THE SMART SCHEDULER: A REVOLUTIONARY SCHEDULING SYSTEM FOR SECONDARY SCHOOLS

by

TIMOTHY LUKE MUGGY

B.S., University of Nebraska-Lincoln, 2006

A THESIS

submitted in partial fulfillment of the requirements for the degree

MASTER OF SCIENCE

Department of Industrial and Manufacturing Systems Engineering
College of Engineering

KANSAS STATE UNIVERSITY
Manhattan Kansas

2011

Approved by:

Major Professor
Dr. Todd Easton

Abstract

Westside High School (WHS) of Omaha, Nebraska utilizes a novel scheduling system called Modular scheduling. This system offers numerous advantages over the standard school day in terms of student learning and faculty development. Modular Scheduling allows teachers to design the structure of their own classes by adjusting the frequency, duration and location of each of their daily lessons. Additionally, teachers are able combine their classes with those of other teachers and team-teach. Modular scheduling also allows for open periods in both students' and teachers' schedules. During this time, students are able to complete school work or seek supplemental instruction with a teacher who is also free. Teachers are able to use their open mods to plan, meet in teams and help students who have fallen behind.

Currently, a semester's class schedules are constructed over the course of a seven week period by a full-time employee using a computer program developed in FORTRAN®. The process is extremely tedious and labor intensive which has led to considerable wasted time, cost and frustration.

This thesis presents a novel scheduling program called the SMART Scheduler that is able to do in seconds what previously took weeks to accomplish. Once parameters have been input, The SMART Scheduler is able to create cohesive class schedules within a modular environment in less than 6 seconds. The research presented describes the steps that were taken in developing the SMART Scheduler as well as computational results of its implementation using actual data provided by WHS. The goal of this research is to enable WHS and other schools to efficiently and effectively utilize modular scheduling to positively affect student learning.

Contents

List of Figures

Chapter 1

Introduction

Scheduling theory broadly refers to the assignment of tasks to a set of resources within a range of time. This concept has been explored throughout most of human existence through waging wars, constructing buildings and running businesses. This thesis presents the Sequential Modular Algorithmic Routines for Timetabling, or the SMART Scheduler. The SMART Scheduler is able to schedule classes of a large secondary school within a modular environment in less than 6 seconds.

A major contributor to initiating the mathematical study of scheduling theory was the development of plant shut-down schedules for DuPont in 1957 (Wong 1964). In 1959, James Kelley Jr. and Morgan Walker published a paper on their critical path scheduling algorithm that saved the company a projected $1 million. With the rise of computational power over the subsequent 50 years, the ability to solve large scheduling problems has become not only possible, but necessary for the success of many modern processes.

Current research has focused on developing techniques for solving large stochastic scheduling problems with original constraints (Bidot 2009). For instance, the scheduling of nurses in a hospital has been found extremely difficult as the availability and need for nurses fluctuates erratically. For this reason, scheduling procedures and software has been created to automate this task (Dowsland 2000). Scheduling research is also being conducted in

manufacturing to save money and optimize operation. Assembly-line scheduling enables the management of the unique abilities of each man or machine to be considered when forecasting maximal throughput of a product to satisfy its demand (Goncalves 2005). For products with short life-spans, production and distribution schedules must be considered together to maximize efficiency (Geismar 2008). Still further research within the scheduling theory field is being conducted in class and exam scheduling within a school setting (Beligiannis 2008, Lewis 2008, Al-Yakoob 2007, Datta 2007) . These problems receive attention not only for their applicability, but also their difficulty to solve. This thesis focuses on this particular scheduling problem.

Scheduling problems are frequently modeled using constraints and an objective function. Constraints can be thought of as limitations on the feasibility of solutions. For instance, if a feasible shift schedule for a set of employees requires that every employee work at least 20 hours per week, a constraint could be constructed to guarantee all feasible solutions have this trait. Constraints are often labeled hard or soft depending on their importance. A hard constraint is used to govern traits of a solution that must be satisfied in every solution, whereas soft constraints manage traits that are desirable but not required. An example of this can be seen in the scheduling of flight crews aboard commercial airplanes. It might be desirable to have two flight attendants on each flight where only one is absolutely necessary. A hard constraint would be constructed to insure one flight attendant is scheduled for each flight and a soft constraint would be constructed to attempt to schedule two whenever possible.

Objective functions measure the quality of a solution. They also outline the goal of the schedule. Objective functions can be constructed to seek the maximum profit or customer satisfaction for an organization. Other times, objective functions are built to minimize problem-

specified “costs”. Costs can be measured in terms of money, time, tardiness and a host of other units.

The majority of modern day secondary schools in the United States share the same general template for scheduling classes. Each day is divided into distinct time-windows called *periods*. Each period has the same length of time. Students attend the same classroom, with the same teacher at the same time each day. Unfortunately, this can lead to wasted class time. If a lesson on a certain day is particularly short, students often sit idly at their desks waiting to move on to the next class. A larger implication of fixed class lengths is that if a student requests two classes that are scheduled at the same time, the student is simply out of luck and is forced to drop one of their requests. This is an unfortunate missed opportunity for the student to learn a new skill or explore a potential career. Furthermore, the inability to take requested classes could force a student to graduate later than necessary.

In most current systems, students have no open, or free periods during the day. From an educational psychology standpoint, this can be damaging as the student is given no time for information to soak in before a new topic is presented (Khazzaka 1997). In addition, if a student would like to receive one-on-one attention from their instructor, they must wait until after school. The inevitability that some students will not understand a concept as quickly as others forces slower working students to be implicitly punished by having to stay at school longer than other students and may not be feasible. From the perspective of a teacher, working with students after school is a great opportunity to provide individualized instruction. However, it does force a teacher to give up time dedicated to planning future lessons or leaving school within contracted time.

