rooms are unary capacity resources, meaning that they can only service one event at a time. Therefore, if two events require the same student or room, they will not be scheduled at the same time.

Constraints

Hard Constraints.

(1) In CP Scheduling 1, the students are represented as unary capacity resources. Therefore, if two events require the same student, they cannot be scheduled at the same time. This constraint is represented as follows.

$$\forall e \in E, \forall s \in attending_e, e.requires(s) \tag{12}$$

(2) Size and feature requirements must be respected. For each event, every room is checked, first for size and then for features. If a room is not large enough for the event or if it does not contain all the required features, that room is removed from the domain of r_e using a not-equals constraint.

$$\begin{aligned} &\forall e \in E, \forall r \in R,\ \ if\ (size_r < size_e) \\ &\vee (\exists f \in feature_e \not\subset features_r) \\ &then(r_e != r) \end{aligned} \tag{13}$$

In CP Scheduling 1, the rooms are set up as unary capacity resources. For this constraint, an alternate resource set is made for each room. The alternate resource set contains all the rooms that the event can be scheduled in while respecting the size and feature requirements; i.e. after rooms are removed from the domain of r_e as above. The event

is constrained to require its alternate resource set. During solving, the solver chooses one of the rooms from its set as r_e.

$$\forall e \in E, \forall r \in R, if\ (r \subset Domain(r_e)$$
$$then(r \rightarrow alternate\_resource\_set_e)$$
$$\wedge\ e.requires(alternate\_resource\_set_e) \quad (14)$$

(3) There can be no conflicts for rooms. This constraint is represented by representing the rooms as unary capacity resources.

Soft Constraints

All of the soft constraints are represented in the same way as in CP 1.

Objective function

The objective function is then to minimize all of the soft constraint violations as in equation (11).

7.4.3 IP 1

IP 1 is a monolithic MIP model.

Decision Variables

For each event, e ? E, we have one decision variable array.

- ts_e - It is an array of variables of size |TS|-1. Variable ts_e[i] takes the value 1 if event e is scheduled in timeslot i and is 0 otherwise.

Set Up

The model is set up using two linked variable arrays. The first is a 3D array of time periods by rooms by events.

$$\forall t \in T, \forall r \in R, \forall e \in E, EventsAssignments[t][r][e]$$

The domain is binary and the value is 1 if the event, e, is assigned to time t and room r, and is zero otherwise. The second is a 2D array of decision variables of timeslots by events. The values of the variables in *Timeslots* are equal to the corresponding time, room, and event combination in *EventAssignments.*

$\forall t \in T, \forall r \in R, \forall e \in E, Timeslots[ts][e] = EventAssignments[t][r][e]$, ts = t*|R| + r – 1.

Timeslots is linked to the decision variables in such a way that *Timeslots*[*i*][*e*] takes the same value as ts_e[i].

$$\forall i \in TS, \forall e \in E, Timeslots[i][e] = ts_e[i]$$

Constraints

Hard Constraints

(1) There can be no conflicts for students. In IP 1, there is a constraint on each time period, that each student cannot have more than one event at that time.

$$\forall s \in S, \forall t \in T, \sum_{e \in E_s} EventatTime_t[e] \leq 1 \quad (15)$$

$EventatTime_t$ is an array of auxiliary variables, the size of $|E_s|$ - 1. $EventatTime_t[e] = 1$ if $StudentEventTimes_s[e] = t$ and is zero otherwise.

(2) Size and feature requirements must be respected. For each event, every room is checked, first for size and then for features. If a room is not large enough for the event or if it does not contain all the required features, any timeslot representing that room is removed from the domain of ts_e using a constraint that sets the value of that variable to zero.

$$\begin{aligned} &\forall e \in E, \forall r \in R, \ if\ (size_r < size_e) \\ &\vee (\exists f \in feature_e \not\subset features_r) \\ &then(\forall t \in T, \forall i \in TS, if\ (i = t + |R| \cdot r), ts[i][e] == 0) \end{aligned} \quad (16)$$

(3) There can be no conflicts for rooms. This constraint states that no room can be used by more than one event at a time. In IP 1 it is a linear constraint, which makes sure that each room is used at most once for any time. It sums over the timeslot array for each event and ensures that at most one event is assigned to any given timeslot.

$$\forall i \in TS, \sum_{e=0}^{e=|E|-1} ts[i][e] \leq 1 \quad (17)$$

Soft Constraints

All the soft constraints are formulated in such a way that they output a variable or expression with the number of points or constraint violations. They are included in the objective, which the model is instructed to minimize. For each soft constraint, the number of constraint violations will be referred to as Points (x), where x is the constraint number.

(4) There should be no events in the last period of a day. This constraint is formulated as in equation (4).

(5) No student should have more than two consecutive events. To find the number of times there is a student with more than two classes in a row, each student's schedule is stepped through one day at a time using a three period window, and the number of times that a student has three events in a row is counted and added to the total number of *Points*(5), violations of the fifth constraint. For IP 1, a logical, yet linear constraint is used.

$$\forall s \in S, Count\_Three_s = \sum_{d \in D} \sum_{p=0}^{|P|-3} (ScheduleThree_s[d][p]) \quad (18)$$

ScheduleThree is a 2D array of auxiliary variables of the size |D| by |P|-3. *ScheduleThree*[d][p]= 1 if $Schedule_s[d][p] = Schedule_s[d][p+1] = Schedule_s[d][p+2] = 1$ and is 0 otherwise. The result, $Count\_Three_s$ contains the total number of times that the student had three events in a row. The final step is to add the count for each student to the total number of constraint violations as in equation (7).

(6) No student should have a single event on a day. For each student, their events are looked at one by one to see how many are on each day. This is done using a logical, yet linear constraint.

$$\forall d \in D, \forall s \in S, Count\_One_s = \sum_{d} \sum_{e \in E_s} OneEventDay \quad (19)$$

OneEventDay is a 2D array of auxiliary variables of the size |D| by $|E_s|$-1. *OneEventDay*[d][e] = 1 if $StudentEventDays_s[d][e] = 1$ and is 0 otherwise. The result, $Count\_One_s$ contains the number of times that student had a single event on a day. The final step is to add the count for each student to the total number of constraint violations as in equation (10).

Objective function

The objective function is then to minimize all of the soft constraint violations as in equation (11).

7.5 The Decomposition Models

The next three models are decomposition models. They each contain two sub-models designed to solve the problem described in section 7.2. The interaction is the same for all of the decomposition models. The first sub-model is referred to as the master problem or the time model. It assigns events to times. It takes into account the hard constraint of students not having conflicts as well as all three soft constraints and the objective function that says that the soft constraint violations should be minimized. The second sub-model, referred to as the sub-problem or room model, assigns the events in each time to rooms. It takes into account the two hard constraints involving the rooms, respecting size and feature requirements as well as room conflicts.

The first sub-model is solved. Then, for each time period independently, the room model is created to assign the events at that time to rooms. If this cannot be done without violating the hard constraints, a cut is created. Once all the time periods have been passed through the room model, the collection of resulting cuts is added to the time model and the process repeats.

In order for this decomposition to be as efficient as possible, there are two things to consider. The first is the representation of a relaxation of the room model in the time model and the second is the quality and type of cut being created by the room model [87]. In the models discussed, several ideas were attempted for both of these points.

The representation of the room model in the time model is a constraint that the total number of events assigned to any time period cannot exceed the number of rooms available.

$$\forall t \in T, |Events_t| \leq |R|$$

It is a relaxation of the room constraints. There is also a constraint that ensures that the sum of the sizes of events assigned to any time period cannot exceed the sum of the capacities of all the rooms.

$$\forall t \in T, \sum_{e \in Events_t} size_e \leq \sum_{r \in R} size_r$$

The cut passed from the room model into the time model is actually a series of constraints. Any time the room model cannot find a solution, the set of events assigned to that time period are analyzed by looking at every combination of that set of events, from combinations of two up to the full set of events less one. The set of combinations is referred to as *EventCombos*. Any time that the sum of the event sizes is greater that the sum of the x biggest room capacities, x being the number of events in the given combination, or the sum of any feature requirement is greater than the number of that feature available in the rooms, a cut is created. The cut states that the set of events in the given combination cannot be assigned to the same time period.

$$\forall ec \in EventCombos, if\ (\sum size_e > \sum size_r \vee \forall f \in F, \sum feature_e > \sum feature_r)$$
$$then, \forall e1 \in ec, \forall e2 \in ec, e1 \neq e2, EventTime[e1] \neq EventTime[e2]$$

7.5.1 CP 2

CP 2 is a decomposition CP model. Both of the sub-models are modeled in CP.

Decision Variables

For each event, e ? E, we have two decision variables. The first decision variable is in the time model.

- t_e - It takes a value from 0 through |T|-1. It represents the time at which the event is scheduled.

The second decision variable is in the room model. There is therefore a separate set of decision variables for each time period.

- r_{et} – It takes a value from 0 through |R|-1. It represents the room in which the event is scheduled.

Set Up

The time model is set up using two linked variable arrays. The first is a 2D array of days by periods.

$$\forall d \in D, \forall p \in P, EventsAssignments[d][p]$$

The domain of each variable is the set of events and the value is the event assigned to day d, and period p. The second array is an array of times, the decision variables.

$$\forall d \in D, \forall p \in P, TimeAssignments[t] = EventsAssignments[d][p]$$

TimeAssignments[*t*] is the variable representing the event that is scheduled at time t. This value is also represented in the 2D array where *EventsAssignments*[*d*][*p*] is the event scheduled on day d, and at period p. The *TimeAssignments* variables are linked to the decision variables in such a way that *TimeAssignments*[*t*] has the value e, while t_e has the value t.

$$\forall e \in E, \exists t \in T, TimeAssignments[t] = e \leftrightarrow t_e = ts$$

Constraints

Hard Constraints

(1) There can be no conflicts for students. This constraint is an all-different constraint on the start times on each student's events as in equation (1). It appears in the time model.

(2) Size and feature requirements must be respected. This constraint is represented in the room model. For each event, every room is checked, first for size and then for features. If a room is not large enough for the event or if it does not contain all the required features, that room is removed from the domain of r_e using a not-equals constraint.

$$\begin{aligned} &\forall e \in E, \forall r \in R, \ \ if\ (size_r < size_e) \\ &\vee (\exists f \in feature_e \not\subset features_r) \\ &then (\forall t \in T, r_{et} != r) \end{aligned} \tag{20}$$

(3) There can be no conflicts for rooms. This constraint states that no room can be used by more than one event at a time. In CP 2 it is represented in the room model using an all-different constraint on the array of room choices for the set of events assigned to each time.

$$\forall t \in T, alldifferent(r_t) \tag{21}$$

Soft Constraints

All of the soft constraints are in the time model and are modeled in the same way as in CP 1.

Objective function

The objective function is in the time model and is to minimize all of the soft constraint violations as in equation (11).

7.5.2 CP Scheduling 2

CP Scheduling 2 is a decomposition CP model. The time model is modeled as a scheduling problem and the room model is the same as in CP 2.

Decision Variables

For each event, e ? E, we have two decision variables. The first decision variable is in the time model.

- t_e - It takes a value from 0 through |T|-1. It represents the time at which the event is scheduled.

The second decision variable is in the room model. There is therefore a separate set of decision variables for each time period.

- r_{et} – It takes a value from 0 through |R|-1. It represents the room that the event is scheduled in.

Set Up

The time model is set up as a scheduling problem. The events are activities that have duration of one time unit. The students are unary capacity resources, meaning that they can only service one event at a time. Therefore if two events require the same student, they will not be scheduled at the same time.

Constraints

Hard Constraints

(1) In CP Scheduling 2, the students are represented as unary capacity resources. Therefore, if two events require the same student, they cannot be scheduled at the same time. This constraint is therefore represented as in equation (12).

(2) Size and feature requirements must be respected. This constraint is represented in the room model. For each event, every room is checked, first for size and then for features. If a room is not large enough for the event or if it does not contain all the required features, any timeslot representing that room is removed from the domain of r_e as in equation (20).

(3) There can be no conflicts for rooms. This constraint is in the room model and is represented as in equation (21).

Soft Constraints

All of the soft constraints are in the time model and are modeled in the same way as in CP 1.

Objective function

The objective function is in the time model and is to minimize all of the soft constraint violations as in equation (11).

7.5.3 IP 2

IP 2 is decomposition MIP/CP model. The time model is a MIP model and the room model is the same as in CP 2.

Decision Variables

For each event, e ? E, we have two decision variable arrays. The first is in the time model.

- t_e - It is an array of variables of size |T|-1. Variable i takes the value 1 if event e is scheduled at time i and is 0 otherwise.

The second array is in the room model. There is therefore a separate set of decision variables for each time period.

- r_{et} – It is an array of variables of size |R|-1. Variable i takes the value 1 if event e is scheduled in room i and is 0 otherwise.

Constraints

Hard Constraints

(1) There can be no conflicts for students. In IP 2, there is a constraint on each time period, that each student cannot have more than one event at that time as in equation (15).

(2) Size and feature requirements must be respected. This constraint is represented in the room model. For each event, every room is checked, first for size and then for features. If a room is not large enough for the event or if it does not contain all the required features, any timeslot representing that room is removed from the domain of r_e as in equation (20).

(3) There can be no conflicts for rooms. This constraint states that no room can be used by more than one event at a time. This constraint is in the room model and is represented as in equation (21).

Soft Constraints

All of the soft constraints are in the time model and are modeled in the same way as in IP 1.

Objective function

The objective function is in the time model and is to minimize all of the soft constraint violations as in equation (11).

7.6 Experiments

The models were tested using a set of 21 problems. They were all of the form described in section 7.2. The first was a medium-sized problem created by Ben Paechter [78]. We created the other 20 by scaling down the 20 problem instances used in the international timetabling competition [75]. They were scaled down to match the size of Ben Paechter's medium-sized problem since the actual competition instances were too large for any of the models, excluding IP 1, to solve in a reasonable amount of time, in this case a three-hour period.

The problem instances all have 100 events, 80 students, 10 rooms, and 5 features. The 20 scaled down problems were created by selecting the 100 events, the first 80 students, the first 10 rooms, and the first 5 features from each of the 20 large problem instances from the competition.

The models were tested on each of the 21 problem instances eight times.

- To find any feasible solution, ignoring all soft constraints.
- To find an optimal solution considering all the hard constraints and only the first soft constraint, constraint (4).
- To find an optimal solution considering all the hard constraints and only the second soft constraint, constraint (5).
- To find an optimal solution considering all the hard constraints and only the third soft constraint, constraint (6).
- To find an optimal solution considering all the hard constraints and only the first two soft constraints, constraints (4) and (5).
- To find an optimal solution considering all the hard constraints and only the first and third soft constraints, constraints (4) and (6).
- To find an optimal solution considering all the hard constraints and only the second two soft constraints, constraints (5) and (6).
- To find an optimal solution considering all the hard constraints and all the soft constraints.
-

All of the experiments were run on a 2.8 GHz Pentium 4 with 512 Mb RAM running Fedora Core 2 and were implemented using the ILOG Optimization Suite as described in Section 7.3. In the following sections, the results from all the experiments will be documented.

7.6.1 Satisfaction Experiments

There were no soft constraints used during the satisfaction experiments. Each of the 21 problems was run on each of the six models without taking into account any of the soft constraints. As soon as a feasible solution was found (i.e. a solution that did not violate any hard constraints) the time was recorded. The entries in the table refer to the time in seconds it took to find the solution. A '–' means that no solution was found in the allotted three hours. Problem 16 has no solution. It was created that way so that the time to see that no

solution was possible could be recorded. The bold entries show the best time to solve for each problem instance.

Table 2. Satisfaction Experiment Results. The best time to solve for each problem instance is in bold.

Problem Instance	CP 1	CP Scheduling 1	IP 1	CP 2	CP Scheduling 2	IP 2
0	1.75	**0.13**	0.63	1.78	0.21	-
1	13.98	**0.07**	1.19	0.73	0.35	0.14
2	14.46	**0.08**	1.2	0.42	36.63	0.13
3	15.03	**0.09**	1.36	8.56	163.6	-
4	13.38	**0.07**	1.05	75.41	-	-
5	14.53	**0.1**	1.23	1.27	132.1	-
6	14.78	**0.08**	1.01	8.5	2003.4	0.13
7	14.25	**0.07**	1.06	0.77	169.4	-
8	13.87	**0.07**	1.09	3.63	-	-
9	14.06	**0.06**	1.1	231.44	26.04	-
10	14.42	**0.08**	1.02	759.9	-	-
11	13.35	**0.07**	0.93	16.06	-	-
12	14.42	**0.05**	1.09	106.28	-	-
13	14.41	**0.07**	1.24	40.48	-	-
14	13.94	**0.1**	1.17	1.14	-	-
15	14.31	**0.09**	1.19	28.7	-	-
16	**0**	**0**	**0**	-	-	-
17	13.89	**0.07**	1.06	38.57	-	-
18	13.15	**0.07**	1.13	3.78	0.03	0.14
19	14.66	**0.06**	1.14	12.86	-	-
20	13.26	**0.1**	1.26	16.9	602.4	-

CP Scheduling 1 has the fastest time to solve for all of the problem instances. It is followed closely by IP 1. CP Scheduling 2 and IP 2 perform the worst and are unable to solve many of the problem instances in the allotted three hours. All three monolithic models are able to discover that problem 16 has no solution very quickly, while the decomposition models are not able to.

7.6.2 Optimization Experiments

Experiments were run for every combination of soft constraints as well as for each soft constraint on its own. Each of the 21 problems was run on each of the six models. The results are tabulated separately for the monolithic and the decomposition models.

Monolithic Models

For the monolithic models, the time to solve as well as the number of violations of the soft constraint(s) being minimized is recorded. The time limit for all the experiments was three

hours (10800 seconds), at which point the best solution found so far was recorded. A '-' means that no feasible solution was found in the allotted time. The following tables summarize the results. It should be noted that I could only output the solution from IP1 if it was optimal, so in the cases where it shows that no solution was found, it may have actually found feasible solutions.

Table 3a. Optimization results for soft constraint (4), events in the last period.