A recent trend in the educational community is to emphasize inter/intra-departmental collaboration among faculty (Gajda 2008). The concepts of standardized curriculum and equal education for each student often suggest that every teacher of a class should give similar lessons. In increasing numbers, schools are scheduling mandatory times when teachers must create lessons together either during or after school hours. The agreed upon lessons are then delivered separately with one teacher and a group of students. Optimistic educational philosophy says that if a group of teachers construct lessons together, creative ideas will be implemented leading to increased student performance (Goddard 2007). However, other research suggests that poorly managed collaboration often causes micro-politics and power struggles that result in wasted time and frustration (Kelchtermans 2006).

Because teaching-styles differ greatly between instructors, a collaboratively created lesson often results in a dull "average" of the contributing styles. This can be attributed to the fact that while the lesson was created as a group, it is presented individually and teachers may alter or skip agreed-upon sections of the lesson. Furthermore, some veteran teachers refuse to change the way they have delivered a lesson over the course of their career (DePaul 2000). The end product of modern day collaboration is often the same lessons that would be given without working together and a waste of the time set aside by administrators (Kelchtermans 2006).

1.1 Westside High School's Unique Scheduling System

Westside High School of Omaha, Nebraska utilizes a unique system dubbed "Modular Scheduling" that resolves several of the difficulties of the standard class schedule. First of all, periods do *not* have fixed length as some are 40 minutes long while others are 20 minutes long. This fluctuation allows lessons to have varying lengths that change each day. Specifically, each

lesson may be 20, 40, 60 or 80 minutes in length. The advantage of classes having differing lengths is that some lessons simply don't require as much time as others. For instance, performing a laboratory experiment may require significantly more time than taking a quiz in a Spanish class.

Modular scheduling allows students to enjoy open periods throughout the day. During open periods, which may be as small as 20 minutes , students are confined to the school building but may eat lunch, work on homework, or seek help from teachers who also have that period open. Students may also dedicate time to extra-curricular activities during their open periods such as building scenery for a theater performance or decorating for an upcoming pep-rally. The ability to get this work done during the school day frees time outside of school for part-time jobs, athletics, clubs and family time.

Modular scheduling not only allows classes to have varying lengths, but also to be offered at different times each day. For instance, a student may have Photography class during period 2 on Monday, but periods 5 and 6 on Tuesday, and so on. The reasoning behind this change in schedule is that lessons may be slotted for periods that maximize students being able to attend the class. For instance, if several students who request U.S. History also request Chemistry and Geometry, it is important to schedule sections of these classes apart from each other so that as many students as possible can attend all of them. Currently, Westside does such a good job of satisfying student class requests that their average graduate receives an entire semester's worth of education *greater* than that of a graduate from a surrounding high school. This is an incredible advantage for their students.

Modular scheduling also allows teachers to *fuse* their lessons together so that more than one group of students (and teachers) can meet in the same room, at the same time and be presented the same lesson. Individuals and groups of teachers often bring multiple classes together so that all of their students receive the same lesson at the same time. Instructors then deliver lessons together or in a rotation. This type of grouping usually occurs in an auditorium or theater. If a teacher has written a particularly creative or interactive lesson, other teachers may supervise students while the lesson is delivered by its creator. By fusing classes together, students are able to experience multiple teaching styles. Currently, Westside offers a novel classroom experience that is co-taught by the English and Social Science departments. Students of both Literature and United States History are brought together throughout the week for lessons that overlap both subjects.

Each section of a class offered at Westside has a structure of meetings independent of other sections. In other words, each teacher is given the freedom to design their own class with regards to several parameters to be outlined later in this thesis. A few of the parameters deal with the frequency and duration of meetings, time constraints on when these meetings take place, and room constraints that govern where each lesson is delivered. In this way, every educator can design the class in which they feel will aid in student learning. This is a liberty that is unheard of in almost every other school system.

1.2 Motivation

The challenge that Westside faces is a lack of technology capable of autonomously constructing a schedule of classes within the structure of each class. Presently, a former math teacher requires 7 weeks of full time work to produce a semester's class and student schedules.

This employee uses a FORTRAN® computer program developed in 1985 that presents class structures as giant conflict matrices to be inspected and interpreted. The current system requires not only a waste of time and money, but also a waste of a good mathematics teacher, who would better serve students in the classroom.

As a former mathematics teacher at Westside High School, I have experienced the benefits of modular scheduling firsthand. During my time at Westside, I enjoyed being able to meet with students outside of class and during school hours for homework help or to get them caught up on missing work. I also found the extra one-on-one time rewarding as I was able to pursue positive mentoring relationships with students needing direction. I have seen the rewards of modular scheduling in my students learning. Additionally, as a new teacher, I was able to fuse my classes with veteran teachers' providing valuable opportunities to learn from them. Modular scheduling permits one-of-a-kind collaboration among teachers leading to better instruction and more learning.

When I learned about the immense time and effort that is put toward a single schedule, I decided to lend my services in the form of research and this thesis. The SMART Scheduler that has been created over the past 2 years through this research is able to do in seconds what normally takes Westside weeks to accomplish. The motivation of this research is to ease the burden of modular scheduling at Westside and permit its growth to other schools.

1.3 Contributions

In collaboration with Westside High School, this thesis presents several tools for implementing a progressive system for the modularized scheduling of classes and students in a high school environment. This system is utilizes a computer program called the Sequential

Modular Algorithmic Routines for Timetabling, or SMART Scheduler. The SMART Scheduler contains algorithms that produce feasible modularized schedules in less than 6 seconds. In experimentation, the program was able to schedule 208 classes comprised of 3,036 individual meetings in 5.1 seconds.