Problem Instance	CP 1 time	CP 1 violations	CP Sched.1 time	CP Sched. 1 violations	IP 1 time	IP 1 violations
0	20.1	0	10800	-	**0.8**	**0**
1	10800	4	10800	9	**1.79**	**0**
2	10800	5	10800	-	**1.56**	**0**
3	10800	3	10800	6	**1.62**	**0**
4	10800	5	10800	8	**1.42**	**0**
5	10800	6	10800	-	**1.76**	**0**
6	10800	3	10800	5	**1.86**	**0**
7	10800	5	10800	8	**1.58**	**0**
8	10800	1	10800	-	**1.5**	**0**
9	10800	1	10800	2	**1.55**	**0**
10	10800	3	10800	-	**1.61**	**0**
11	10800	4	10800	7	**1.41**	**0**
12	10800	3	10800	3	**1.26**	**0**
13	10800	6	10800	6	**1.52**	**0**
14	10800	5	10800	-	**1.62**	**0**
15	10800	2	10800	9	**1.5**	**0**
16	**0**	**N/a**	**0**	**N/a**	**0**	**N/a**
17	10800	5	10800	-	**1.37**	**0**
18	10800	2	10800	5	**1.59**	**0**
19	10800	2	10800	6	**1.67**	**0**
20	10800	1	10800	8	**1.82**	**0**

IP 1 is the best. It solves all the problem instances to a solution without constraint violations quickly. CP Scheduling 1 is the worst since it cannot solve all the instances.

Table 3b.Optimization results for soft constraint (5), three events in a row.

Problem Instance	CP 1 time	CP 1 violations	CP Sched.1 time	CP Sched. 1 violations	IP 1 time	IP 1 violations
0	**143**	**0**	10800	-	497.2	0
1	375.42	0	10800	3	**111.72**	**0**
2	621	0	10800	2	**63.19**	**0**
3	846.7	0	10800	8	**36.9**	**0**
4	239.8	0	**1.16**	**0**	18.01	0
5	776.2	0	10800	1	**179.7**	**0**
6	1342.5	0	10800	23	**159.2**	**0**
7	665.4	0	10800	10	**36.22**	**0**
8	530.1	0	10800	4	**42.3**	**0**
9	920.9	0	10800	18	**76.79**	**0**
10	547.3	0	10800	-	**143.2**	**0**
11	517.3	0	10800	13	**49.51**	**0**
12	465.37	0	10800	36	**7.07**	**0**
13	749.8	0	10800	14	**23.48**	**0**
14	716	0	10800	-	**158.2**	**0**
15	673.3	0	10800	5	**45.58**	**0**
16	**0**	**N/a**	**0**	**N/a**	**0**	**N/a**
17	501.57	0	10800	13	**79.91**	**0**
18	814.6	0	10800	21	**37.08**	**0**
19	517.9	0	10800	21	**75.06**	**0**
20	946.4	0	10800	4	**593.9**	**0**

IP 1 was the best in every case except for problem instance 0 when CP 1 was the best and for problem instance 4, when CP Scheduling 1 was the best, but both IP 1 and CP 1 were able to find solutions without soft constraint violations for every model.

Table 3c. Optimization results for soft constraint (6), a single event on a day.

Problem Instance	CP 1 time	CP 1 violations	CP Sched.1 time	CP Sched. 1 violations	IP 1 time	IP 1 violations
0	10800	29	10800	**-**	**1370**	**0**
1	**10800**	**31**	10800	-	10800	-
2	**10800**	**39**	10800	-	10800	-
3	**10800**	**38**	10800	119	10800	-
4	**10800**	**42**	10800	141	10800	-
5	**10800**	**20**	10800	-	10800	-
6	**10800**	**33**	10800	110	10800	-
7	**10800**	**60**	10800	79	10800	-
8	**10800**	**35**	10800	-	10800	-
9	**10800**	**38**	10800	88	10800	-
10	**10800**	**37**	10800	-	10800	-
11	**10800**	**42**	10800	101	10800	-
12	10800	53	**10800**	**36**	10800	-
13	**10800**	**37**	10800	115	10800	-
14	**10800**	**31**	10800	-	10800	-
15	**10800**	**15**	10800	124	10800	-
16	**0**	**N/a**	**0**	**N/a**	**0**	**N/a**
17	**10800**	**41**	10800	130	10800	-
18	**10800**	**35**	10800	98	10800	-
19	**10800**	**38**	10800	71	10800	-
20	**10800**	**43**	10800	-	10800	-

In almost all cases, the models were run until the three-hour time limit was reached. CP 1 was the best in all cases except for problem instance 1, where IP 1 was able to find the optimal solution in less than the allotted three hours, and problem instance 12 where CP Scheduling 1 was able to find a solution with less soft constraint violations. However, I could only output the solution from IP1 if it was optimal, so it may have found feasible solutions.

Table 3d. Optimization results for soft constraint (4) and (5).

Problem Instance	CP 1 time	CP 1 violations	CP Sched.1 time	CP Sched. 1 violations	IP 1 time	IP 1 violations
0	**174.6**	**0**	10800	-	1176	0
1	10800	3	10800	-	**85.86**	**0**
2	10800	5	10800	-	**80.38**	**0**
3	10800	1	10800	-	**20.97**	**0**
4	10800	4	10800	12	**15.06**	**0**
5	10800	6	10800	-	**131.6**	**0**
6	10800	3	10800	11	**122.18**	**0**
7	10800	3	10800	17	**32**	**0**
8	10800	1	10800	-	**40.46**	**0**
9	10800	1	10800	9	**49.6**	**0**
10	10800	5	10800	-	**99.47**	**0**
11	10800	4	10800	36	**27.88**	**0**
12	10800	3	10800	38	**8.67**	**0**
13	10800	5	10800	20	**31.46**	**0**
14	10800	5	10800	-	**131.2**	**0**
15	10800	2	10800	9	**41.21**	**0**
16	**0**	**N/a**	**0**	**N/a**	**0**	**N/a**
17	10800	5	10800	-	**44.79**	**0**
18	10800	2	10800	30	**24.62**	**0**
19	10800	2	10800	22	**71.36**	**0**
20	10800	1	10800	-	**383.1**	**0**

IP 1 was the best in every case except for problem instance 0 when CP 1 was the best. CP Scheduling 1 was worst since it could not find solutions to many of the problem instances.

Table 3e. Optimization results for soft constraint (4) and (6).

Problem Instance	CP 1 time	CP 1 violations	CP Sched.1 time	CP Sched. 1 violations	IP 1 time	IP 1 violations
0	**10800**	**37**	10800	-	10800	-
1	**10800**	**36**	10800	-	10800	-
2	**10800**	**45**	10800	-	10800	-
3	**10800**	**43**	10800	82	10800	-
4	**10800**	**48**	10800	-	10800	-
5	**10800**	**26**	10800	-	10800	-
6	**10800**	**41**	10800	99	10800	-
7	**10800**	**67**	10800	53	10800	-
8	**10800**	**39**	10800	-	10800	-
9	**10800**	**44**	10800	83	10800	-
10	**10800**	**44**	10800	-	10800	-
11	**10800**	**47**	10800	80	10800	-
12	10800	61	**10800**	**47**	10800	-
13	**10800**	**44**	10800	84	10800	-
14	**10800**	**38**	10800	-	10800	-
15	**10800**	**18**	10800	76	10800	-
16	**0**	**N/a**	**0**	**N/a**	**0**	**N/a**
17	**10800**	**47**	10800	-	10800	-
18	**10800**	**40**	10800	86	10800	-
19	**10800**	**46**	10800	74	10800	-
20	**10800**	**49**	10800	-	10800	-

In all cases, the models were run until the three-hour time limit was reached. CP 1 was the best in all cases except for problem instance 12 where CP Scheduling 1 was able to find a solution with less soft constraint violations.

Table 3f. Optimization results for soft constraint (5) and (6).

Problem Instance	CP 1 time	CP 1 violations	CP Sched.1 time	CP Sched. 1 violations	IP 1 time	IP 1 violations
0	**10800**	**-**	**10800**	**-**	**10800**	**-**
1	**10800**	**50**	10800	-	10800	-
2	**10800**	**80**	10800	-	10800	-
3	**10800**	**90**	10800	127	10800	-
4	**10800**	**59**	10800	142	10800	-
5	**10800**	**76**	10800	-	10800	-
6	10800	-	**10800**	**133**	10800	-
7	10800	91	**10800**	**89**	10800	-
8	**10800**	**66**	10800	-	10800	-
9	10800	-	**10800**	**106**	10800	-
10	**10800**	**65**	10800	-	10800	-
11	**10800**	**71**	10800	114	10800	-
12	10800	-	**10800**	**72**	10800	-
13	**10800**	**85**	10800	129	10800	-
14	**10800**	**-**	**10800**	**-**	**10800**	**-**
15	**10800**	**64**	10800	147	10800	-
16	**0**	**N/a**	**0**	**N/a**	**0**	**N/a**
17	10800	-	**10800**	**143**	10800	-
18	10800	-	**10800**	**119**	10800	-
19	10800	-	**10800**	**92**	10800	-
20	**10800**	**106**	10800	-	10800	-

In all cases, the models were run until the three-hour time limit was reached. CP 1 was the best in most cases, although there were several instances where CP 1 was not able to find a solution and CP Scheduling was. IP 1 was not able to find solutions to any of the instances. However, I could only output the solution from IP1 if it was optimal, so it may have found feasible solutions.

Table 3g. Optimization results for soft constraint (4), (5), and (6).

Problem Instance	CP 1 time	CP 1 violations	CP Sched.1 time	CP Sched. 1 violations	IP 1 time	IP 1 violations
0	**10800**	**-**	**10800**	**-**	**10800**	**-**
1	**10800**	**55**	10800	-	10800	-
2	**10800**	**86**	10800	-	10800	-
3	**10800**	**95**	10800	142	10800	-
4	**10800**	**65**	10800	-	10800	-
5	**10800**	**82**	10800	-	10800	-
6	10800	-	**10800**	**134**	10800	-
7	10800	98	**10800**	**64**	10800	-
8	**10800**	**70**	10800	-	10800	-
9	10800	-	**10800**	**94**	10800	-
10	**10800**	**72**	10800	-	10800	-
11	**10800**	**76**	10800	109	10800	-
12	10800	-	**10800**	**98**	10800	-
13	**10800**	**92**	10800	107	10800	-
14	**10800**	**-**	**10800**	**-**	**10800**	**-**
15	**10800**	**67**	10800	97	10800	-
16	**0**	**N/a**	**0**	**N/a**	**0**	**N/a**
17	**10800**	**-**	**10800**	**-**	**10800**	**-**
18	10800	-	**10800**	**105**	10800	-
19	10800	-	**10800**	**91**	10800	-
20	**10800**	**112**	10800	-	10800	-

In all cases, the models were run until the three-hour time limit was reached. CP 1 was the best in most cases, although there were several instances where CP 1 was not able to find a solution and CP Scheduling was. IP 1 was not able to find solutions to any of the instances. However, I could only output the solution from IP1 if it was optimal, so it may have found feasible solutions.

Decomposition Models

For the decomposition models, the time to solve, the number of violations, and feasibility status is recorded. If the solution found is not feasible, it only solves the first sub-model, a '*' appears next to the number of violations. If the entry for violations is a '-', it means that no feasible solution was found in the allotted time of three hours. The following tables summarize the results. It should be noted that the IP model in IP2 could only put out a solution if it was optimal.

Table 4a. Optimization results for soft constraint (4), events in the last period.

Problem Instance	CP 2 time	CP 2 violations	CP Sched.2 time	CP Sched. 2 violations	IP 2 time	IP 2 violations
0	3.87	0	**2.55**	**0**	10800	0*
1	10800	4	10800	10	**3.87**	**0**
2	10800	7	10800	10	**3.72**	**0**
3	**10800**	**6**	10800	10	10800	0*
4	**10800**	**10**	10800	-	10800	0*
5	**10800**	**4**	10800	10	10800	0*
6	10800	2	10800	10	**3.79**	**0**
7	**10800**	**5**	10800	-	10800	0*
8	**10800**	**4**	10800	-	10800	0*
9	**10800**	**1**	10800	-	10800	0*
10	**10800**	**6**	10800	-	10800	0*
11	**10800**	**6**	10800	-	10800	0*
12	**10800**	**7**	10800	-	10800	0*
13	**10800**	**7**	10800	-	10800	0*
14	**10800**	**5**	10800	-	10800	0*
15	**10800**	**4**	10800	-	10800	0*
16	**10800**	**-**	**10800**	**-**	**10800**	**-**
17	**10800**	**8**	10800	-	10800	0*
18	4.93	0	10800	-	**3.74**	**0**
19	**10800**	**1**	10800	-	10800	0*
20	**10800**	**2**	10800	-	10800	0*

In most of the cases, CP 2 was the best, as it was the only model that was able to find feasible solutions to all the problem instances. In the few cases where IP 2 did find a feasible solution, it found a better one faster than CP 2. CP Scheduling 2 was not able to find solutions to many of the problems.

Table 4b.Optimization results for soft constraint (5), three events in a row.

Problem Instance	CP 2 time	CP 2 violations	CP Sched.2 time	CP Sched. 2 violations	IP 2 time	IP 2 violations
0	**129.7**	**0**	10800	10	10800	0*
1	**3.21**	**0**	855.23	0	10800	0*
2	100.9	0	7481.5	0	**1.11**	**0**
3	**24.49**	**0**	10800	-	10800	0*
4	**110.67**	**0**	10800	-	10800	0*
5	**53.38**	**0**	1089.6	0	10800	0*
6	97.7	0	950.6	0	**1.17**	**0**
7	**23.73**	**0**	10800	-	10800	0*
8	41.18	0	10800	-	**1.08**	**0**
9	10800	-	9162.4	0	**1.13**	**0**
10	**1587**	**0**	10800	-	10800	0*
11	**56.12**	**0**	10800	-	10800	0*
12	**354.41**	**0**	10800	-	10800	0*
13	**138.54**	**0**	10800	-	10800	0*
14	**21.01**	**0**	10800	-	10800	0*
15	**171.96**	**0**	10800	-	10800	0*
16	**10800**	**-**	**10800**	**-**	**10800**	**-**
17	58.96	0	10800	-	**1.2**	**0**
18	23.36	0	616.39	0	**1.12**	**0**
19	**31.66**	**0**	10800	-	10800	0*
20	**91.12**	**0**	10800	-	10800	0*

In most of the cases, CP 2 was the best, as it was the only model that was able to find solutions to all the problem instances. In the few cases where IP 2 did find a solution, it found a better one faster than CP 2. CP Scheduling 2 was not able to find solutions to many of the problems.

Table 4c. Optimization results for soft constraint (6), a single event on a day.

Problem Instance	CP 2 time	CP 2 violations	CP Sched.2 time	CP Sched. 2 violations	IP 2 time	IP 2 violations
0	10800	-	10800	-	**3643.6**	**0**
1	**10800**	-	**10800**	-	**10800**	-
2	**10800**	**106***	10800	-	10800	-
3	**10800**	**116***	10800	-	10800	-
4	**10800**	**105***	10800	-	10800	-
5	**10800**	**74***	10800	-	10800	-
6	**10800**	-	**10800**	-	**10800**	-
7	10800	56*	10800	-	**10800**	**1***
8	10800	92*	10800	-	**10800**	**5***
9	**10800**	**107***	10800	-	10800	-
10	**10800**	**80***	10800	-	10800	-
11	10800	-	10800	-	**10800**	**4***
12	10800	97*	10800	-	**10800**	**2***
13	**10800**	**119***	10800	-	10800	-
14	**10800**	-	**10800**	-	**10800**	-
15	**10800**	**83***	10800	-	10800	-
16	**10800**	-	**10800**	-	**10800**	-
17	**10800**	**88***	10800	-	10800	-
18	**10800**	-	**10800**	-	**10800**	-
19	**10800**	-	**10800**	-	**10800**	-
20	**10800**	-	**10800**	-	**10800**	-

In almost all the cases, the models were run until the three-hour time limit. For problem instance 0, IP 2 was able to find a feasible solution without constraint violations. CP 2 and IP 2 were able to find partial solutions to some of the problems and CP Scheduling 2 was not able to find any solutions.

Table 4d.Optimization results for soft constraint (4) and (5).

Problem Instance	CP 2 time	CP 2 violations	CP Sched.2 time	CP Sched. 2 violations	IP 2 time	IP 2 violations
0	**170.4**	**0**	10800	24	10800	0*
1	10800	1	10800	7	**8.07**	**0**
2	10800	6	10800	-	**5.92**	**0**
3	**10800**	**3**	10800	-	10800	0*
4	**10800**	**6**	10800	-	10800	0*
5	**10800**	**1**	10800	-	10800	0*
6	155.5	0	10800	-	**6.78**	**0**
7	**10800**	**4**	10800	-	10800	0*
8	10800	3	10800	-	**7.74**	**0**
9	10800	-	10800	-	**10800**	**0***
10	**10800**	**5**	10800	-	10800	0*
11	**10800**	**6**	10800	-	10800	0*
12	**10800**	**6**	10800	-	10800	0*
13	**10800**	**7**	10800	-	10800	0*
14	10800	3	10800	-	**8.12**	**0**
15	**10800**	**5**	10800	-	10800	0*
16	**10800**	**-**	**10800**	**-**	**10800**	**-**
17	**10800**	**4**	10800	-	10800	0*
18	29.13	0	10800	-	**8.6**	**0**
19	**10800**	**1**	10800	-	10800	0*
20	**120.17**	**0**	10800	-	10800	0*

In most of the cases, CP 2 was the best, as it was able to find feasible solutions to almost all of the problem instances. In the few cases where IP 2 did find a feasible solution, it found a better one faster than CP 2. CP Scheduling 2 was not able to find solutions to many of the problems.

Table 4e. Optimization results for soft constraint (4) and (6).