These algorithms involved have been written in the c language to be used by schools who desire to implement this modular scheduling system. In addition, a website and database has been created to help students, faculty and parents access schedules from anywhere. The website also permits administration to alter each class's structure, teacher/student information, and students' class requests. To our knowledge, this is the first class scheduling computer system capable of scheduling modular classes in this modularized environment.

1.4 Outline

Chapter 2 is a literature review of research involving scheduling theory. In addition to the history of scheduling theory, several approaches and their corresponding techniques are described. These techniques often involve variations of exact and heuristical methods.

Chapter 3 begins with an in-depth description of the modular scheduling system with emphasis on the parameters that may be changed in constructing a class. Then, a procedure for scheduling classes with respect to their parameters and a body of students' requests is given. Finally, computational results of the system are presented using data provided by Westside High School.

Chapter 4 contains the tools of an interactive website that presents scheduling data to students, faculty and parents. The website gives usability to the schedules produced by the

SMART Scheduler by allowing people to view and sometimes modify the schedules of others. Furthermore, the website contains new means of managing profile and class-structure data from semester to semester.

Chapter 5 is a summary of this thesis' contributions and a conclusion. Potential research endeavors for the future is presented. They will focus on ways to improve the SMART Scheduler and modular scheduling as a whole.

Chapter 2

Background Information

This chapter provides a literature review of significant research in scheduling theory. The first section introduces scheduling theory history and briefly models two classic problems both qualitatively and mathematically. The next section describes the Timetabling Problem in depth. This section also explores popular solution techniques that have been applied specifically to class scheduling problems. The third section summarizes the chapter.

2.1 Classic Scheduling Problems

Scheduling theory has been divided into several distinct, however related categories. Each category is often understood through the explanation of one of its most generalized problems. The problems that most closely relate to the purpose of this thesis are The Job Shop Problem, The Vehicle Routing Problem and The Timetabling Problem. Solutions to each of these classic optimization problems are efficient schedules that manage the utilization of resources performing specific tasks in a way that all tasks are completed while optimizing some objective function.

Supply Chain Management often use scheduling theory in the manufacturing of products. The value of efficient manufacturing has led to the creation and study of the Job Shop Problem (Garey 1976); a generalized problem that attempts to schedule resources, be it man or machine, in a way that minimizes the time required to complete all jobs.

The inputs of the Job Shop Problem are a finite set of jobs $J = \{j_1, j_2, \dots, j_n\}$ and a finite set of processes $L_i = (l_1^{j_i}, l_2^{j_i}, \dots, l_s^{j_i})$ to be completed for each $j_i \in J$. Since individual jobs require unique processes in varied orders, let $M_i = (m_1^{j_i}, m_2^{j_i}, \dots, m_s^{j_i})$ be the machines or stations where L_i is performed to complete j_i. This is the order in which a product arrives at a station or machine. Let $p_{l_q}^{j_i}$ represent the time required to complete the q^{th} process of job j_i.

From the inputs described above, a model can be built by letting $x_{j_i l_q}$ be the starting time of the q^{th} process of job j_i. Additionally, let z be the total time required to complete all jobs. A model is then:

$$min \quad z$$

$$s.t. \quad x_{j_i l_q} \geq x_{j_i l_{q-1}} + p_{j_i l_{q-1}} \forall j_i \in J, l_q \in L \quad (1)$$

$$x_{j_{i+1} l_q} \geq x_{j_i l_q} + p_{j_i l_q} \ \forall j_i \in J, l_q \in L \quad (2)$$

$$z \geq x_{j_i l_q} + p_{j_i l_q} \forall j_i \in J, l_q \in L \quad (3)$$

$$x_{j_i l_q} \geq 0 \ \forall \ j_i \in \ J, l_q \in L \quad (4)$$

The first set of constraints insures that each job cannot be started until the previous job has had time to complete. The second constraint set makes sure that process l_q cannot contribute to another job before completing its current job. The third constraint assures that the total time to finish each job is at least the cumulative time required for each process within the job. The fourth and final constraint guarantees that each job has a non-negative starting time. The objective function seeks to minimize the amount of time required to complete all jobs.

While this is a general formulation, several variations of the *Job Shop Problem* have been developed for differing types of manufacturing. Some formulations seek to maximize

production throughput [Hanen 1994] while others seek to minimize the time required to complete all jobs [Della Croche 1995]. Solutions to each formulation have been found highly valuable in industry [Xia 2005,Colorni 1994, Fortemps 1997]. Because each manufacturing system presents unique challenges, algorithms have been developed for manufacturing chains with both single and parallel machines [Carlier 1989, Hurink 1993], fixed and flexible routes [Dai 2001, Chen 1999], and deterministic and stochastic flow times [Graham 1977, Lin 1997].

The Vehicle Routing Problem (VRP) is a classic scheduling problem in which a finite fleet of $V = \{v_1, v_2, \dots, v_p\}$ vehicles must visit *n* customers before the day is over. Let $S = \{s_0, s_1, s_2, \dots, s_n\}$ be this set of stops with s_0 representing the depot. Each vehicle v_k has a capacity, denoted $cap(v_k)$ and each stop s_i has a demand $dem(s_i)$ An originating location (or depot) s_0 is known at which all vehicles must start and finish routes. It is also necessary to know the distance (or cost) between each of the stops and from each stop to the depot. This is notated by $dist(s_i, s_j)\ \forall\ s_i, s_j \in S$. The goal of VRP is to identify the optimal routes for each vehicle to take so that each stop is visited at a minimal cumulative *cost*.