Problem Instance	CP 2 time	CP 2 violations	CP Sched.2 time	CP Sched. 2 violations	IP 2 time	IP 2 violations
0	**10800**	**-**	**10800**	**-**	**10800**	**-**
1	**10800**	**-**	**10800**	**-**	**10800**	**-**
2	**10800**	**106***	10800	-	10800	-
3	**10800**	**116***	10800	-	10800	-
4	**10800**	**105***	10800	-	10800	-
5	**10800**	**74***	10800	-	10800	-
6	**10800**	**88***	**10800**	**-**	**10800**	**-**
7	10800	56*	10800	-	**10800**	**1***
8	10800	92*	10800	-	**10800**	**5**
9	**10800**	**-**	**10800**	**-**	**10800**	**-**
10	**10800**	**-**	**10800**	**-**	**10800**	**-**
11	**10800**	**83***	10800	-	10800	-
12	**10800**	**-**	**10800**	**-**	**10800**	**-**
13	**10800**	**119***	10800	-	10800	-
14	**10800**	**-**	**10800**	**-**	**10800**	**-**
15	**10800**	**83***	10800	-	10800	-
16	**10800**	**-**	**10800**	**-**	**10800**	**-**
17	**10800**	**88***	10800	-	10800	-
18	**10800**	**-**	**10800**	**-**	**10800**	**-**
19	10800	-	10800	-	**10800**	**2**
20	**10800**	**-**	**10800**	**-**	**10800**	**-**

Almost all the models were run until the three-hour time limit. For problems 8 and 19, IP 2 was able to find a feasible solution. CP 2 and IP 2 were able to find partial solutions to some of the remaining models, but CP Scheduling 2 was not able to find any solutions.

Table 4f. Optimization results for soft constraint (5) and (6).

Problem Instance	CP 2 time	CP 2 violations	CP Sched.2 time	CP Sched. 2 violations	IP 2 time	IP 2 violations
0	**10800**	**-**	**10800**	**-**	**10800**	**-**
1	**10800**	**-**	**10800**	**-**	**10800**	**-**
2	**10800**	**106***	10800	-	10800	-
3	**10800**	**116***	10800	-	10800	-
4	**10800**	**105***	10800	-	10800	-
5	**10800**	**74***	10800	-	10800	-
6	10800	-	10800	-	**673.5**	**0**
7	10800	56*	10800	-	**10800**	**5***
8	**10800**	**92***	10800	-	10800	-
9	**10800**	**107***	10800	-	10800	-
10	**10800**	**80***	10800	-	10800	-
11	**10800**	**-**	**10800**	**-**	**10800**	**-**
12	**10800**	**97***	10800	-	10800	-
13	**10800**	**119***	10800	-	10800	-
14	**10800**	**-**	**10800**	**-**	**10800**	**-**
15	**10800**	**83***	10800	-	10800	-
16	**10800**	**-**	**10800**	**-**	**10800**	**-**
17	**10800**	**88***	10800	-	10800	-
18	**10800**	**-**	**10800**	**-**	**10800**	**-**
19	10800	-	10800	-	**10800**	**2***
20	**10800**	**-**	**10800**	**-**	**10800**	**-**

Almost all the models were run until the three-hour time limit. For problem 6, IP 2 was able to find a feasible solution. CP 2 and IP 2 were able to find partial solutions to some of the remaining models, but CP Scheduling 2 was not able to find any solutions.

Table 4g. Optimization results for soft constraint (4), (5), and (6).

Problem Instance	CP 2 time	CP 2 violations	CP Sched.2 time	CP Sched. 2 violations	IP 2 time	IP 2 violations
0	10800	-	10800	-	10800	-
1	10800	-	10800	-	10800	-
2	10800	-	10800	-	10800	-
3	10800	-	10800	-	10800	-
4	10800	-	10800	-	10800	-
5	10800	-	10800	-	10800	-
6	10800	-	10800	-	10800	-
7	10800	-	10800	-	10800	-
8	10800	-	10800	-	10800	-
9	10800	-	10800	-	10800	-
10	10800	-	10800	-	10800	-
11	10800	-	10800	-	10800	-
12	10800	-	10800	-	10800	-
13	10800	-	10800	-	10800	-
14	10800	-	10800	-	10800	-
15	10800	-	10800	-	10800	-
16	10800	-	10800	-	10800	-
17	10800	-	10800	-	10800	-
18	10800	-	10800	-	10800	-
19	10800	-	10800	-	10800	-
20	10800	-	10800	-	10800	-

All of the models were run until the time limit and none of them could find any solutions to any of the models.

7.7 Discussion

Below is an aggregate table of the results in the previous section. The average time for each of the models, for each experiment is calculated. As well, for the optimization experiments, the number of solutions (out of 21 possible) that were feasible and the number of solutions that had the best score (i.e. the least amount of soft constraint violations). It should be noted that I could only output the solution from the IP models if it was optimal, so there may be cases where sub-optimal, yet feasible solutions were found.

Table 5. Aggregate Experimental Results. The best (i.e. a combination of the best time and the most solved) model for each experiment is in bold.

Included Constraints	CP 1	CP Scheduling 1	IP 1	CP 2	CP Scheduling 2	IP 2
1,2,3						
avg. time	11.53	**0.075**	1.05	578.9	5805.1	8742.9
# solved	21	**21**	21	20	10	4
1,2,3,4						
avg. time	9772.4	10285.7	**1.47**	9771.8	10285.8	8743.6
# feas.	21	13	**21**	20	6	4
# best	2	1	**21**	2	1	4
1,2,3,5						
avg. time	582.9	9771.5	**115.9**	1177.1	8674.1	7714.6
# feas.	21	17	**21**	19	7	6
# best	21	2	**21**	19	6	6
1,2,3,6						
avg. time	**10285.7**	10285.7	9836.7	10800	10800	10459.2
# feas.	**21**	13	2	0	0	1
# best	**1**	1	2	0	0	1
1,2,3,4,5						
avg. time	9779.7	10285.7	**124.66**	9274	10800	7716.4
# feas.	21	11	**21**	19	2	6
# best	2	1	**21**	4	0	6
1,2,3,4,6						
avg. time	**10285.7**	10285.7	10285.7	10800	10800	10800
# feas.	**21**	11	1	0	0	2
# best	**1**	1	1	0	0	0
1,2,3,5,6						
avg. time	**10285.7**	**10285.7**	10285.7	10800	10800	10317.8
# feas.	**13**	**13**	1	0	0	1
# best	**1**	**1**	1	0	0	1
1,2,3,4,5,6						
avg. time	**10285.7**	**10285.7**	10285.7	10800	10800	10800
# feas.	**13**	**11**	1	0	0	0
# best	**1**	**1**	1	0	0	0

The results seem to show that, in general, the monolithic models work better than the decomposition models and that the pure IP model is the best, except for when constraint 6 is introduced. In the cases where soft constraint 6 is included, CP 1 is the best. However, I could only output the solution from IP1 if it was optimal, so it may have found feasible solutions that were better than those from CP1. It should be noted that for the 21 test problems, the optimal solution is not known, so it is unclear how far from the optimal any of the solutions are. In the cases where a solution with a value of zero was found, it is known

that the optimal solution is zero. It should also be noted, that this research began as CP research. First came the pure CP models and it was because they did not work well on the larger competition problems that decomposition models were introduced as well. After observing the performance of the decomposition IP/CP model, a pure IP model was developed for the purpose of comparison. Before the introduction of the pure IP model, it was the CP decomposition model that had the most success with the competition instances. The following table displays the results when solving for any feasible solution, ignoring all the soft constraints, on the 20 competition instances. The entries in the table are as follows: A time in seconds means that a feasible solution was found, a **P** means that solutions were found to the first part of the decomposition models, but that the solutions were not feasible, and a **'-'** means that no solution was found in the allotted three hours.

Table 6. Satisfaction experiment results for the competition instances.

Problem Instance	**CP 1**	**CP Scheduling 1**	**IP 1**	**CP 2**	**CP Scheduling 2**	**IP 2**
1	-	-	6.91	P	-	P
2	-	-	6.82	P	-	P
3	-	-	7.97	P	-	P
4	-	-	9.19	-	-	P
5	-	-	8.17	-	-	P
6	-	-	9.37	-	-	P
7	-	-	10.14	533.34	-	P
8	-	-	8.76	P	-	P
9	-	-	8.94	P	-	P
10	-	-	7.91	P	-	P
11	-	-	7.47	P	-	P
12	-	-	7.03	P	-	P
13	-	-	8.22	-	-	P
14	-	-	10.2	-	-	P
15	-	-	8.45	-	-	P
16	-	-	**9.52**	P	-	P
17	-	-	7.63	P	-	P
18	-	-	6.85	P	-	P
19	-	-	10.5	-	-	P
20	-	-	9.13	62.9	-	P

If the pure IP model is not considered, the CP decomposition model works the best, but the IP model far surpasses the performance of any other model. This is surprising, since CP seems to be better suited to timetabling problems than IP due to the nature of the constraints.

In an attempt to understand why the pure IP model was the best, a linear programming (LP) model was created. The LP model was a relaxation of the IP model. It

had no integrality constraints and no soft constraints. The rest of the set up was the same as the pure IP model. Each of the 21 test problems was run on the LP model. The results were as follows. Results are time to solve in seconds.

Table 7. Satisfaction experiment results for the test problems using the LP model.

	Test Problems	**Competition Problems**
0	0.49	N/a
1	0.86	6.96
2	0.88	6.85
3	0.95	7.44
4	0.81	9.77
5	0.92	9.02
6	0.9	9.21
7	0.83	10.53
8	0.81	8.61
9	0.83	8.92
10	0.81	7.48
11	0.8	7.41
12	0.75	7.05
13	0.85	8.31
14	0.93	10.76
15	1	9.17
16	0	9.21
17	0.85	8.95
18	0.86	7.23
19	0.86	10.39
20	0.94	9.28

Surprisingly, all of the solutions are integer. The likely explanation for why the IP model worked the best was that it was able to make good use of the LP bounds. In the case of the satisfaction experiments, the LP bounds even provided solutions, since the problem was naturally integer. The LP model was run on the competition instances also, to see if they were naturally integer as well and the model provided integer solutions to the competition instances as well.

The results also highlighted drawbacks inherent in the decomposition models. The first is that in the optimizations problems, the decomposition models could get stuck in the first section trying to find a good objective function value, when the results may not even be a feasible solution, thereby wasting a lot of time. The second drawback is that they are unable to see if a problem is not solvable. For example, test problem 16 had no solution. The decomposition models were not able to discover this.

7.6 Conclusion

In this chapter, we looked at several models using CP. Some of these models involved CP decomposition, a novel approach in timetabling. The models did show a little promise in terms of CP being used to solve timetabling problems. The constraints fit nicely with existing CP constraints, making the model very natural. The decomposition models did not work out well. There was one enlightening discovery that came out of the experiments: the LP model of the timetabling problem, when solving for feasibility alone, was naturally integer.

For the international competition, the solution techniques were extremely complicated, as are most of the successful solution techniques found in the literature. It would therefore be quite a breakthrough if in fact linear programming could be used to solve timetabling problems, even of a particular form. The form of this particular problem is similar to many university timetabling problems. In most universities, the problem could be broken down and a section could be modeled in this form so that linear programming could at least be part of the solution process. Although the LP gave an integer solution for the feasibility problems, it is unknown what solution it would give for any of the optimization problems or if the problem description were to be changed. We can therefore say that the results of this research are promising and merit further work.

The results of the experiments run in this chapter show that there is potential for linear programming as a tool for solving university course timetabling problems. Although, most real-world timetabling problems in their entirety would not be solvable using linear programming, this research shows that there may be a way to use linear programming to solve a good portion of a timetabling problem. In this chapter, it was shown that problems that are of the same form as the timetabling problems used for the international competition of metaheuristics in 2003, when feasibility alone is considered, appear to be naturally integer. The problems used for the competition were meant to resemble real-world university course timetabling problems, so it makes sense that most universities would be able to represent, at least part of their problem, in that form. They would then be able to use linear programming to solve that part of their problem. The advantage of this is that an LP model can be solved easily using well-known algorithms such as the simplex method. Currently, the successful timetabling algorithms are mostly local search techniques, which do not guarantee a feasible solution and integer programming techniques that can be very time consuming.

Future work would be to test the LP model on the problem instances and include some soft constraints. It would be interesting to see what objectives could be included and for the solution to remain integer. It would also be interesting to attempt to represent a real-world timetabling problem in the format described in this paper and to use the LP model to solve it. We could then get a better idea for how useful of a tool LP can be for solving a real-world timetabling problem.

Chapter 8

Conclusions and Future Work

In this chapter we reiterate the contributions from this thesis and suggest directions for future work based on this research.

8.1 Contributions

The contributions of this thesis follow the process of solving a real world problem. The three steps in solving a real world problem are (1) to analyze the problem, (2) develop a problem definition and evaluation criteria, and (3) to model and solve the problem. This thesis makes contributions to each of those three areas in the domain of operations research.

8.1.1 Analyzing the Problem

The main contribution to this area of problem solving is in taking a real world problem and going in detail over the process of how the problem is solved. By looking at the timetabling problem at APSC, we show that real world problems are much more complicated than what typically appears in a mathematical model. It is also more complicated than what appears in a typical research paper on timetabling. The complexity of the APSC problem emphasizes how difficult, if not impossible, it is to come up with a definition of an optimization problem that could be used to define a mathematical model. In the APSC problem, there wasn't one definition to the problem. As well, the definition was dynamic, based on human judgments about what constraints can be relaxed and data that has been gathered. The complexity of the APSC problem is motivation for research into how to define a problem, a problem-solving step that is not directly studied in the operations research domain. Also, a detailed process description is made of the APSC problem and problem areas, specifically ones where automation may be helpful, are highlighted. Solutions are suggested for all problem areas.

8.1.2 Developing a Problem Definition

The main contribution to this area of problem solving is in taking a real world problem and defining evaluation criteria. The creation of evaluation criteria is part of a problem definition. The evaluation criteria are complex and may be difficult to incorporate into a tradition optimization function for two reasons. (1) The metrics meant to show the quality of the schedule can be misleading and cannot be used in isolation from human judgment. (2) It is unclear how to balance different metrics automatically. Since, the metrics cannot be incorporated into a traditional optimization function, it is necessary to evaluate how a solution works. In the APSC case, a detailed set of evaluation criteria is useful and necessary

if one is to continue on in trying to find an automated timetabling solution. These evaluation criteria are implemented in the form of a Microsoft Access database that can score the quality of a timetable created through their current process. These evaluation criteria, together with the process description form one possible problem definition.

8.1.3 Modeling and Solving the Problem

The main contribution to this area of problem solving is in developing a mathematical model. Though the APSC problem is not used, since it is too big, a simpler university timetabling problem is used to test several mathematical models. Six mathematical programming models were created to solve a university timetabling problem of similar style to that of the APSC. These six models experimented particularly with Constraint Programming (CP) and decomposition techniques. These are ideas that have not been explored, in depth, as of yet in the automated timetabling research. The results of the experiments showed that the Integer Programming (IP) was the best technique for solving the problem, due to the fact that the timetabling problem being studied, when solved using a Linear Programming model, appeared to be naturally integer. This suggests potential for linear programming as a tool for solving university course timetabling problems.

8.2 Future Work

Many interesting questions arose from this thesis, suggesting possibilities for further research. The following sections outline possible directions for future work.

8.2.1 The Problem Definition Stage

The problem definition stage consists of choosing what information to include in a model of a real situation. It is settling on a level of abstraction that accurately represents the domain, but is not too complex that it is too hard to solve in a reasonable amount of time. There is more than one problem definition for each real world problem and, as shown in Chapter 2, a different problem definition can result in a different solution to the "same" problem.

The problem definition stage has been overlooked in the Operations Research literature. This is surprising because it seems to be a fundamental concept when applying optimization techniques to the real world. The problem definition stage exists in optimization problems, but it is skipped over in the research. In Chapter 3, we saw that software engineering and enterprise modeling are two areas that actively research the problem definition stage as well as validating different problem definitions in the real world. One direction for future research would be to try to develop some of the techniques that exist in the model-based diagnosis, software engineering, and enterprise modeling domains and apply them to operations research problems. This thesis made contributions towards these goals, but a lot of work still remains to achieve them.

As a first step, we would have to see what sorts of simplifying assumptions are useful given the nature of a problem. One way to research this would be to take a group of real world optimization problems that are of similar form and put them into mathematical

models. While doing this, we would keep track of the assumptions that we had to make for each of the problems. We could then see what assumptions, if any, were necessary for most of the problems. If we then repeated this process for other groupings of problems, we could develop a compilation of which assumptions are necessary for problems sharing a certain set of characteristics as well as which assumptions might be necessary for those problems.

Another direction for future work would be to experiment with different problem definitions using the APSC problem. The complexity of the APSC timetabling problem shows how difficult, if not impossible, it would be to create a definition that could be put into a mathematical model, making it a prime candidate for problem definition research. Some examples of possible problem definitions for the APSC problem are:

- A small part of the APSC problem as it is currently.
- A simplified version of the entire process.
- The scheduling part of the process ignoring soft constraints.
- The scheduling part of the process with some of the hard constraints relaxed.

We could put the different problem definitions into a mathematical model and see what effect different problem definitions had on the solution. We could then attempt to see how useful any of the solutions coming from those definitions are in the real world. We could also compare the solutions using the evaluation database.

Another direction for future work would be to expand the research on defining the problem and seeing how different definitions can have different results in the real world to optimization and the field of operations research as a whole by looking at other domains in the operations research field. We could take other problems, such as queuing theory problems or transportation routing problems and try to develop sets of problem definitions whose solutions we could then compare.

8.2.2 The APSC Evaluation Database

Several directions could be taken to extend on the work described in Chapter 5. From a research perspective, it would be useful to study quality metrics of all forms to see if there are metrics that could be placed in an objective function and that would accurately represent the desired qualitative effect.

Another direction would be to improve the current database. One idea is to add more and more detailed quality metrics. There are more complicated metrics, such as how courses are spread over the week and how students' breaks are spread over the course of a day. These metrics would provide more information to the schedulers. Another idea is to incorporate the database into the scheduling process at APSC. Microsoft Access can interface with Course Planner, the software used to create the timetable at APSC. The database could then be used to inform the scheduler as they make decisions throughout the scheduling process. For example, when the scheduler chooses to place a meeting for a course in a specific timeslot, it could show them how that changes the value of the quality metrics.

8.2.3 Extensions of the Mathematical Models

The results of the experiments run in Chapter 7 showed that there is potential for linear programming (LP) as a tool for solving university course timetabling problems. Although, most real-world timetabling problems in their entirety would not be solvable using linear programming, this research shows that there may be a way to use linear programming to solve a good portion of a timetabling problem. The experiments showed that problems that are of the same form as the timetabling problems used for the international competition of metaheuristics in 2003, when feasibility alone is considered, appear to be naturally integer. The problems used for the competition were meant to resemble real-world university course timetabling problems, so it makes sense that most universities would be able to represent, at least part of their problem, in that form. They would then be able to use LP to solve that part of their problem. The advantage of this is that an LP model can be solved easily using well-known algorithms such as the simplex method. Currently, the successful timetabling algorithms are mostly local search techniques, which do not guarantee a feasible solution and integer programming techniques that can be very time consuming.

Future work would be, firstly, to analyze the structure of the timetabling problems used in the competition in order to bring to light what it is that makes the problem naturally integer. There is quite a bit of research on what problem structures lead to naturally integer solutions in LP [89, 90] and it would be useful to know what it is in the competition problems that cause this property. Another direction for future work would be to test the LP model on the problem instances and include some soft constraints. It would be interesting to see what objectives could be included where the solution would remain integer. It would also be interesting to attempt to represent a real-world timetabling problem in the format described in Chapter 7 and to use the LP model to solve it. We could then get a better idea for how useful of a tool LP can be for solving a real-world timetabling problem.

8.3 Conclusions

The central thesis of this dissertation is that the problem definition stage of solving real world problems should be directly studied and that the problems must be studied in their entirety in order to create a solution that is useful in the real world. This was shown through the use of university course timetabling problems and the Faculty of Applied Science and Engineering at the University of Toronto's timetable, in particular. The problem definition stage has been identified as important and is studied in depth in both software engineering and enterprise modeling, yet this is not the case in the area of operations research. This work was an introduction of such research into the Operations Research (OR) field. University course timetabling has received much attention from researchers in both the OR and Artificial Intelligence (AI) fields and this work was a continuation of such effort. In particular, in this dissertation:

- We conducted a thorough analysis of the university course timetabling problem at the Faculty of Applied Science and Engineering at the University of Toronto (APSC) as an example of a real-world problem in operations research. The complexity of the APSC timetabling problem showed how difficult, if not

impossible, it would be to create a definition that could be put into a mathematical model. It also provided motivation for researching the problem definition phase.

- We created detailed evaluation criteria for the APSC timetable. Through the creation of an evaluation system for APSC, we saw that putting quality measures into an objective function of a mathematical model is difficult. The objective function would not accurately represent the desired quality metrics and since this is the case, it is important to evaluate how a given solution works back in the real world.

- Motivated by the constraint structure of university course timetabling problems, we applied constraint programming (CP), Integer Programming (IP), and decomposition techniques to a benchmark university course timetabling problem found in the literature. The results of the experiments showed that there is potential for linear programming (LP) as a tool for solving university course timetabling problems.

Bibliography

[1] Luca Di Gaspero, Stefano Mizzaro, and Andrea Schaerf. "A MultiAgent Architecture for Distributed Course Timetabling," in the Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling (PATAT-2004), Pittsburgh (PA), USA, August 2004. pp. 471-474.

[2] Atish Chand. "A Constraint Based Generic Model for Representing Complete University Timetabling Data," in the Proceedings of the Fifth International Conference on the Practice and Theory of Automated Timetabling. University of the South Pacific, 2004. pp. 125-150.

[3] Hadrien Cambazard, Fabien Demazeau, Narendra Jussien and Philippe David. "Interactively Solving School Timetabling Problems using Extensions of Constraint Programming," in the Proceedings of the Fifth International Conference on the Practice and Theory of Automated Timetabling. France, 2004. pp. 107-124.

[4] Tomas Muller and Hana Rudova. "Minimal Perturbation Problem in Course Timetabling," in the Proceedings of the Fifth International Conference on the Practice and Theory of Automated Timetabling. Prague, Czech Republic, 2004. pp. 283-304.

[5] Philipp Kostuch. "The University Course Timetabling Problem with a 3-Phase Approach," in the Proceedings of the Fifth International Conference on the Practice and Theory of Automated Timetabling. Oxford, UK, 2004. pp. 251-266.

[6] S. Peichowiak, J. Ma, R. Mandiau. "EDT-2004 : An Open Interactive Timetabling Tool," in the Proceedings of the Fifth International Conference on the Practice and Theory of Automated Timetabling. Valenciennes, France, 2004. pp. 305-322.

[7] Haroldo G. Santos, Luiz S. Ochi and Marcone J.F. Souza. "A Tabu Search Heuristic with Efficient Diversification Strategies for the Class/Teacher Timetabling Problem," in the Proceedings of the Fifth International Conference on the Practice and Theory of Automated Timetabling. Brazil, 2004. pp. 343-358.

[8] Carter, M.W. and Tovey, C. "When Is the Classroom Assignment Problem Hard?" Operations Research vol. 40, Supp. no. 1, January-February 1992, pp. 528-539.

[9] Carter, M.W. and Laporte, G. "Recent Developments in Practical Course Timetabling," Lecture Notes in Computer Science vol.1408. pp. 3-19. Springer Verlag, 1998.

[10] Petr Slechta. "Decomposition and Parallelization of Multi Resource Timetabling Problems," in the Proceedings of the Fifth International Conference on the Practice and Theory of Automated Timetabling. Prague, Czech Republic, 2004. pp. 359-370.

[11] Carter, M.W. "Comprehensive Course Timetabling and Student Scheduling System at the University of Waterloo," Lecture Notes in Computer Science no. 2079. pp. 64-84. Springer Verlag, 2001.

[12] Mirjana Cangalovic and Jan A.M. Schreuder. "Modelling and Solving an Acyclic Multi-Period Timetabling Problem," Discrete Applied Mathematics vol. 35. 1992. pp.177-195.

[13] Daniel Costa. "A Tabu Search Algorithm for Computing an Operational Timetable," European Journal of Operational Research vol.76. 1994. pp.98-110.

[14] Jean Aubin and Jacques A. Ferland. "A Large Scale Timetabling Problem," Computers and Operations Research vol.16, 1. 1989. pp.67-77.

[15] R. Fahrion and G. Dollansky. "Construction of University Faculty Timetables Using Logic Programming Techniques," Discrete Applied Mathematics vol. 35. 1992. pp.221-236.

[16] Nele Custers, Patrick De Causmaecker, Peter Demeester, Greet Vanden Berghe. "Semantic Components for Timetabling," in the Proceedings of the Fifth International Conference on the Practice and Theory of Automated Timetabling. Belgium, 2004. pp.169-182.

[17] Carter, M.W. "A Lagrangian Relaxation Approach to the Classroom Assignment Problem," INFOR vol. 27, No. 2. May 1989, pp. 230-246.

[18] John J. Dinkel, John Mote, and M. A. Venkataramanan. "An Efficient Decision Support System for Academic Course Scheduling," Operations Research vol. 37, no. 6. November/December, 1989. pp. 853.

[19] Tim B. Cooper and Jefferey H. Kingston. "The Solution of Real Instances of the Timetabling Problem," The Computer Journal vol. 36, no. 7. Sydney, Australia, 1993.

[20] Le Kang, George H. Von Schoenberg, and George M. White. "Complete University Timetabling Using Logic," Computers and Education vol.17, no. 2.1991. pp.145-153.

[21] Gilbert Laporte and Sylvain Desroches. "The Problem of Assigning Students to Course Sections in a Large Engineering School," Computers and Operations Research vol. 13, no.4. 1986. pp.387-394.

[22] Isao Miyaji, Katsuhisa Ohno, and Hisashi Mine. "Solution Method for Partitioning Students into Groups," European Journal of Operational Research vol. 33. 1987. pp.82-90.

[23] S. M. Selim. "An Algorithm for Constructing a University Faculty Timetable," Computers and Education vol.6. 1982. pp.323-332.

[24] Marc J. Schniederjans and Gyu Chan Kim. "A Goal Programming Model to Optimize Departmental Preference in Course Assignments," Computers and Operations Research vol.14, no. 2. 1987. pp. 87-96.

[25] Mike Wright. "School Timetabling Using Heuristic Search," Journal of the Operational Research Society vol. 47. 1996. pp.347-357.

[26] Le Kang and George M. White. "A Logic Approach to the Resolution of Constraints in Timetabling," European Journal of Operational Research vol. 61. 1992. pp.306-317.

[27] G. C. W. Sabin and G. K. Winter. "The Impact of Automated Timetabling on Universities - A Case Study," The Journal of the Operational Research Society vol. 37, no. 7. July, 1989. pp.689-693.

[28] Lledo Museros and M. Teresa Escrig. "A Qualitative Theory for Shape Representation and Matching," in the Proceedings of The Qualitative Reasoning Workshop. Brazil, 2003.

[29] A. M. Bos and M. J. L. Tiernego. "Formula Manipulation in the Bond Graph Modelling and Simulation of Large Mechanical Systems," Journal of The Franklin Institute. 1985.

[30] Tokuro Matsuo, Toramatsu Shintani, and Takayuki Ito. "An Economic Support System Based on Qualitative/Quantitative Simulations," in the Proceedings of The Qualitative Reasoning Workshop. Brazil, 2003.

[31] Jeroen Keppens and Qiang Shen. "Causality Enabled Compositional Modelling of Bayesian Networks," in the Proceedings of The Qualitative Reasoning Workshop. Brazil, 2003.

[32] Raffaella Guglielmann and Liliana Ironi. "The Need for Qualitative Reasoning in Fuzzy Modelling: Robustness and Interpretability Issues," in the Proceedings of The Qualitative Reasoning Workshop. Brazil, 2003.

[33] Carlos Alonso, Juan J. Rodriguez, and Belarmino Pulido. "Enhancing Consistency Based Diagnosis with Machine Learning Techniques," in the Proceedings of The Workshop on Principles of Diagnosis. 2001.

[34] Markus Stumptner and Franz Wotawa. "Coupling CSP Decomposition and Diagnosis for Tree-Structured Systems," in the Proceedings of The Workshop on Principles of Diagnosis. 2001.

[35] Sriram Narasimhan, Gautam Biswas, Gabor Karsai, Tal Pasternak, and Feng Zhao. "Building Observers to Address Fault Isolation and Control Problems in Hybrid Dynamic Systems." in Proceedings of the 2000 IEEE Intl. Conference on Systems, Man, and Cybernetics, Nashville, TN, October 2000. pp. 2393-2398.

[36] Gabor Karsai, Gautam Biswas, Tal Pasternak, and Sriram Narasimhan. "Fault-Adaptive Control: A CBS Application," in Proceedings of the Fourth International Conference on Intelligent Systems Design and Applications, Budapest, Hungary, Aug. 2004.

[37] Rita Marques Brandao and Acacio M. O. Porta Nova. "Non-Stationary Queue Simulation Analysis Using Time Series," in the Proceedings of the 2003 Winter Simulation Conference. New Orleans, Louisiana, USA. pp. 408-413.

[38] Nong Ye, Esma S. Gel, Xueping Li, Toni Farley, and Ying-Cheng Lai. "Web Server QoS Models: Applying Scheduling Rules From Production Planning," Computers and Operations Research vol. 32. 2005. pp.1147-1164.

[39] Eric Ruethe. "A Definition of Software Architecture," in Software Architecture in Practice. Addison-Wesley, 1997.

[40] Roger S. Pressman. Software Engineering: A Practitioner's Approach.5th edition, Ch.10-11. McGraw-Hill Higher Education, 2000.

[41] Armin Eberlein and Julio Cesar Sampaio do Prado Leite. "Agile Requirements Definition: A View from Requirements Engineering." in the Proceedings of the International Workshop on Time-Constrained Requirements Engineering, Germany, 2002.

[42] Bashar Nuseibeh and Steve Easterbrook. "Requirements Engineering: A Roadmap." in the Proceedings of the 22nd International Conference on on Software Engineering pp. 35-46

[43] David Garlan. "Software Architecture: A Roadmap." Addison Wesley, 2000.

[44] R. C. Rosenberg, E.D. Goodman, and Kisung Seo. "Some Key Issues in Using Bond Graphs and Genetic Programming for Mechatronic System Design." in the Proceedings of the ASME International Mechanical Engineering Congress and Exposition, New York, November 2001

[45] K. Medjaher, A. K. Samantaray, and B. Ould Bouamama. "Diagnostic Bond Graphs for Direct Residual Evaluation." in the Proceedings of the 2005 International Conference on Bond Graph Modeling, New Orleans, Louisiana, 2005.

[46] Martin Dzbor. "Design as a Problem of Requirements Explication."in the Proceedings of the Workshop on Knowledge-Based Systems for Model-Based Engineering Design, European Conference on AI (ECAI'2000), Berlin, Germany, 2000.

[47] Leliane Nunes de Barros, Marilza Lemos, Volnys Bernal, and Jaques Wainer. “Model Based Diagnosis for Network Communication Faults.” In the Proceedings of the Third International Workshop on Artificial Intelligence in Distributed Information Networking Orlando, Florida, July,1999. pp. 57-62.

[48] Mattias Nyberg. “Framework and Method for Model Based Diagnosis with Application to an Automotive Engine.” in the Proceedings of the European Control Conference, Linkoping, Sweden. 1999.

[49] Hiroyuki Sawada and Xiu-Tian Yan. “Applying a Generic Constraint Solving Technique to Engineering Design.” in the Proceedings of the 13th International Conference on Engineering Design, 2001.

[50] Michael Valasek and Zdenek Zdrahal. “Knowledge Models in Engineering Design.” Lecture notes in computer science Vol 2736, 2003.

[51] Kent Andersson. “Components for Integrating Heuristic and Model-Based Diagnosis.” Sweden. in the Proceedings of the Knowledge Acquisition Workshop ’98, 1998.

[52] Dieter Fensel and Richard Benjamins. “Assumptions in Model-Based Diagnosis.” in the Proceedings of the Knowledge Acquisition Workshop’96, 1996.

[53] Walt Sacchi. “Process Models in Software Engineering.” in J. J. Marciniak (ed.), Encyclopedia of Software Engineering, 2nd Edition. John Wiley and Sons, Inc, New York, December, 2001.

[54] Gary J. Nutt. “Software Engineering Process Model - A Case Study.” in: Proceedings of the ACM Conference on Organizational Computing Systems 1995. August 13-16, 1995, Milpitas, California, USA. pp.324-335.

[55] Mark S. Fox and Michael Gruninger. "Enterprise Modelling", AI Magazine, AAAI Press, Fall 1998, pp. 109-121.

[56] K. Donald Tham and Mark S. Fox. “Determining Requirements and Specifications of Enterprise Information Systems for Profitability.” in the Proceedings of the 6th International Conference on Enterprise Information Systems. 2004.

[57] Michael Gruninger and Mark S. Fox. “The Role of Competency Questions in Enterprise Engineering,” submitted to IFIP WG5.7 Workshop on Benchmarking -Theory and Practice. Trodheim, Norway, 1994.

[58] Ron Jeffries. “What is Extreme Programming.” (web document) in XProgramming.com an Agile Software Developement Resource. November 8, 2001. available at: http://xprogramming.com/xpmag/whatisxp.htm. Accessed on April 22, 2007.

[59] J. E. Beasley. "OR-Notes,"(online document), 2004. available at: *http://people.brunel.ac.uk/~mastjjb/jeb/or/contents.html*. Accessed on April 22, 2007.

[60] Michael Trick and Gerard Cornuejols. "Quantitative Methods for the Management Sciences" 45-760 Course Notes. for the Graduate School of Industrial Administration, Carnegie Mellon University. Pittsburgh, PA. Fall, 1998. Chapter 11.

[61] "Bond Graph" (online article) available at: http://en.wikipedia.org/wiki/Bond_graph. Accessed on April 22, 2007.

[62] Woljciech Legierski and Pawe Parys. "System for Solving Timetabling Problems," in the Proceedings of the Symposium on Methods of Artificial Intelligence, pp 76-77, 2003.

[63] Tuan-Anh Duong and Kim-Hoa Lam. "Combining Constraint Programming and Simulated Annealing on University Exam Timetabling," in the Proceedings of the International Conference of Research, Innovation and Vision for the Future 2004. Hanoi, Vietnam. February 2-5.

[64] Slim Abdennadher and Micahel Marte. "University Course Timetabling Using Constraint Handling Rules." Applied Artificial Intelligence vol. 14. 2000. pp. 311-325.

[65] Christelle Gueret, Narendra Jussien, Patrice Boizumault, and Christian Prins. "Building University Timetables Using Constraint Logic Programming." in the Proceedings of the First International Conference on the Practice and Theory of Automated Timetabling (ICPTAT '95)1995.

[66] S. Daskalaki, T. Birbas, and E. Housos. "An Integer Programming Formulation for a Case Study in University Timetabling." European Journal of Operational Research vol.153. 2004. pp.117-135.

[67] N. L. Lawrie. "An Integer Linear Programming Model of a School Timetabling Problem." The Computer Journal 1969, vol. 12, no. 4, pp.307-316.

[68] Serge Kruk with Eddie Cheng and Laszlo Liptak. "CP vs IP Hybrid Approach to Timetabling Problems." in the Midwest Optimization Seminar. Kalamazoo, October 2005.

[69] A. Schaerf. "A Survey of Automated Timetabling," Artificial Intelligence Review vol.13. 1999. pp.87-127.

[70] Roman Bartak. "Constraint Programming: In Pursuit of the Holy Grail," in the Proceedings of the Week of Doctoral Students '99. Prague, June, 1999

[71] Barbara M. Smith. "Modelling for Constraint Programming," in Constraint Programming Summer School. Italy, September, 2005.

[72] P. Prosser. “Hybrid algorithms for the constraint satisfaction problem,” Computational Intelligence, vol. 9, no. 3. 1993. pp. 268-299.

[73] Barbara. M. Smith and Stuart. A. Grant. “Trying harder to fail first,” in the Proceedings of the Thirteenth European Conference on Artificial Intelligence (ECAI 98). 1998, pp. 249-253.

[74] W.J. van Hoeve, “The all-different constraint: A systematic overview.” in the Proceedings of the Sixth Annual Workshop of the ERCIM Working Group on Constraints, Prague, June 2001.

[75] International Timetabling Competition. Metaheuristics Network.(web page) available at: http://www.idsia.ch/Files/ttcomp2002/oldindex.html. accessed on April 22, 2007

[76] Marco Chiarandini, Krzyztof Socha, Mauro Birattari, and Olivia Rossi Doria. “An Effective Approach for the University Course Timetabling Problem.“ Journal of Scheduling, vol. 9, no. 5, pp. 403-432. 2006.