Formally, the generalized vehicle routing problem can be modeled in the following manner. Let $x_{i,j}^k$ be a binary variable that equals 1 if vehicle v_k makes stop s_j directly after stop s_i and 0 if not.

$$min \sum_{i:s_i \in S} \sum_{j:s_j \in S} \sum_{k:v_k \in V} dist(s_i, s_j)\, x_{i,j}^k$$

$$s.t. \quad \sum_{j:\, s_j \in S} \sum_{k:\, v_k \in V} x_{i,j}^k = 1 \ \ \forall s_i \in S \setminus \{s_0\} \qquad (1)$$

$$\sum_{j:\, s_j \in S} x_{0,j}^k = 1 \ \ \forall v_k \in V \qquad (2)$$

$$\sum_{i:\, s_i \in S} x_{i,0}^{k} = 1 \;\; \forall v_k \in V \qquad (3)$$

$$\sum_{i:\, s_i \in S} \sum_{j:\, s_j \in S} \frac{dem(s_j) x_{i,j}^{k}}{2} \leq cap(v_k) \;\forall\; v_k \in V \qquad (4)$$

$$x_{i,j}^{k} \in \{0,1\} \; \forall s_i, s_j \in S, v_k \in V \qquad (5)$$

The first constraint insures each stop is made exactly once by only one vehicle. The second constraint forces each vehicle to begin its route at the depot while the third constraint insures each vehicle ends its route there. The fourth constraint guarantees that the vehicle delivering to stop s_j has the capacity to meet that stop's demand. The final constraint restricts x_{ij} to binary values.

The objective function seeks to minimize the overall distance that the fleet must travel. Variations of these problems utilize objective functions that minimize costs in terms of the financial implications, time expended, or customer dissatisfaction. Each VRP model contains additional constraints regarding the unique demands and limitations of the fleet or cargo. Applications of vehicle routing problems can be found in identifying optimal locations of distribution centers [Cambell 1994], package delivery routes [Tan 1984], and international trading [Fagerholt 2004].

The VRP can also be modeled utilizing graph theory. Let *G=(S,E)* be a graph composed of $n+k$ distinct vertices. Let vertices $s_1, s_2, \ldots, s_n$ denote stops that must be visited and s_{n+1}, s_{n+2}, ,…, s_{n+k} represent a single depot at which all vehicles originate. The edge *{i,j}* represents that a vehicles travels from customer or depot *i* to *j*. A successful solution *x* to the Traveling Salesman Problem [Clark 1991] on this graph provides an optimal solution to the VRP. Let vehicle $v_i \in V$ be assigned the part of the tour between vertices s_{n+i} and s_{n+i+1}. In this way, all stops are made and all vehicles' routes begin and end at the depot.

2.2 The Timetabling Problem

The Timetabling Problem (TTP) is a combinatorial optimization problem consisting of scheduling a limited number of resources within pre-defined time windows. A successful time table assigns blocks of time such that every resource can complete its assigned tasks without disrupting other resources. The development of methodology to solve a unique variation of TTP is the concentration of this thesis.

TTP's inputs begin with a set of activities to be completed within a specified window of time. A set of resources is available to complete these activities but each resource is subject to availability constraints. Sets of hard and soft dependency relationships are established between activities. In this way, some activities cannot occur at the same time and some must be completed before others can begin. Each activity requires the same amount of time and is therefore scheduled within a set of fixed time windows.

Because of the broad nature of TTP, several variations have been constructed to fit unique situations. The scheduling of personnel in a work-environment is one classic example. In this instance, employees are needed to complete work but cost the company money on an hourly basis. One tries to find a schedule of employees that insures all necessary work is completed at minimal cost. Because demand is unforeseen in some businesses such as restaurants, schedules are often created that knowingly place more employees than necessary on a shift. It is then up to a manager to decide when employees are to be released.

A classical example of TTP can be found in the scheduling of school classes. First presented at the IFIP Congress (Gotlieb 1962), the Class-Scheduling Problem outlines a model that seeks to find a schedule of classes, teachers, and students that minimizes conflicts. Types of

conflicts come from teachers' availabilities, students' class requests, and the number of hours required per week for each class. Further restrictions can incorporate lesson-specific classrooms (i.e. Labs, Gymnasiums, etc.) in which certain meetings must be held.

Another example of Time Tabling can be found in the scheduling of tournaments (Easton, Nemhauser, Trick 2003). Often times, round robin tournament schedules are created so that teams play a set of opponents with a balanced load of home and away games. Constraints can be added to force teams to play a specified number of other teams from within or outside of their conference. Further constraints can be added to ensure television time for certain games.

All of the preceding examples of TTPs follow the same structure. Let $T = \{t_1, t_2, \ldots, t_m\}$ be a finite set of tasks to be completed. Tasks can further be broken down into individual processes. Let $B^i = \{b_1^i, b_2^i, \ldots, b_s^i\}$ be the set of processes required to complete task t_i and $H = \{h_1, h_2, \ldots, h_p\}$ be a finite set of available hours in which each task must be completed. Let $R = \{r_1, r_2, \ldots, r_n\}$ be a set of resources and A_j represent the hours in which r_j is available. Formally, let $a_{jk} = 1 \; \forall r_j \in R$ if resource j is available during hour k; and 0 if not.