[77] International Timetabling Competition Results. Metaheuristics Network.(web page) available at: http://www.idsia.ch/Files/ttcomp2002/results.html. accessed on April 22, 2007

[78] Public Course Timetabling Data. (web link) available at: http://iridia.ulb.ac.be/~msampels/tt.data/ accessed on April 22, 2007

[79] Mahadevan, S., Marchalleck, N., Das, T., and Gosavi, A.,”Self Improving Factory Simulation using Continuous-time-Average-Reward Reinforcement Learning.“ in the Proceedings of the 14th International Conference on Machine Learning), Nashville, TN, July 1997.

[80] Gjerdrum, J., Shah, N., Papageorgiou, L, ”A combined optimization and agent-based approach to supply chain modelling and performance assessment,” Production Planning & Control, vol.12, no. 1.2001. pp. 81-88.

[81] Adams, J., Balas, E., and Zawack, D., ”The Shifting Bottleneck Procedure for Job Shop Scheduling,” Management Science, Vol. 34, No. 3. 1988. pp. 391-401.

[82] Brucker, P., Jurisch, B., and Sievers, B., ”A branch and bound algorithm for the job-shop scheduling problem,”Discrete Applied Mathematics,Vol. 49 , Issue 1-3. 1994. pp.107-127.

[83] Cordeau, J., Gendreau, M., Laporte, G, Potvin, J., ”A guide to vehicle routing heuristics.” Journal of the Operational Research Society, vol. 53, 2000. pp.512–522.

[84] Lysgaard, J., Letchford, A., and Eglese, R.,”A new branch-and-cut algorithm for the capacitated vehicle routing problem.” Mathematical Programming, Vol. 100, No. 2. 2004. pp. 423-445.

[85] Bazaraa, M., Jarvis, J., Sheal, H., Linear Programming and Network Flows Third Edition Chapter 1, page 7.John Wiley and Sons. NJ, 2005.

[86] Flener, P., Frisch, A. M., Hnich, B., Kiziltan, Z., Miguel, I., and Walsh, T., "Matrix Modelling: Exploiting Common Patterns in Constraint Programming." in the Proceedings of the International Workshop on Reformulating Constraint Satisfaction Problems, pp. 27-41, 2002

[87] Guerri, A. and Milano, M., "The Importance of Relaxations and Bender's Cuts in Decomposition Techniques: Two Case Studies." in Proceedings of the CP 2006 Doctoral Programme, Nantes, France, September 2006, pp. 162-167.

[88] Rossi, F., van Beek, P., Walsh, T.(eds.),Constraint Programming. Chapter 1. 2006. Elsevier.

[89] Chinneck, John W.,Practical Optimization: A Gentle Introduction. Chapter 10. (Web Document) available at: http://www.sce.carleton.ca/faculty/chinneck/po/Chapter10.pdf, 2001. Accessed on April 22, 2007.

[90] Dubhashi, Devdatt, P., "What Can't You Do With an LP?," in the Basic Research In Computer Science Lecture Series. December, 1996.

[91] J. N. Hooker, "A hybrid method for planning and scheduling," Constraints vol. 10, 2005. pp. 385-401 .

Appendix A: SQL Queries

q_Conflicts_by_POSt_49

```
SELECT [Copy of q_conflict1].SESSION, [Copy of q_conflict1].POST_CD, [Copy of
q_conflict1].CountOfMEET_END_SUFFIX1 AS Conflicts, Count([Copy of
q_conflict1].POST_CD) AS [Conflict Count]
FROM [Copy of q_conflict1]
GROUP BY [Copy of q_conflict1].SESSION, [Copy of q_conflict1].POST_CD, [Copy of
q_conflict1].CountOfMEET_END_SUFFIX1
HAVING ((([Copy of q_conflict1].SESSION)=20049));
```

q_Conflicts_by_POSt_51

```
SELECT [Copy of q_conflict1].SESSION, [Copy of q_conflict1].POST_CD, [Copy of
q_conflict1].CountOfMEET_END_SUFFIX1 AS Conflicts, Count([Copy of
q_conflict1].POST_CD) AS [Conflict Count]
FROM [Copy of q_conflict1]
GROUP BY [Copy of q_conflict1].SESSION, [Copy of q_conflict1].POST_CD, [Copy of
q_conflict1].CountOfMEET_END_SUFFIX1
HAVING ((([Copy of q_conflict1].SESSION)=20051));
```

q_Conflicts_by_POSt_59

```
SELECT [Copy of q_conflict1].SESSION, [Copy of q_conflict1].POST_CD, [Copy of
q_conflict1].CountOfMEET_END_SUFFIX1 AS Conflicts, Count([Copy of
q_conflict1].POST_CD) AS [Conflict Count]
FROM [Copy of q_conflict1]
GROUP BY [Copy of q_conflict1].SESSION, [Copy of q_conflict1].POST_CD, [Copy of
q_conflict1].CountOfMEET_END_SUFFIX1
HAVING ((([Copy of q_conflict1].SESSION)=20059));
```

q_Conflicts_by_POSt_61

```
SELECT [Copy of q_conflict1].SESSION, [Copy of q_conflict1].POST_CD, [Copy of
q_conflict1].CountOfMEET_END_SUFFIX1 AS Conflicts, Count([Copy of
q_conflict1].POST_CD) AS [Conflict Count]
FROM [Copy of q_conflict1]
GROUP BY [Copy of q_conflict1].SESSION, [Copy of q_conflict1].POST_CD, [Copy of
q_conflict1].CountOfMEET_END_SUFFIX1
HAVING ((([Copy of q_conflict1].SESSION)=20061));
```

copy of q_conflict1

```
SELECT t_Student_Schedule.SESSION, t_Student_Schedule.PERSON_ID,
t_Student_Schedule.MEET_DAY1, t_Student_Schedule.MEET_START_TM1,
Count(t_Student_Schedule.MEET_END_SUFFIX1) AS CountOfMEET_END_SUFFIX1,
t_Student_Schedule.POST_CD
FROM t_Student_Schedule
GROUP BY t_Student_Schedule.SESSION, t_Student_Schedule.PERSON_ID,
t_Student_Schedule.MEET_DAY1, t_Student_Schedule.MEET_START_TM1,
t_Student_Schedule.POST_CD
HAVING (((Count(t_Student_Schedule.MEET_END_SUFFIX1))>1));
```

Make_conlict_count

```
SELECT q_conflict1.SESSION, q_conflict1.CountOfMEET_END_SUFFIX1 AS Conflict,
Count(q_conflict1.CountOfMEET_END_SUFFIX1) AS [Count]
FROM q_conflict1
GROUP BY q_conflict1.SESSION, q_conflict1.CountOfMEET_END_SUFFIX1;
```

q_conflict1

```
SELECT t_Student_Schedule.SESSION, t_Student_Schedule.PERSON_ID,
t_Student_Schedule.MEET_DAY1, t_Student_Schedule.MEET_START_TM1,
Count(t_Student_Schedule.MEET_END_SUFFIX1) AS CountOfMEET_END_SUFFIX1
FROM t_Student_Schedule
GROUP BY t_Student_Schedule.SESSION, t_Student_Schedule.PERSON_ID,
t_Student_Schedule.MEET_DAY1, t_Student_Schedule.MEET_START_TM1
HAVING (((Count(t_Student_Schedule.MEET_END_SUFFIX1))>1));
```

q_Early_Starts_by_POSt_49

```
SELECT [Copy of q_early2].SESSION, [Copy of q_early2].POST_CD, [Copy of
q_early2].CountOfPERSON_ID AS Early_Starts, Count([Copy of
q_early2].CountOfPERSON_ID) AS [Count]
FROM [Copy of q_early2]
GROUP BY [Copy of q_early2].SESSION, [Copy of q_early2].POST_CD, [Copy of
q_early2].CountOfPERSON_ID
HAVING ((([Copy of q_early2].SESSION)=20049));
```

q_Early_Starts_by_POSt_51

```
SELECT [Copy of q_early2].SESSION, [Copy of q_early2].POST_CD, [Copy of
q_early2].CountOfPERSON_ID AS Early_Starts, Count([Copy of
q_early2].CountOfPERSON_ID) AS [Count]
FROM [Copy of q_early2]
GROUP BY [Copy of q_early2].SESSION, [Copy of q_early2].POST_CD, [Copy of
q_early2].CountOfPERSON_ID
HAVING ((([Copy of q_early2].SESSION)=20051));
```

q_Early_Starts_by_POSt_59

```
SELECT [Copy of q_early2].SESSION, [Copy of q_early2].POST_CD, [Copy of
q_early2].CountOfPERSON_ID AS Early_Starts, Count([Copy of
q_early2].CountOfPERSON_ID) AS [Count]
FROM [Copy of q_early2]
GROUP BY [Copy of q_early2].SESSION, [Copy of q_early2].POST_CD, [Copy of
q_early2].CountOfPERSON_ID
HAVING ((([Copy of q_early2].SESSION)=20059));
```

q_Early_Starts_by_POSt_61

```
SELECT [Copy of q_early2].SESSION, [Copy of q_early2].POST_CD, [Copy of
q_early2].CountOfPERSON_ID AS Early_Starts, Count([Copy of
q_early2].CountOfPERSON_ID) AS [Count]
FROM [Copy of q_early2]
GROUP BY [Copy of q_early2].SESSION, [Copy of q_early2].POST_CD, [Copy of
q_early2].CountOfPERSON_ID
HAVING ((([Copy of q_early2].SESSION)=20061));
```

Copy of q_early2

```
SELECT [Copy of q_early1].SESSION, [Copy of q_early1].POST_CD, [Copy of
q_early1].PERSON_ID, Count([Copy of q_early1].PERSON_ID) AS CountOfPERSON_ID
FROM [Copy of q_early1]
GROUP BY [Copy of q_early1].SESSION, [Copy of q_early1].POST_CD, [Copy of
q_early1].PERSON_ID;
```

copy of q_early1

```
SELECT t_Student_Schedule.SESSION, t_Student_Schedule.PERSON_ID,
t_Student_Schedule.MEET_DAY1, t_Student_Schedule.POST_CD
FROM t_Student_Schedule
WHERE (((t_Student_Schedule.MEET_START_TM1)<=#12/30/1899 9:0:0#))
GROUP BY t_Student_Schedule.SESSION, t_Student_Schedule.PERSON_ID,
t_Student_Schedule.MEET_DAY1, t_Student_Schedule.POST_CD;
```

Make_early_starts

```
SELECT q_early2.SESSION, q_early2.CountOfPERSON_ID AS Early_Starts,
Count(q_early2.CountOfPERSON_ID) AS [Count]
FROM q_early2
GROUP BY q_early2.SESSION, q_early2.CountOfPERSON_ID;
```

q_early2

```
SELECT q_early1.SESSION, q_early1.PERSON_ID, Count(q_early1.PERSON_ID) AS
CountOfPERSON_ID
FROM q_early1
GROUP BY q_early1.SESSION, q_early1.PERSON_ID;
```

q_early1

```
SELECT t_Student_Schedule.SESSION, t_Student_Schedule.PERSON_ID,
t_Student_Schedule.MEET_DAY1
FROM t_Student_Schedule
WHERE (((t_Student_Schedule.MEET_START_TM1)<=#12/30/1899 9:0:0#))
GROUP BY t_Student_Schedule.SESSION, t_Student_Schedule.PERSON_ID,
t_Student_Schedule.MEET_DAY1;
```

q_Friday_Prayer_by_POSt

```
SELECT [Copy of q_friday2].SESSION, [Copy of q_friday2].POST_CD, Count([Copy of
q_friday2].POST_CD) AS No_Friday_Break
FROM [Copy of q_friday2]
GROUP BY [Copy of q_friday2].SESSION, [Copy of q_friday2].POST_CD;
```

Copy of q_friday2

```
SELECT [Copy of q_friday1].SESSION, [Copy of q_friday1].PERSON_ID, [Copy of
q_friday1].POST_CD, Count([Copy of q_friday1].MEET_START_TM1) AS
CountOfMEET_START_TM11
FROM [Copy of q_friday1]
GROUP BY [Copy of q_friday1].SESSION, [Copy of q_friday1].PERSON_ID, [Copy of
q_friday1].POST_CD
HAVING (((Count([Copy of q_friday1].MEET_START_TM1))=2));
```

copy of q_Friday1

```
SELECT t_Student_Schedule.SESSION, t_Student_Schedule.PERSON_ID,
t_Student_Schedule.MEET_DAY1, t_Student_Schedule.MEET_START_TM1,
t_Student_Schedule.POST_CD
FROM t_Student_Schedule
WHERE (((t_Student_Schedule.MEET_DAY1)="FR") AND
((t_Student_Schedule.MEET_START_TM1)=#12/30/1899 12:0:0#)) OR
(((t_Student_Schedule.MEET_START_TM1)=#12/30/1899 13:0:0#));
```

q_Make_Friday_Prayer

```
SELECT q_friday2.SESSION, Count(q_friday2.SESSION) AS No_Friday_Break
FROM q_friday2
GROUP BY q_friday2.SESSION;
```

q_Friday2

```
SELECT q_friday1.SESSION, q_friday1.PERSON_ID,
Count(q_friday1.MEET_START_TM1) AS CountOfMEET_START_TM1
FROM q_friday1
GROUP BY q_friday1.SESSION, q_friday1.PERSON_ID
HAVING (((Count(q_friday1.MEET_START_TM1))=2));
```

q_Friday1

```
SELECT t_Student_Schedule.SESSION, t_Student_Schedule.PERSON_ID,
t_Student_Schedule.MEET_DAY1, t_Student_Schedule.MEET_START_TM1
FROM t_Student_Schedule
WHERE (((t_Student_Schedule.MEET_DAY1)="FR") AND
((t_Student_Schedule.MEET_START_TM1)=#12/30/1899 12:0:0#)) OR
(((t_Student_Schedule.MEET_START_TM1)=#12/30/1899 13:0:0#));
```

q_Late_Ends_by_POSt_49

```
SELECT [Copy of q_late2].SESSION, [Copy of q_late2].POST_CD, [Copy of
q_late2].CountOfPERSON_ID AS La te_Ends, Count([Copy of
q_late2].CountOfPERSON_ID) AS [Count]
FROM [Copy of q_late2]
GROUP BY [Copy of q_late2].SESSION, [Copy of q_late2].POST_CD, [Copy of
q_late2].CountOfPERSON_ID
HAVING ((([Copy of q_late2].SESSION)=20049));
```

q_Late_Ends_by_POSt_51

```
SELECT [Copy of q_late2].SESSION, [Copy of q_late2].POST_CD, [Copy of
q_late2].CountOfPERSON_ID AS Late_Ends, Count([Copy of
q_late2].CountOfPERSON_ID) AS [Count]
FROM [Copy of q_late2]
GROUP BY [Copy of q_late2].SESSION, [Copy of q_late2].POST_CD, [Copy of
q_late2].CountOfPERSON_ID
HAVING ((([Copy of q_late2].SESSION)=20051));
```

q_Late_Ends_by_POSt_59

```
SELECT [Copy of q_late2].SESSION, [Copy of q_late2].POST_CD, [Copy of
q_late2].CountOfPERSON_ID AS Late_Ends, Count([Copy of
q_late2].CountOfPERSON_ID) AS [Count]
FROM [Copy of q_late2]
GROUP BY [Copy of q_late2].SESSION, [Copy of q_late2].POST_CD, [Copy of
q_late2].CountOfPERSON_ID
HAVING ((([Copy of q_late2].SESSION)=20059));
```

q_Late_Ends_by_POSt_61

```
SELECT [Copy of q_late2].SESSION, [Copy of q_late2].POST_CD, [Copy of
q_late2].CountOfPERSON_ID AS Late_Ends, Count([Copy of
q_late2].CountOfPERSON_ID) AS [Count]
FROM [Copy of q_late2]
GROUP BY [Copy of q_late2].SESSION, [Copy of q_late2].POST_CD, [Copy of
q_late2].CountOfPERSON_ID
HAVING ((([Copy of q_late2].SESSION)=20061));
```

copy of q_late2

```
SELECT [Copy of q_late1].SESSION, [Copy of q_late1].PERSON_ID, [Copy of
q_late1].POST_CD, Count([Copy of q_late1].PERSON_ID) AS CountOfPERSON_ID
FROM [Copy of q_late1]
GROUP BY [Copy of q_late1].SESSION, [Copy of q_late1].PERSON_ID, [Copy of
q_late1].POST_CD;
```

copy of q_late1

```
SELECT t_Student_Schedule.SESSION, t_Student_Schedule.PERSON_ID,
t_Student_Schedule.MEET_DAY1, t_Student_Schedule.POST_CD
FROM t_Student_Schedule
WHERE (((t_Student_Schedule.MEET_END_SUFFIX1)>=#12/30/1899 17:0:0#))
GROUP BY t_Student_Schedule.SESSION, t_Student_Schedule.PERSON_ID,
t_Student_Schedule.MEET_DAY1, t_Student_Schedule.POST_CD;
```

q_Make_late_ends

```
SELECT q_late2.SESSION, q_late2.CountOfPERSON_ID1 AS Late_Ends,
Count(q_late2.CountOfPERSON_ID1) AS [Count]
FROM q_late2
GROUP BY q_late2.SESSION, q_late2.CountOfPERSON_ID1;
```

q_late2

```
SELECT q_late1.SESSION, q_late1.PERSON_ID, Count(q_late1.PERSON_ID) AS
CountOfPERSON_ID1
FROM q_late1
GROUP BY q_late1.SESSION, q_late1.PERSON_ID;
```

q_late1

```
SELECT t_Student_Schedule.SESSION, t_Student_Schedule.PERSON_ID,
t_Student_Schedule.MEET_DAY1
FROM t_Student_Schedule
WHERE (((t_Student_Schedule.MEET_END_SUFFIX1)>=#12/30/1899 17:0:0#))
GROUP BY t_Student_Schedule.SESSION, t_Student_Schedule.PERSON_ID,
t_Student_Schedule.MEET_DAY1;
```

q_Lunch_Break_by_POSt

```
SELECT [Copy of q_lunch2].SESSION, [Copy of q_lunch2].POST_CD, Count([Copy of
q_lunch2].POST_CD) AS No_Lunch_Break
FROM [Copy of q_lunch2]
GROUP BY [Copy of q_lunch2].SESSION, [Copy of q_lunch2].POST_CD;
```