Next, let C_q^i signify the set of hours in which process b_q^i is available to be performed. Thus, $c_{iqk} = 1$ if process b_q^i is available during hour k and 0 if not. This is particularly valuable when external forces affect the availability of materials, support, etc. Let $S = \{s_1, s_2, \ldots, s_z\}$ be the set of locations or sites where processes are performed. Let $P^i = \{P_1^i, P_2^i, \ldots P_w^i\}$ be the set of availabilities of "participants" (i.e. students, passengers, audience members, etc.) who desire to take part in task t_i.

Objective functions for TTPs often try to minimize the time necessary for all tasks to be completed. Other objective functions seek to minimize scheduling conflicts so that a maximum

number of participants P_u^i are able to attend $\mathrm{t_i} \in T\ \forall i \leq m$. In this case, Let *f(i,q,j,k,l)* represent the cost of assigning the q^{th} process of task t_i to resource r_j during hour h_k at site s_l.

To model this problem, let x_{ijkl}^q be a binary variable with a value of 1 if the q^{th} process of t_i is performed by r_j during hour h_k at site s_l and 0 if not. An integer program is then:

$$min \sum_{i:\, t_i \in T} \sum_{q:\, b_q^i \in B^i} f(i,q,j,k,l)\, x_{ijkl}^q$$

$$s.t.\quad x_{ijkl}^q \leq a_{jk} * c_{iqk}\ \forall \mathrm{b_q^i} \in B^i, \mathrm{t_i} \in T, h_k \in H, s_l \in S \qquad (1)$$

$$\sum_{q:\, b_q^i \in B^i} \sum_{k:\, h_k \in H} \sum_{l: s_l \in S} x_{ijkl}^q = d_{ij}\ \forall t_i \in \mathrm{T}, \mathrm{r_j} \in R \qquad (2)$$

$$\sum_{j: r_j \in R} \sum_{k:\, h_k \in H} \sum_{l: s_l \in S} x_{ijkl}^q \leq 1\ \forall \mathrm{t_i} \in T\ , \mathrm{b_q^i} \in B^i \qquad (3)$$

$$\sum_{j: r_j \in R} x_{ijkl}^q \leq 1\ \forall \mathrm{b_q^i} \in B^i, \mathrm{t_i} \in T, h_k \in H, s_l \in S \qquad (4)$$

$$x_{ijkl}^q \in \{0,1\}\ \forall t_i \in \mathrm{T}, \mathrm{r_j} \in R, h_k \in H, s_l \in S \qquad (5)$$

The first constraint forces each task to be performed at a time in which both the resource and the process of a task are available. The second constraint insures that all d_{ij} hours required for r_j to perform t_i are allocated. The third constraint insures that no more than one resource be assigned to a task at the same time. The fourth constraint requires that one resource be assigned to no more than one task at a given time. The final constraint restricts the values of x_{ijkl}^q to binary values. The objective function minimizes the overall cost of the schedule.

2.3 Approaches to the Timetabling Problem

The TTP is NP-Complete (Even 1975). The majority of the literature approaches the TTP from two chief perspectives, utilizing either exact or heuristic methods. Exact methods are

composed of algorithms often rooted in linear/integer programming that seek global optima. While finding and proving an absolute superior solution is desirable, the computation time can be so high that such algorithms become impractical for many real world scheduling problems.

Heuristic approaches, on the other hand, search for good feasible solutions (Qu 2009, Rossi Doria 2003). Because many problems have exponentially many feasible solutions, heuristic algorithms find solutions that are not proven to be optimal.

2.3.1 Exact Methods

Algorithms utilizing exact methods are used to find provably optimal solutions. The majority of these require exponential time to find a solution but some (i.e. Interior Point method) can find solutions in polynomial time given good circumstances. Two exact methods used to solve TTP that are relevant to this thesis involve graph theory and integer programs.

Graph colorings (Clark 1991) are applied as a way of grouping non-adjacent vertices and edges in a graph. Let *G=(V,E)* be a graph. A vertex coloring of *G* assigns a color to each vertex of *G* so that no two adjacent vertices have the same color. The chromatic number $\chi(G)$ is the minimum number of colors required to color *G*. It is important to note that determining $\chi(G)$ is NP-Complete (Sánchez-Arroyoa 1989).

It has been shown that a successful coloring of a special type of graph provides a necessary and sufficient condition for the existence of a feasible time table. This was done for the class-scheduling TTP by denoting vertex v_{ijkl}^{q} as the q^{th} meeting of class $c_i \in C$ taught by teacher $t_j \in T$ during hour $h_k \in H$ in room $s_l \in S$. Let r_{ij} be the number of meetings that t_j must teach c_i. In essence, each vertex in *G* represents an assignment of a meeting, a class and a

teacher to a specific hour and location. The edge set of G represents conflicts between nodes. In particular, edge $e = \{v_{ijkl}^{q}, v_{mnop}^{r}\} \in E$ if and only if the assignment of the q^{th} meeting of class c_i with teacher t_j in hour h_k *to* room s_l and the r^{th} meeting of class c_m with t_n to hour h_o in room s_p would cause one of the following conflicts:

1. A teacher is assigned to teach a class outside of either the class's or teacher's availabilities.

2. Two meetings are assigned to the same room at the same time.

3. Teacher meets more than one class at the same time.

When coloring such a graph, each color represents an hour in which a meeting can take place without conflict. Coloring the graph identifies $\chi(G)$, the minimum number of hours necessary to schedule all meetings successfully. If a successful $\chi(G)$-coloration of this graph exists, a feasible schedule can be made to the corresponding TTP where each color represents a specific period in which the class' meeting takes place. An interesting result is that $\chi(G) \leq k$ for some $k \in N$ if and only if a feasible solution exists for a corresponding TTP using at most k time-windows. In addition, there exists a one-to-one correspondence between the number of $\chi(G)$-colorations and the number of feasible time tables that can be created (Neufeld, 1974).