Copy of q_lunch2

```
SELECT [Copy of q_lunch1].SESSION, [Copy of q_lunch1].PERSON_ID, [Copy of
q_lunch1].POST_CD, [Copy of q_lunch1].MEET_DAY1
FROM [Copy of q_lunch1]
GROUP BY [Copy of q_lunch1].SESSION, [Copy of q_lunch1].PERSON_ID, [Copy of
q_lunch1].POST_CD, [Copy of q_lunch1].MEET_DAY1
HAVING (((Count([Copy of q_lunch1].MEET_DAY1))=2));
```

copy of q_lunch1

```
SELECT t_Student_Schedule.SESSION, t_Student_Schedule.PERSON_ID,
t_Student_Schedule.MEET_DAY1, t_Student_Schedule.MEET_START_TM1,
t_Student_Schedule.POST_CD
FROM t_Student_Schedule
```

```
WHERE (((t_Student_Schedule.MEET_START_TM1)=#12/30/1899 11:0:0# Or
(t_Student_Schedule.MEET_START_TM1)=#12/30/1899 12:0:0#));
```

Make_lunch_break

```
SELECT q_lunch2.SESSION, Count(q_lunch2.SESSION) AS No_Lunch_Break
FROM q_lunch2
GROUP BY q_lunch2.SESSION;
```

q_lunch2

```
SELECT q_lunch1.SESSION, q_lunch1.PERSON_ID, q_lunch1.MEET_DAY1
FROM q_lunch1
GROUP BY q_lunch1.SESSION, q_lunch1.PERSON_ID, q_lunch1.MEET_DAY1
HAVING (((Count(q_lunch1.MEET_DAY1))=2));
```

q_lunch1

```
SELECT t_Student_Schedule.SESSION, t_Student_Schedule.PERSON_ID,
t_Student_Schedule.MEET_DAY1, t_Student_Schedule.MEET_START_TM1
FROM t_Student_Schedule
WHERE (((t_Student_Schedule.MEET_START_TM1)=#12/30/1899 11:0:0# Or
(t_Student_Schedule.MEET_START_TM1)=#12/30/1899 12:0:0#));
```

q_UHigh_ POSt

```
SELECT q_U5.SESSION, q_U5.POST_CD, Count(q_U5.Utilization) AS High_Ut
FROM q_U5
GROUP BY q_U5.SESSION, q_U5.POST_CD
HAVING (((Count(q_U5.Utilization))>0.7));
```

q_U5

```
SELECT q_U2.SESSION, q_U2.POST_CD, q_U2.PERSON_ID, q_U2.MEET_DAY1,
q_U2!Hours_On/q_U4!Day_length AS Utilization
FROM q_U2 INNER JOIN q_U4 ON (q_U2.SESSION = q_U4.SESSION) AND
(q_U2.POST_CD = q_U4.POST_CD) AND (q_U2.PERSON_ID = q_U4.PERSON_ID)
AND (q_U2.MEET_DAY1 = q_U4.MEET_DAY1);
```

qU2

```
SELECT q_U1.SESSION, q_U1.POST_CD, q_U1.PERSON_ID, q_U1.MEET_DAY1,
Sum(q_U1.Course_length) AS Hours_On
FROM q_U1
GROUP BY q_U1.SESSION, q_U1.POST_CD, q_U1.PERSON_ID, q_U1.MEET_DAY1;
```

qU4

```
SELECT q_U3.SESSION, q_U3.POST_CD, q_U3.PERSON_ID, q_U3.MEET_DAY1,
Hour(q_U3!End-q_U3!Start) AS Day_length
FROM q_U3;
```

qU1

```
SELECT t_Student_Schedule.SESSION, t_Student_Schedule.PERSON_ID,
t_Student_Schedule.POST_CD, t_Student_Schedule.MEET_DAY1,
t_Student_Schedule.MEET_START_TM1, t_Student_Schedule.MEET_END_SUFFIX1,
Hour(t_Student_Schedule!MEET_END_SUFFIX1-
t_Student_Schedule!MEET_START_TM1) AS Course_length
FROM t_Student_Schedule;
```

qU3

```
SELECT t_Student_Schedule.SESSION, t_Student_Schedule.POST_CD,
t_Student_Schedule.PERSON_ID, t_Student_Schedule.MEET_DAY1,
Min(t_Student_Schedule.MEET_START_TM1) AS Start,
Max(t_Student_Schedule.MEET_END_SUFFIX1) AS [End]
FROM t_Student_Schedule
GROUP BY t_Student_Schedule.SESSION, t_Student_Schedule.POST_CD,
t_Student_Schedule.PERSON_ID, t_Student_Schedule.MEET_DAY1;
```

q_UMed_POSt

```
SELECT q_UMed1.SESSION, q_UMed1.POST_CD, Count(q_UMed1.Utilization) AS
Med_Ut
FROM q_UMed1
GROUP BY q_UMed1.SESSION, q_UMed1.POST_CD;
```

q_UMed1

```
SELECT q_U5.SESSION, q_U5.POST_CD, q_U5.Utilization
FROM q_U5
WHERE (((q_U5.Utilization)>0.4 And (q_U5.Utilization)<=0.7));
```

q_ULow_POSt

```
SELECT q_ULow1.SESSION, q_ULow1.POST_CD, Count(q_ULow1.Utilization) AS
Low_Ut
FROM q_ULow1
GROUP BY q_ULow1.SESSION, q_ULow1.POST_CD;
```

q_ULow1

```
SELECT q_U5.SESSION, q_U5.POST_CD, q_U5.Utilization
FROM q_U5
WHERE (((q_U5.Utilization)<=0.4));
```

q_ULevel_Year

```
SELECT q_UHigh_Year.SESSION, q_ULow_Year.Low_Ut, q_UMed_Year.Med_Ut,
q_UHigh_Year.High_Ut
FROM (q_UHigh_Year INNER JOIN q_ULow_Year ON q_UHigh_Year.SESSION =
q_ULow_Year.SESSION) INNER JOIN q_UMed_Year ON q_ULow_Year.SESSION =
q_UMed_Year.SESSION;
```

q_UHigh_Year

```
SELECT q_U5.SESSION, Count(q_U5.Utilization) AS High_Ut
FROM q_U5
GROUP BY q_U5.SESSION
HAVING (((Count(q_U5.Utilization))>0.7));
```

q_UMed_Year

```
SELECT q_U5.SESSION, Count(q_U5.Utilization) AS Med_Ut
FROM q_U5
GROUP BY q_U5.SESSION
HAVING (((Count(q_U5.Utilization)>0.4 And (q_U5.Utilization)<=0.7));
```

q_ULow_Year

```
SELECT q_U5.SESSION, Count(q_U5.Utilization) AS Low_Ut
FROM q_U5
GROUP BY q_U5.SESSION
HAVING (((Count(q_U5.Utilization))<=0.4));
```

q_UAvg_Year

```
SELECT q_U5.SESSION, Avg(q_U5.Utilization) AS Avg_Ut
FROM q_U5
GROUP BY q_U5.SESSION;
```

q_UAvg_POSt

```
SELECT q_U5.SESSION, q_U5.POST_CD, Avg(q_U5.Utilization) AS Avg_Ut
FROM q_U5
GROUP BY q_U5.SESSION, q_U5.POST_CD;
```

q_Make_Room_Util

```
SELECT q_roomut3.SESSION, q_roomut3.MEET_BUILDING_CD1,
q_roomut3.MEET_ROOM_NR1, (q_roomut3!SumOfTime+q_roomut4!SumOfTime)/40
AS Room_Util
FROM q_roomut3 INNER JOIN q_roomut4 ON (q_roomut3.MEET_ROOM_NR1 =
q_roomut4.MEET_ROOM_NR1) AND (q_roomut3.MEET_BUILDING_CD1 =
q_roomut4.MEET_BUILDING_CD1) AND (q_roomut3.SESSION =
q_roomut4.SESSION);
```

q_roomut3

```
SELECT q_roomut1.SESSION, q_roomut1.MEET_BUILDING_CD1,
q_roomut1.MEET_ROOM_NR1, Sum(q_roomut1.Time) AS SumOfTime
FROM q_roomut1
GROUP BY q_roomut1.SESSION, q_roomut1.MEET_BUILDING_CD1,
q_roomut1.MEET_ROOM_NR1
HAVING (((q_roomut1.MEET_BUILDING_CD1) Is Not Null) AND
((q_roomut1.MEET_ROOM_NR1) Is Not Null));
```

q_roomut4

```
SELECT q_roomut2.SESSION, q_roomut2.MEET_BUILDING_CD1,
q_roomut2.MEET_ROOM_NR1, Sum(q_roomut2.Time) AS SumOfTime
FROM q_roomut2
GROUP BY q_roomut2.SESSION, q_roomut2.MEET_BUILDING_CD1,
q_roomut2.MEET_ROOM_NR1;
```

q_roomut1

```
SELECT t_Course_Sched_Norm.SESSION,
t_Course_Sched_Norm.MEET_BUILDING_CD1,
t_Course_Sched_Norm.MEET_ROOM_NR1, t_Course_Sched_Norm.MEET_DAY1,
Hour([MEET_START_TM1]-t_Course_Sched_Norm!MEET_END_SUFFIX1) AS [Time],
t_Course_Sched_Norm.MEET_ALT_WEEKS1
FROM t_Course_Sched_Norm
WHERE (((t_Course_Sched_Norm.MEET_BUILDING_CD1) Is Not Null) AND
((t_Course_Sched_Norm.MEET_ROOM_NR1) Is Not Null) AND
((t_Course_Sched_Norm.MEET_ALT_WEEKS1)="E"));
```

q_roomut1

```
SELECT t_Course_Sched_Norm.SESSION,
t_Course_Sched_Norm.MEET_BUILDING_CD1,
t_Course_Sched_Norm.MEET_ROOM_NR1, t_Course_Sched_Norm.MEET_DAY1,
```

```
Hour([MEET_START_TM1]-t_Course_Sched_Norm!MEET_END_SUFFIX1)/2 AS
[Time], t_Course_Sched_Norm.MEET_ALT_WEEKS1
FROM t_Course_Sched_Norm
WHERE (((t_Course_Sched_Norm.MEET_BUILDING_CD1) Is Not Null) AND
((t_Course_Sched_Norm.MEET_ROOM_NR1) Is Not Null) AND
((t_Course_Sched_Norm.MEET_ALT_WEEKS1)="A"));
```

q_Make_Student Schedule

```
SELECT DISTINCT t_Course_Choices.SESSION, t_Course_Choices.COURSE1,
t_Course_Choices.COURSE2, t_Course_Choices.PERSON_ID,
t_Course_Choices.POST_CD, t_Course_Choices.PRIME_TEACH_METHOD,
t_Course_Choices.PRIME_SECTION_NR, t_Course_Schedule_no_rooms.MEET_DAY1,
t_Course_Schedule_no_rooms.MEET_START_TM1,
t_Course_Schedule_no_rooms.MEET_END_SUFFIX1 INTO t_Student_Schedule
FROM t_Course_Schedule_no_rooms INNER JOIN t_Course_Choices ON
(t_Course_Schedule_no_rooms.TEACH_METHOD =
t_Course_Choices.PRIME_TEACH_METHOD) AND
(t_Course_Schedule_no_rooms.SECTION_NR =
t_Course_Choices.PRIME_SECTION_NR) AND
(t_Course_Schedule_no_rooms.COURSE1 = t_Course_Choices.COURSE1) AND
(t_Course_Schedule_no_rooms.SESSION = t_Course_Choices.SESSION);
```

q_make no rooms

```
SELECT DISTINCT t_Course_Sched_Norm.SESSION,
t_Course_Sched_Norm.COURSE1, t_Course_Sched_Norm.COURSE2,
t_Course_Sched_Norm.SECTION_NR, t_Course_Sched_Norm.TEACH_METHOD,
t_Course_Sched_Norm.MEET_DAY1, t_Course_Sched_Norm.MEET_START_TM1,
t_Course_Sched_Norm.MEET_END_SUFFIX1 INTO t_Course_Schedule_no_rooms
FROM t_Course_Sched_Norm;
```

q_make_course_schedules

```
SELECT [Course Schedules].SESSION, [Course Schedules].COURSE1, [Course
Schedules].COURSE2, [Course Schedules].SECTION_NR, [Course
Schedules].TEACH_METHOD, [Course Schedules].PRIM_TEACH_METHOD, [Course
Schedules].MEET_BUILDING_CD1, [Course Schedules].MEET_ROOM_NR1, [Course
Schedules].MEET_ROOM_SUFFIX1, [Course Schedules].MEET_DAY1, [Course
Schedules].MEET_START_TM1, [Course Schedules].MEET_END_SUFFIX1, [Course
Schedules].MEET_ALT_WEEKS1 INTO t_Course_Sched_Norm
FROM [Course Schedules]
WHERE ((([Course Schedules].MEET_DAY1) Is Not Null) AND (([Course
Schedules].MEET_START_TM1) Is Not Null));
```

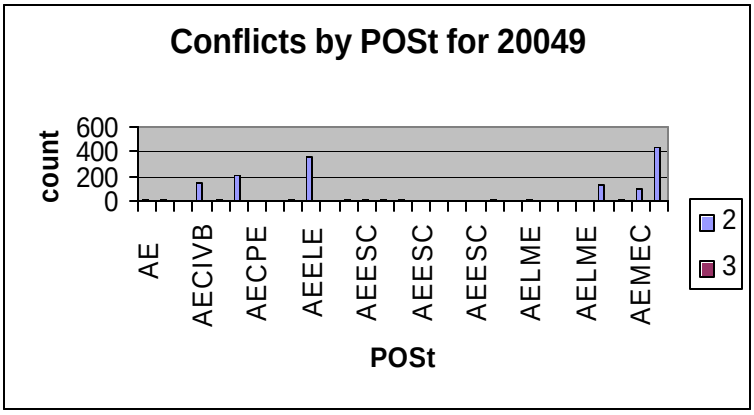
q_make course choices

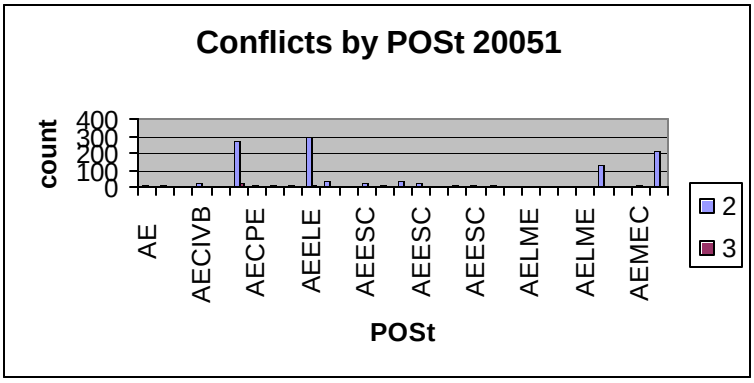
```
SELECT [Course Choices].SESSION AS Expr1, Left([COURSE],8) AS COURSE1,
Right([COURSE],1) AS COURSE2, [Course Choices].PERSON_ID AS Expr2, [Course
Choices].POST_CD AS Expr3, [Course Choices].PRIME_TEACH_METHOD AS Expr4,
[Course Choices].PRIME_SECTION_NR AS Expr5, [Course
Choices].OTH_TEACH_METHOD1 AS Expr6, [Course Choices].OTH_SECTION_NR1
AS Expr7, [Course Choices].OTH_TEACH_METHOD2 AS Expr8, [Course
Choices].OTH_SECTION_NR2 AS Expr9, [Course Choices].PRIMARY_ORG_CD AS
Expr10, [Course Choices].SECOND_ORG_CD AS Expr11, [Course
Choices].STUDENT_STATUS_CD AS Expr12, [Course Choices].ADMIN_ORG_CD AS
Expr13, [Course Choices].REG_STS_CD AS Expr14, [Course
Choices].YEAR_OF_STUDY AS Expr15, [Course Choices].Program AS Expr16 INTO
t_Course_Choices
FROM [Course Choices];
```

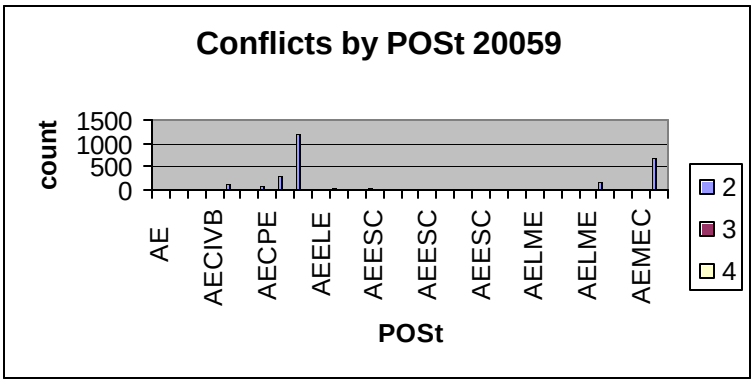
Appendix B: Quality Metric Charts

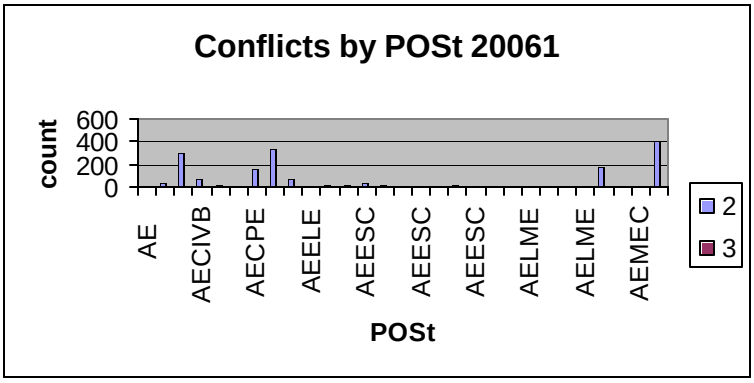
The following are the charts output by the evaluations database. For the charts that are grouped by POSt, the abbreviations are as follows:

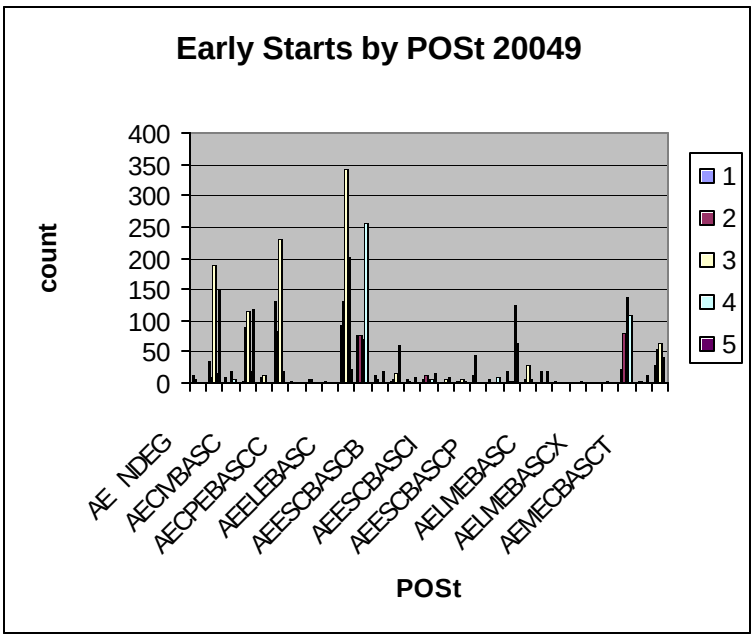
- AE followed by a space refers to a non degree program
- AECIV refers to a civil engineering POSt
- AECPE refers to a computer engineering POSt
- AEELE refers to an electrical engineering POSt
- AEESC refers to an engineering science POSt
- AELME refers to a mineral engineering POSt
- AEMEC refers to a mechanical engineering POSt
- AECHE refers to a chemical engineering POSt
- AEIND refers to an industrial engineering POSt

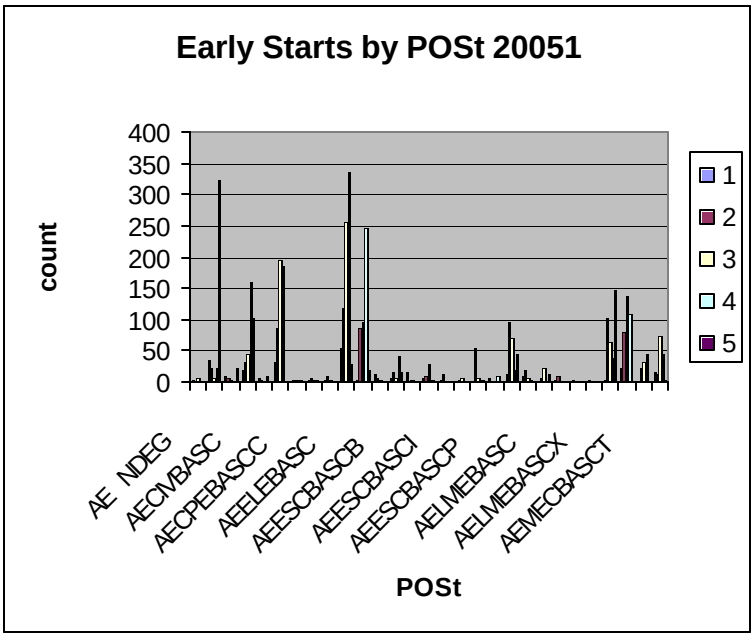


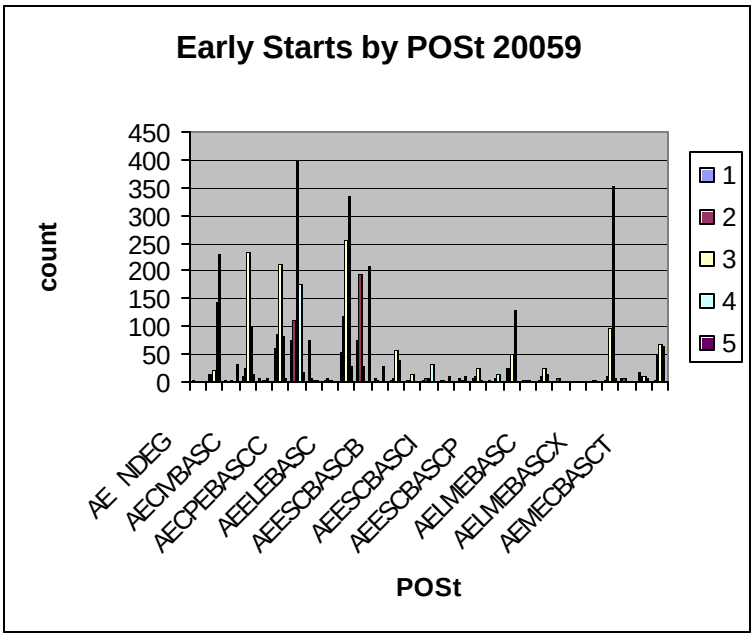


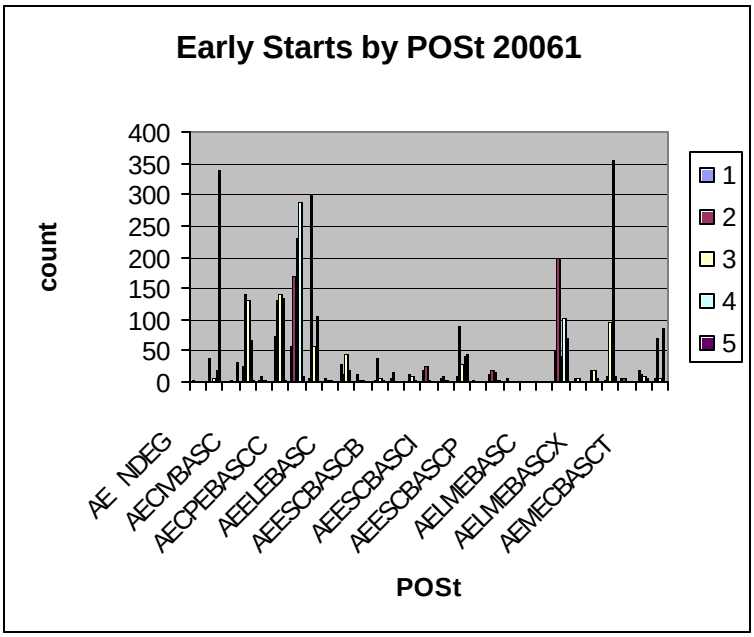

Conflicts by POSt 20059
count
1500
1000
500
0
AE
AECIVB
AECPE
AEELE
AEESC
AEESC
AEESC
AELME
AELME
AEMEC
POSt
2
3
4

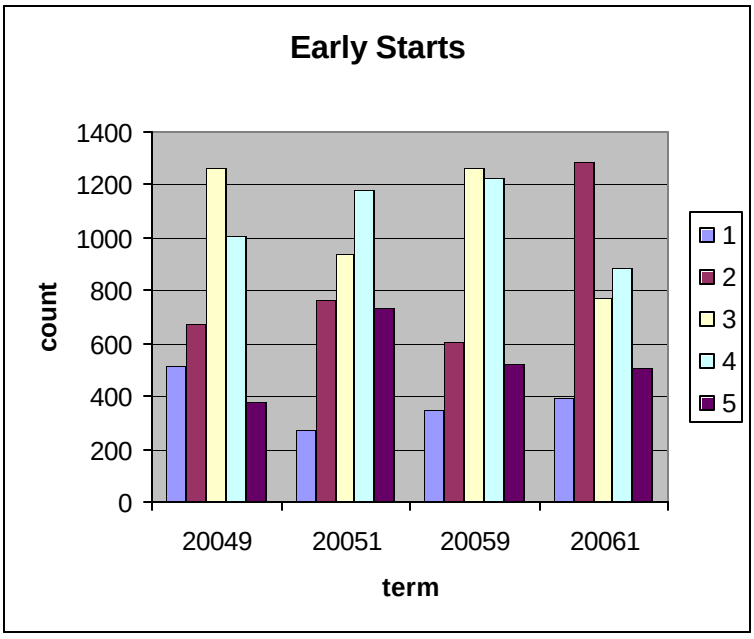

Conflicts by POSt 20061
count
600
400
200
0
AE
AECIVB
AECPE
AEELE
AEESC
AEESC
AEESC
AELME
AELME
AEMEC
POSt
2
3

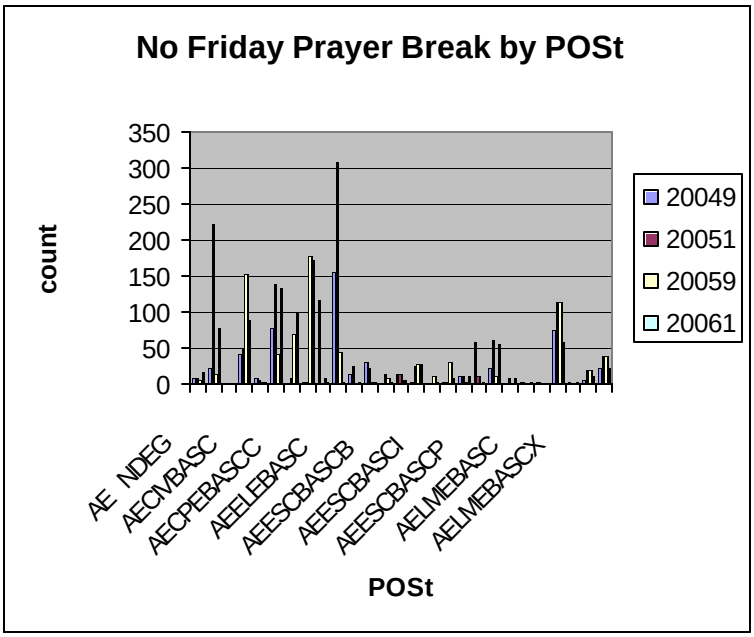

Early Starts by POSt 20049
count
400
350
300
250
200
150
100
50
0
AE_NDEG
AECMBASC
AECPEBASCC
AEELEBASC
AEESCBASCB
AEESCBASCI
AEESCBASCP
AELMEBASC
AELMEBASCX
AEMECBASCT
POSt
1
2
3
4
5

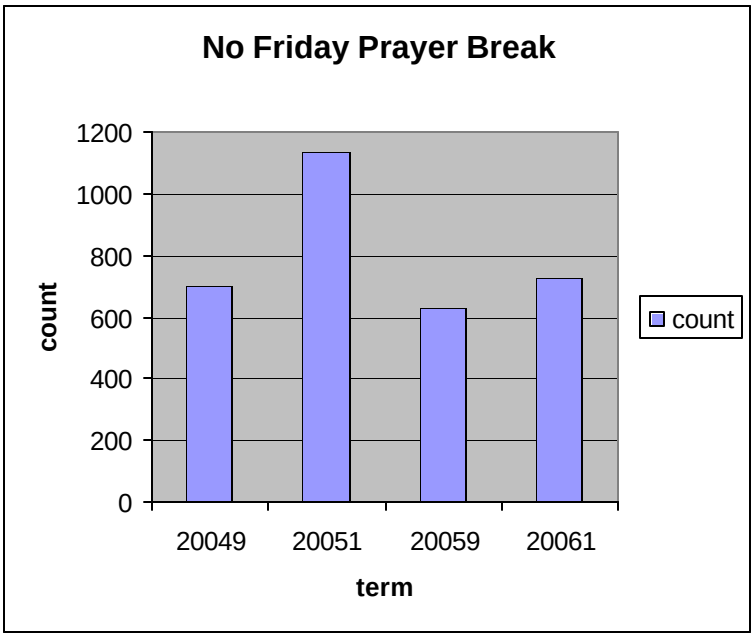

Early Starts by POSt 20051
count
400
350
300
250
200
150
100
50
0
1
2
3
4
5
AE_NDEG
AECMBASC
AECPEBASCC
AEELEBASC
AEESCBASCB
AEESCBASCI
AEESCBASCP
AELMEBASC
AELMEBASCX
AEMECBASCT
POSt

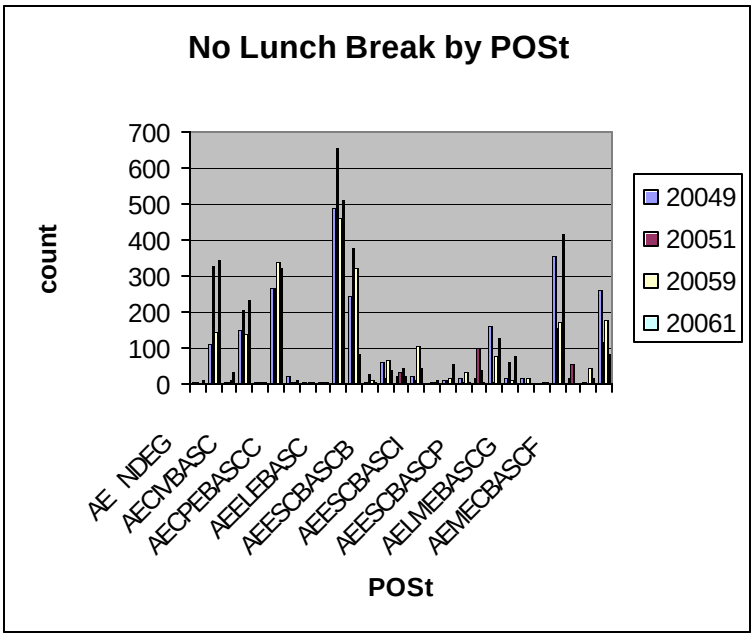

Early Starts by POSt 20059
count
450
400
350
300
250
200
150
100
50
0
1
2
3
4
5
AE_NDEG
AECMBASC
AECPEBASCC
AEELEBASC
AEESCBASCB
AEESCBASCI
AEESCBASCP
AELMEBASC
AELMEBASCX
AEMECBASCT
POSt

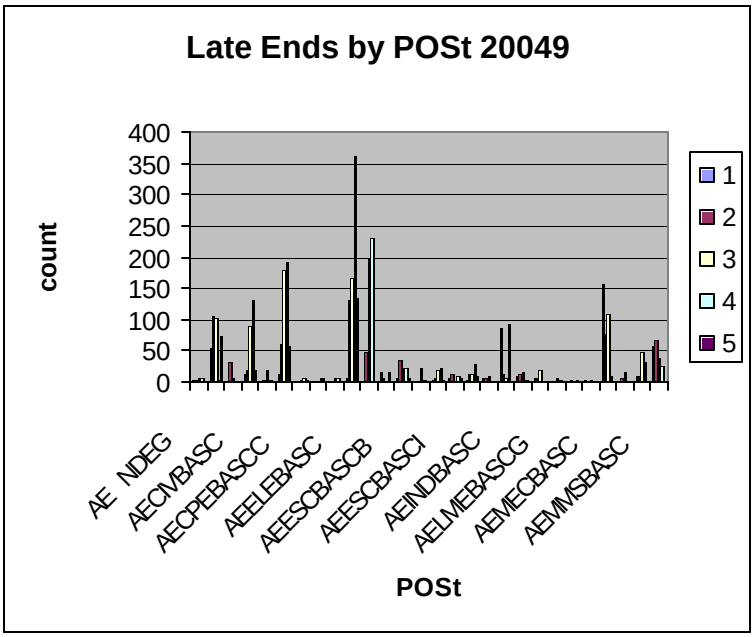

Early Starts by POSt 20061
count
400
350
300
250
200
150
100
50
0
AE_NDEG
AECMBASC
AECPEBASCC
AEELEBASC
AEESCBASCB
AEESCBASCI
AEESCBASCP
AELMEBASC
AELMEBASCX
AEMECBASCT
POSt
1
2
3
4
5

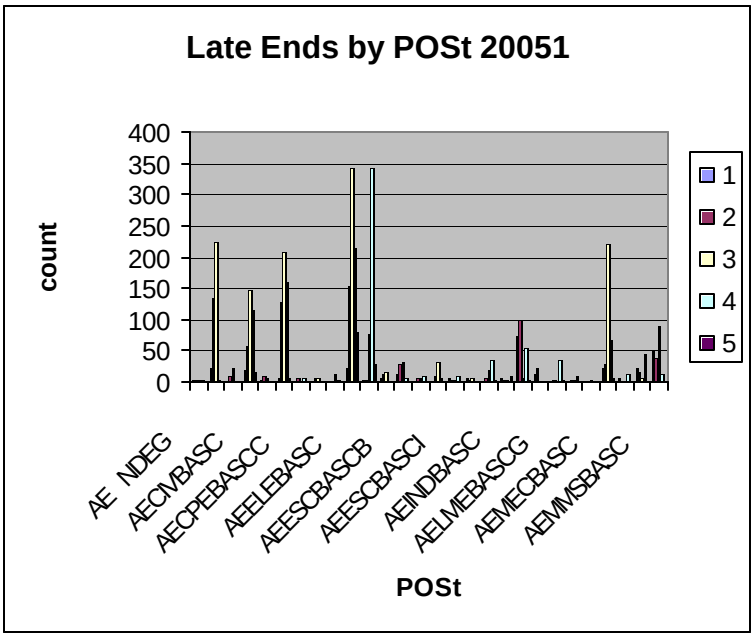

Early Starts
count
1400
1200
1000
800
600
400
200
0
20049
20051
20059
20061
term
1
2
3
4
5

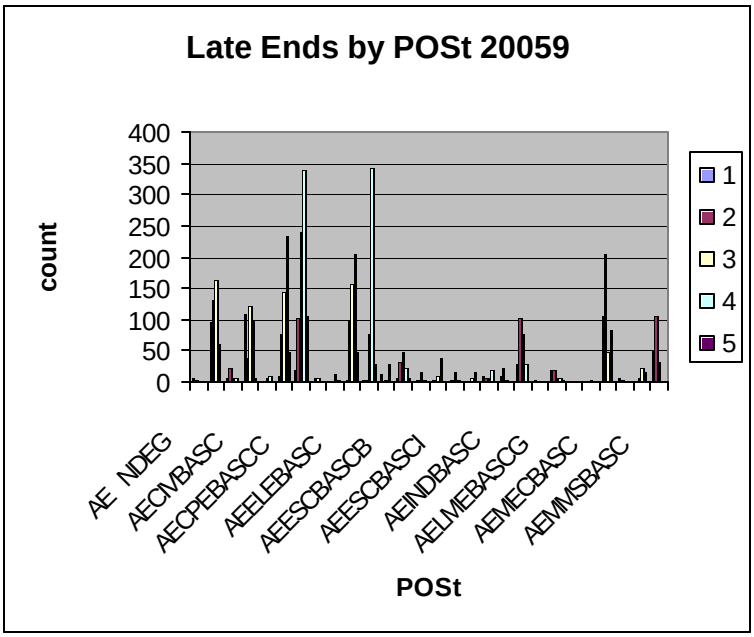

No Friday Prayer Break by POSt
count
350
300
250
200
150
100
50
0
AE_NDEG
AECMBASC
AECPEBASCC
AEELEBASC
AEESCBASCB
AEESCBASCI
AEESCBASCP
AELMEBASC
AELMEBASCX
POSt
20049
20051
20059
20061