A $\chi(G)$-coloration and various scheduling problems can be solved using integer programs (IPs). IPs can be applied to identify the schedule that maximizes the number of participants that can be included. Integer Programming is an extension of linear programming with the additional restriction that all variables must be confined to integer values. For most practical applications,

these values are also required to be greater than or equal to zero. Integer programs are modeled as an objective function and a set of constraints as illustrated below.

$$max\ c^T x$$

$$s.t.\ a_j^T x_i \leq b_j \text{ for } j=1, 2,\ldots,m$$

$$x \in Z^n$$

Constructing an IP to solve a school's classes or exams can be done in the following manner. Let *C* be a set of classes, *T* be a set of identical teachers with identical availabilities, *H* be a set of hours, and *S* be a set of identical rooms. Let x_{ijkl} be a binary variable that equals one if class $c_i \in C$ is taught by $t_j \in T$ during hour $h_k \in H$ in room $s_l \in S$ and 0 if not. Additionally, let y_{mnop}^{ijkl} be a binary variable with a value of one if $x_{ijkl} = x_{mnop} = 1$ and 0 if not. Let d_{ij} signify the number of times that teacher $t_j \in T$ must teach class $c_i \in C$ during the week. Lastly, let p_{mnop}^{ijkl} be the "price" or cost of $x_{ijkl} = x_{mnop} = 1$. These costs can be adjusted so that some assignments are more/less favorable than others. Specifically, $p_{mnop}^{ijkl} = M$ for a suitably large *M* if the assignment forces infeasibility. An IP is then:

$$min \sum_{i:p_i \in P} \sum_{j:\, t_j \in T} \sum_{k:\, h_k \in H} \sum_{l:\, r_l \in R} p_{mnop}^{ijkl} * y_{mnop}^{ijkl}$$

$$s.t. \sum_{k:\, h_k \in H} \sum_{l:\, r_l \in R} x_{ijkl} = d_{ij}\ \forall c_i \in \mathrm{C}, \mathrm{t_j} \in T \qquad (1)$$

$$\sum_{i:p_i \in P} \sum_{j:t_j \in T} \sum_{l:\, r_l \in R} x_{ijkl} \leq 1\ \forall h_k \in H \qquad (2)$$

$$\sum_{j:t_j \in T} \sum_{k:\, h_k \in H} \sum_{l:s_l \in S} x_{ijkl} \leq 1\ \forall \mathrm{c_i} \in C \qquad (3)$$

$$2\ y_{mnop}^{ijkl} \geq x_{ijkl} + x_{mnop} - 1 \quad \forall c_i, c_m \in \mathrm{C}, \mathrm{t_j}, \mathrm{t_n} \in T, h_k, h_o \in H, s_l, s_p \in S \qquad (4)$$

$$2\, y_{mnop}^{ijkl} \leq\ x_{ijkl} + x_{mnop} \quad \forall c_i, c_m \in \mathrm{C}, \mathrm{t_j}, \mathrm{t_n} \in T, h_k, h_o \in H, s_l, s_p \in S \qquad (5)$$

$$x_{ijkl} \in \{0,1\}\ \forall c_i \in \mathrm{C}, \mathrm{t_j} \in T, h_k \in H, s_l \in S \qquad (6)$$

$$y_{mnop}^{ijkl} \in \{0,1\}\ \forall c_i, c_m \in \mathrm{C}, \mathrm{t_j}, \mathrm{t_n} \in T, h_k, h_o \in H, s_l, s_p \in S \qquad (7)$$

The first constraints insure all classes meet the correct number of times throughout the week. The second constraints make sure that no more than one resource be assigned to a certain task in the same hour. The third constraint restricts teachers to perform no more than one task in a given hour. The fourth and fifth constraints ensure that y_{mnop}^{ijkl} equals one if and only if both x_{ijkl} and x_{mnop} are equal to one. The final two constraints restrict the values of y_{mnop}^{ijkl} and x_{ijkl} to binary values. The objective function then seeks to minimize the cumulative cost of assignments. This model could be solved using *branch and bound* but it is unlikely to terminate in a reasonable amount of time.

Akkoyunlu (1999) produced an alternate method which models a simpler TTP as an integer program. Let *C* be a set of classes and *P* be a set of periods in a school day. The model denotes x_{ai} as a binary variable that takes the value 1 when class *a* is scheduled to be completed during period *i* and 0 otherwise. Let $y_{ai,bj}$ be a binary variable signifying lesson *a* being taught in period *i* and lesson *b* being taught in period *j*. Let $D^* = \{d_{ai}, d_{bj}, \dots, d_{zk}\}$ be the set of binary variables where $d_{ai} = d_{bj} = 1$ if and only if lesson *a* being assigned to period *i* is *compatible* with lesson *b* being assigned to period *j* and 0 otherwise. That is, each class is assigned no more than one period and every class is scheduled. Now, c_{a_i,b_j} represents the cost of assigning lesson *a* to period *i* and lesson *b* to period *j*. Costs of infeasible assignments, such as c_{a_i,a_j} which would schedule class *a* during both periods *i* and *j*, are set to infinity. Costs are then divided into

two distinct categories: $C_1 = \left\{ c_{a_i,b_j} \middle| c_{a_i,b_j} \leq 0 \right\}$ and $C_2 = \left\{ c_{a_i,b_j} \middle| c_{a_i,b_j} > 0 \right\}$. Finally, let W be a suitably large constant. Akkoyunlu 's model utilizes constraints similar to the model above but the objective function is slightly different:

$$min \sum_{a \in C, i \in P} (c_{ai,bj} - W) x_{ai} + \sum_{C_1} c_{ai,bj}(x_{ai} - y_{ai,bj}) + \sum_{C_2} c_{ai,bj} y_{ai,bj}$$

The first term adds together large negative costs that occur when class *a* is scheduled during period *i.* The second term adds a negative cost for every period class *b* is assigned to other than *j*. The third term adds all positive costs of assigning classes *a* and *b* to periods *i* and *j* respectively.

2.3.2 Non-exact Methods

While exact methods are ideal for identifying global optima, they typically require exponential computational effort. For large problems, vast amounts of memory are required and sometimes can take too long to solve. Because of these problems' computational intractability, non-exact methods have been created that search for good, but not provably optimal solutions. These methods often require less computational power and are called *metaheuristics,* most of which utilize some sort of a solution's neighborhood.

2.3.2-1 Neighborhoods.

Given a solution *x* to some problem *Π*, the neighborhood of *x, N(x)* is the set of solutions sufficiently close to *x*. Most heuristics start with a feasible solution to a problem and systematically search its neighborhood for another solution that improves an objective function.

The details of a neighborhood are user-specified but the following example illustrates the concept by encoding bit-strings similar to those stored in computers.

Suppose an employer needs to schedule a set *E* of three employees during an 8-hour work day. Let $H = \{h_1, h_2, \ldots, h_8\}$. It is required that at least one person is working in the shop at any given time and the objective is to minimize cost. The schedule $X = \{x_{ij} | i \in E, j \in H\}$ can be expressed as a set of three binary 8-tuples where a 1 signifies that the employee is working during a particular hour and a 0 otherwise. Also, let c_i be the hourly wage of employee *i*. An IP model is

$$\min \sum_{j:h_j \in H} c_i x_{ij}$$

$$s.t. \sum_{i:\, e_i \in E} x_{ij} \geq 1 \; \forall h_j \in H$$

$$x_{ij} = \{0,1\} \; \forall \; e_i \in E, h_j \in H$$

Consider the problem assuming $c_1=4$, $c_2=3$, and $c_3=3$ and let a starting solution allocate employee 1 during the first 4 hours, employee 2 during hours 4, 5 and 6, and employee 3 during hours 6, 7 and 8. This is encoded as $x = [x_1, x_2, x_3]$=[(1,1,1,1,0,0,0,0), (0,0,0,1,1,1,0,0), (0,0,0,0,0,1,1,1)]. The *Total Cost (TC) =34*. This is a feasible solution because it satisfies the sole constraint which requires someone to be in the shop at all times.

Define the neighborhood of a solution, *N(x), to* be the set of solutions that differ from *x* by exactly one bit. It is important to note that not all members of $N(x)$ are feasible. For example, y = [(**0**,1,1,1,0,0,0,0), (0,0,0,1,1,1,0,0), (0,0,0,0,0,1,1,1)] leaves the store empty in hour 1. Clearly *y'* =[(1,1,1,**0**,0,0,0,0), (0,0,0,1,1,1,0,0), (0,0,0,0,0,1,1,1)] is an element of $N(x)$. Furthermore, $|N(x)| \leq 24$ because some are of its potential neighbors are infeasible.

A natural step in many heuristics is to find the best neighbor. Clearly y' is a best neighbor with cost 30. A neighbor of y' is *z'* = [(1,1,1,0,0,0,0,0), (0,0,0,1,1,1,0,0), (0,0,0,0,0,**0**,1,1)] which eliminates the overlap during hour 6. The total cost of *z'* is 27. The solution z' represents a local optimal solution because every candidate neighboring solution is either infeasible or has a total cost higher than z'.

Observe that the solution z'' = [(1,1,0,0,0,0,0,0), (0,0,1,1,1,1,0,0), (0,0,0,0,0,0,1,1)] is feasible and has a total cost of 26. This solution has a better value than z' hence z' is not a global optimal solution. A global optimal solution in this case is solution z* = [(0,0,0,0,0,0,0,0), (1,1,1,1,0,0,0,0), (0,0,0,0,1,1,1,1)] with a cost of 24.

2.3.2-2 Metaheuristics

A meta-heuristic is an algorithm that searches for a "best" solution to a problem by identifying a candidate solution and trying to improve upon it with regard to the optimization of an objective function. This improvement can be done in a number of ways from searching neighborhoods of candidate solutions to modifying the candidate itself.

A popular metaheuristic is the *Hill-Climbing Algorithm* in which neighbors of a feasible solution are selected as candidate solutions only if they improve the objective function. Once, no such solution exists, the algorithm terminates. The limitation of such an algorithm is that it often gets stuck at local optima. Three common metaheuristics discussed in this section are Tabu Search, Genetic and Simulated Annealing algorithms.

2.3.2-2A Tabu Search

Tabu Search (TS) algorithms have been used in numerous types of optimization problems across several branches of research. Its applications have been found in the job-shop problem (Nowicki 1996, Dell'Amico 1993), vehicle routing (Gendreau 1994, Taillard 1997), and athletic team season and tournament schedules (Burke 2003, Hamiez 2001, Gaspero 2007).

Briefly, the TS algorithm begins with an initial feasible solution $x \epsilon X$, a well-defined neighborhood of x and a tabu list $T=\Phi$. Typically, a random neighbor, $x' \epsilon N(x)$, is chosen. If the operation to move from x to x' is not in T, then x' becomes the new solution and the inverse of this operation is added to T. If T becomes large, its oldest entry is removed. Similar to most metaheuristics, TS records and maintains the best current solution. The goal of the tabu list is to avoid becoming trapped at local minima by frequently selecting neighbors that worsen the objective function. The tabu list is useful as it prevents TS from cycling between fewer than $|T|$ solutions. Thus, a tabu search algorithm avoids trapping itself and should eventually find a near optimal solution.