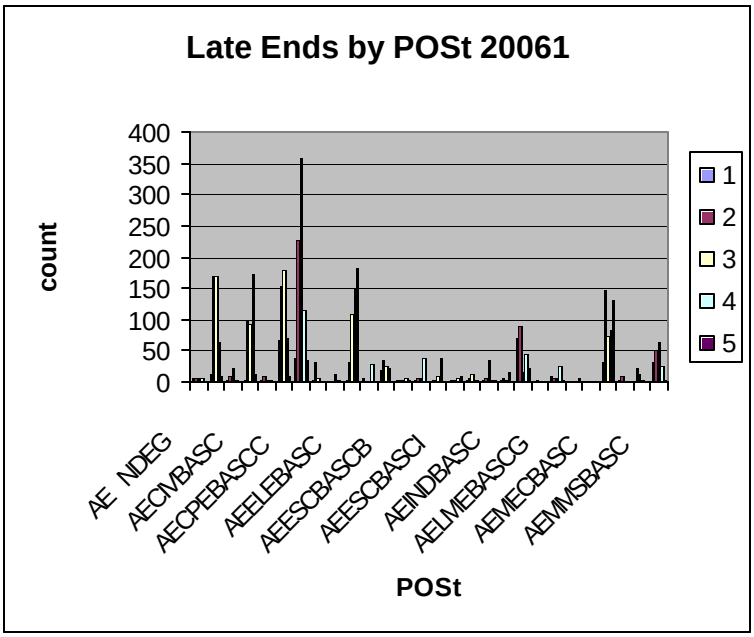

No Friday Prayer Break
count
1200
1000
800
600
400
200
0
20049
20051
20059
20061
term
count

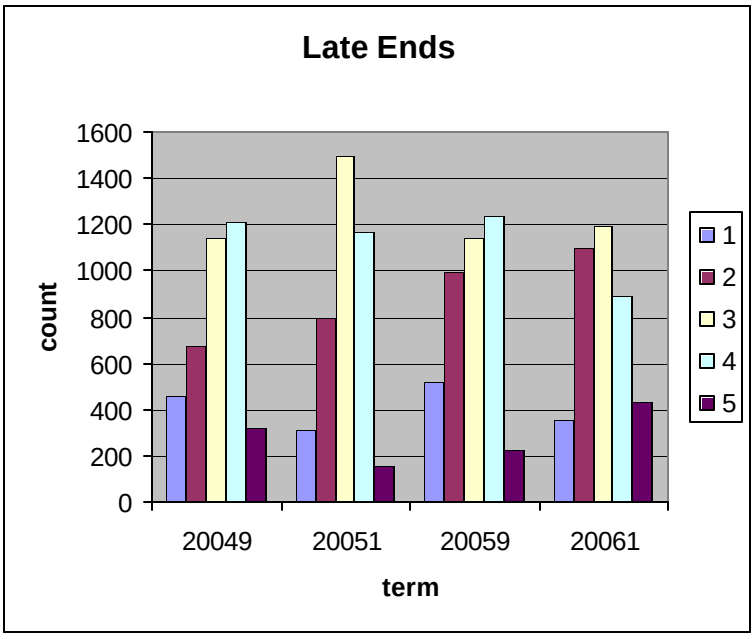

No Lunch Break by POSt
count
700
600
500
400
300
200
100
0
20049
20051
20059
20061
AE_NDEG
AECMBASC
AECPEBASCC
AEELEBASC
AEESCBASCB
AEESCBASCI
AEESCBASCP
AEESCBASCG
AELMEBASCG
AEMECBASCF
POSt

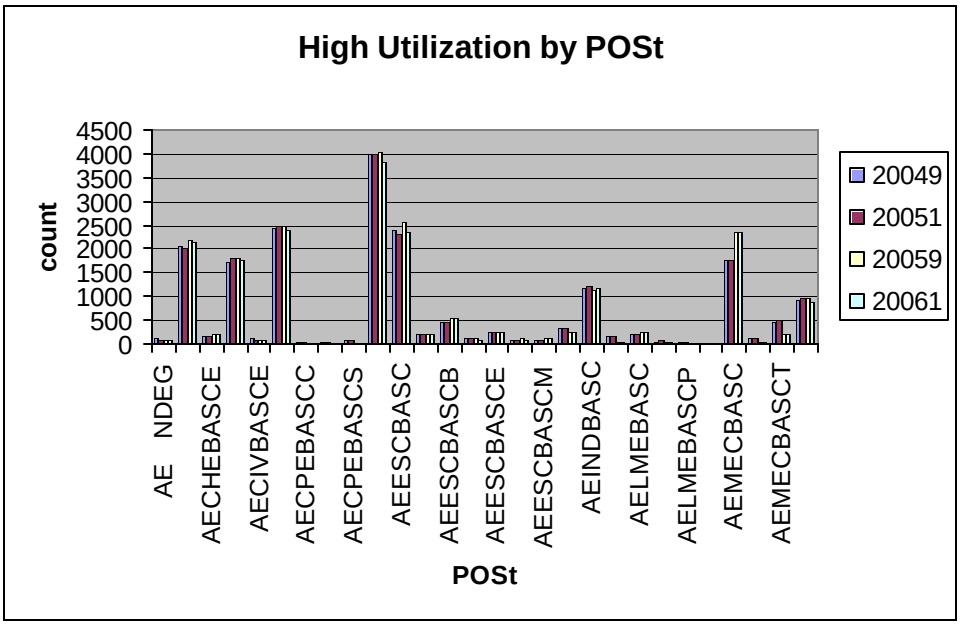

Late Ends by POSt 20049
count
400
350
300
250
200
150
100
50
0
1
2
3
4
5
AE_NDEG
AECMBASC
AECPEBASCC
AEELEBASC
AEESCBASCB
AEESCBASCI
AEINDBASC
AELMEBASCG
AEMECBASC
AEMMSBASC
POSt

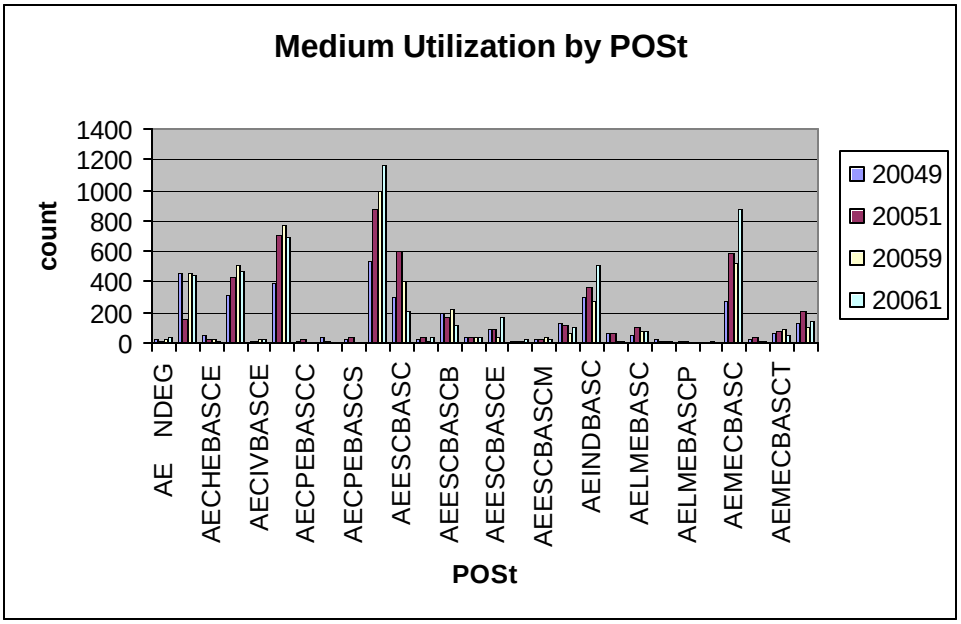

Late Ends by POSt 20051
count
400
350
300
250
200
150
100
50
0
1
2
3
4
5
AE_NDEG
AECMBASC
AECPEBASCC
AEELEBASC
AEESCBASCB
AEESCBASCI
AEINDBASC
AELMEBASCG
AEMECBASC
AEMMSBASC
POSt

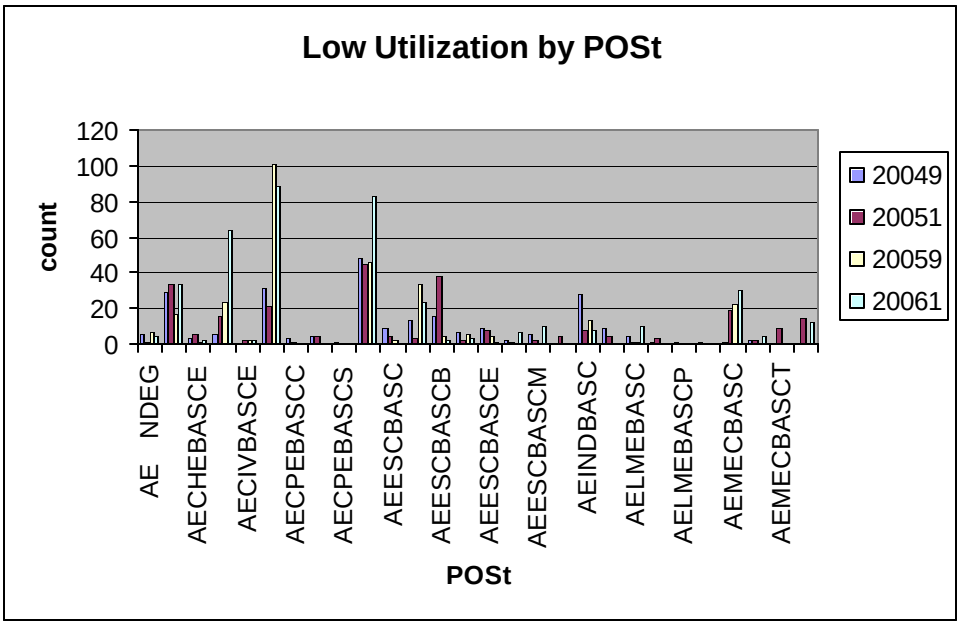

Late Ends by POSt 20059
count
400
350
300
250
200
150
100
50
0
1
2
3
4
5
AE_NDEG
AECMBASC
AECPEBASCC
AEELEBASC
AEESCBASCB
AEESCBASCI
AEINDBASC
AELMEBASCG
AEMECBASC
AEMMSBASC
POSt

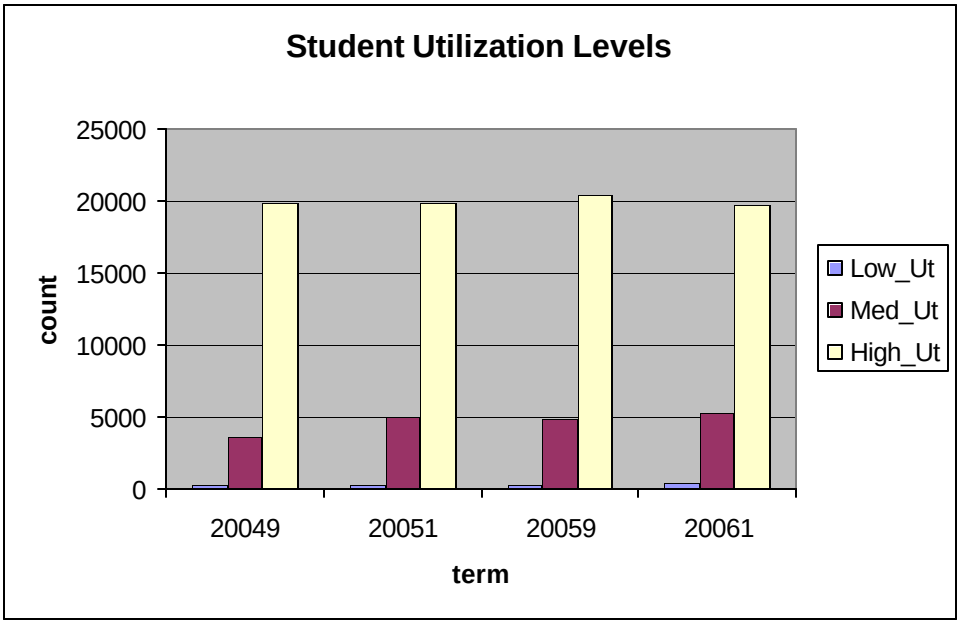

Late Ends by POSt 20061
count
400
350
300
250
200
150
100
50
0
AE_NDEG
AECMBASC
AECPEBASCC
AEELEBASC
AEESCBASCB
AEESCBASCI
AEINDBASC
AELMEBASCG
AEMECBASC
AEMMSBASC
POSt
1
2
3
4
5

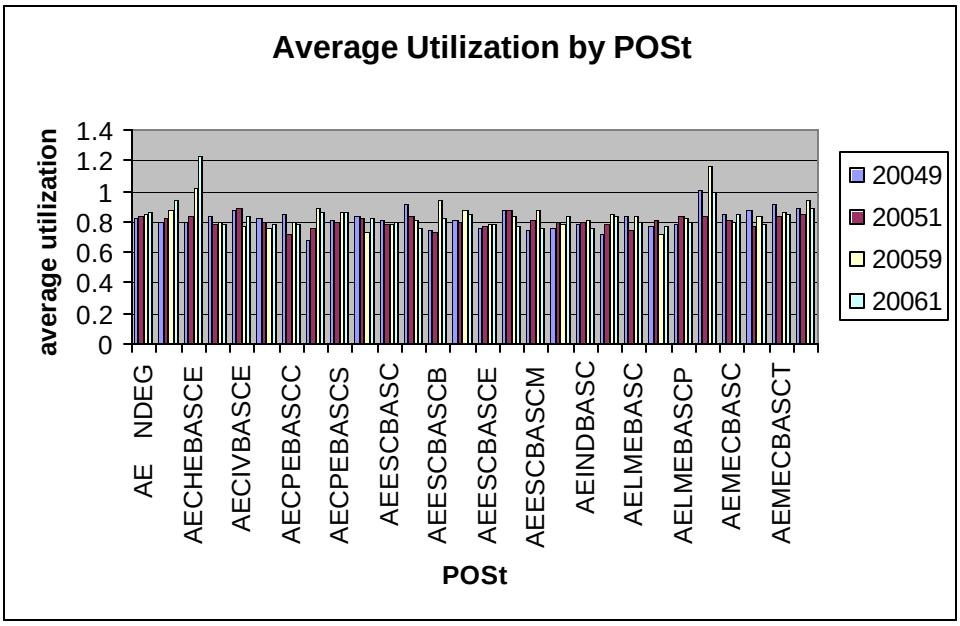

Late Ends
count
1600
1400
1200
1000
800
600
400
200
0
20049
20051
20059
20061
term
1
2
3
4
5

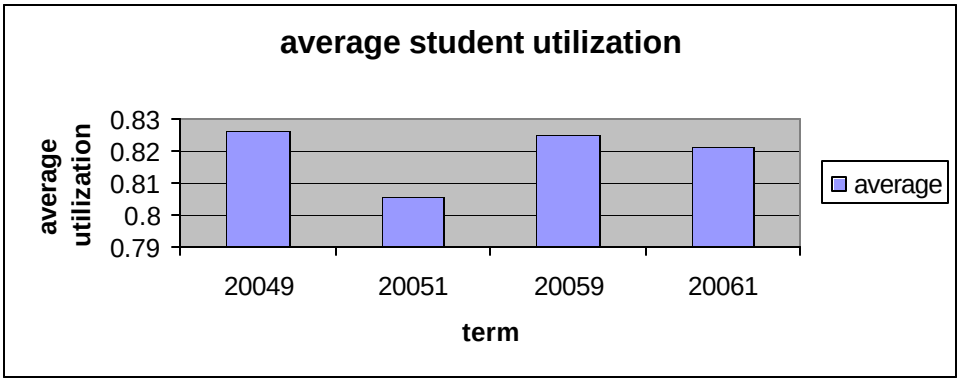

High Utilization by POSt
count
4500
4000
3500
3000
2500
2000
1500
1000
500
0
AE NDEG
AECHEBASCE
AECIVBASCE
AECPEBASCC
AECPEBASCS
AEESCBASC
AEESCBASCB
AEESCBASCE
AEESCBASCM
AEINDBASC
AELMEBASC
AELMEBASCP
AEMECBASC
AEMECBASCT
POSt
20049
20051
20059
20061


Medium Utilization by POSt
count
1400
1200
1000
800
600
400
200
0
AE NDEG
AECHEBASCE
AECIVBASCE
AECPEBASCC
AECPEBASCS
AEESCBASC
AEESCBASCB
AEESCBASCE
AEESCBASCM
AEINDBASC
AELMEBASC
AELMEBASCP
AEMECBASC
AEMECBASCT
POSt
20049
20051
20059
20061


Low Utilization by POSt
120
100
80
60
40
20
0
count
20049
20051
20059
20061
AE NDEG
AECHEBASCE
AECIVBASCE
AECPEBASCC
AECPEBASCS
AEESCBASC
AEESCBASCB
AEESCBASCE
AEESCBASCM
AEINDBASC
AELMEBASC
AELMEBASCP
AEMECBASC
AEMECBASCT
POSt


Student Utilization Levels
25000
20000
15000
10000
5000
0
count
Low_Ut
Med_Ut
High_Ut
20049
20051
20059
20061
term


Average Utilization by POSt
average utilization
1.4
1.2
1
0.8
0.6
0.4
0.2
0
AE NDEG
AECHEBASCE
AECIVBASCE
AECPEBASCC
AECPEBASCS
AEESCBASC
AEESCBASCB
AEESCBASCE
AEESCBASCM
AEINDBASC
AELMEBASC
AELMEBASCP
AEMECBASC
AEMECBASCT
POSt
20049
20051
20059
20061


average student utilization
average utilization
0.83
0.82
0.81
0.8
0.79
20049
20051
20059
20061
term
average