Disadvantages of TS involve storing the tabu list as it can require a large amount of a computer's memory. Additionally, TS makes the move irrelevant of the objective function. Thus, the best reported solution may not be of utmost quality if the algorithm is given too few iterations to complete a thorough search.

A variety of researchers have used TS in timetabling problems. Gaspero (2001) utilized the TS algorithm to schedule university exams. His objective function introduces costs assigned to each variable depending on whether all constraints pertaining to that variable are at a particular iteration. High costs are assigned randomly within a specified range to variables

whose constraints are not being satisfied, where low costs are assigned to those whose constraints are met. In this way, a solution is a feasible schedule that minimizes the cumulative cost.

Gaspero's neighborhood is then created for each course. Two solutions are neighbors if they differ for the period assigned to a single course. From here, "moves" can be denoted by *(course number, old period, new period).* Gaspero defines the inverse operation of the move (x,h_1,h_2) to be anything of the form $(x,\_,\_)$. In that way, once an exam has been moved, it cannot be moved again while it is in the tabu list.

His search technique begins with an initial feasible solution created from a greedy algorithm. Then, lists are created that contain exams whose current times violate either hard or soft constraints. After each move, the lists are updated. Any exam not on one of these lists is never analyzed. His algorithm was able to schedule exams under various conditions, the most difficult of which being 138 exams in only 13 time slots across a large number of rooms. In most cases, the solutions were found within 20,000 iterations and 5 minutes CPU time.

Costa (1994) utilized TS search routines in the scheduling of high school courses. His approach differs from Gaspero's in several places. First, the creation of an initial solution is much more systematic. Courses scheduled first are the ones having the smallest number of available periods. In other words, Costa began by scheduling the classes that are hardest to schedule. A hierarchical set of "overlap degree values was created so that classes that carry higher overlap restrictions are scheduled first.

Costa defined the neighborhood $N(T)$ of a feasible time table T as another feasible time table T' where T and T' differ from each other by one meeting of one class being assigned to a

different period. Two tabu lists, T_1 and T_2 are generated every time a lecture is moved. If lecture x is moved from period p_1 to p_2, x is added to T_1, and the ordered pair (x,p_1) is added to T_2. Then, for $|T_1|$ iterations of the TS, x is not allowed to be moved into p_1 again. The result of his research is the successful schedule of 375 courses in a Swiss high school. The time table produced was "as good, if not better" than a time table that was produced by hand. His algorithm required 41 minutes of CPU time and 20521 iterations. Equally impressive is that one experiment yielded an overlap-free time table produced in only 969 iterations.

2.3.2-2B Genetic Algorithms

Genetic Algorithms(GAs) have been found very useful in the construction of time tables. At its essence, a genetic algorithm simulates the evolution of a population. However, instead of requiring thousands of years, these algorithms require thousands of iterations on a computer. Mirroring the theory of natural selection, solutions generated at later stages of the algorithm are evolved and generally better than their predecessors. Genetic algorithms are interesting in that they permit the manipulation of feasible solutions with less emphasis placed on the search of an entire neighborhood. Applications of GAs can be found in almost all fields that focus on optimization [Busacca 2001, Mares 1996, Linkens 1995, Haidar 1998, Jagielskaa 1999] . Several researchers have been able to implement similar methodology to machine learning (Goldberg 1988), and the Job Shop Problem (Davis 1985). GAs have also been applied to electromagnetic optimization (Rahmat-Samii 1999) and surface wave inversion [Yamanaka 1996]. In the financial sector, GAs have been applied to bankruptcy prediction modeling (Shin 2002).

GAs use special genetic operators such as *cross over* and *mutation* that create the next generation of candidate solutions from a parent set where each solution usually consists of a sequence of 1's and 0's. These operators mimic patterns seen in nature where the breeding of a

population produces children who carry similar traits to their parents. Furthermore, children inherit both beneficial and harmful combinations of their parents' traits. GAs utilize an operator dubbed cross over to carry out this phenomena on a set of solutions. In nature, some children develop traits distinct from any of their parents. This phenomenon is called "mutation" and occurs at a seemingly random frequency.

Cross over selects two parent solutions from a pool and creates two children, each having characteristics from both parents. Determining which characteristics are shared is algorithm-specific, but in general a "cross-over point" is selected and divisions of parent solutions are made based upon it. The following diagram serves as a good illustration:

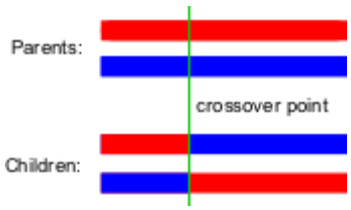

Figure 2.1 Cross Over (source: Wikipedia, accessed: 6-24-2011)

As seen above, the parents' code was divided into two parts: A and B. Child 1 receives part A from parent 1 and part B from parent 2. Child 2 receives part A from parent 2 and Part B from parent 1. In this way, both children share traits of their parents but are still independent from one another.

Using encoded bit-strings, an example of cross over can be given. Let parent p_1=(1,0,1,1,0,0) and parent p_2=(0,0,1,1,1,1). Let the cross over point be the fourth bit in each string. In other words, each parent is divided into two groups: bits 1-4, and bits 5-6. The children of these two parents are c_1=(1,0,1,1,1,1) and c_2=(0,0,1,1,0,0) .

Mutation operators are employed to create diversity in solutions from one generation to another. Single-point mutation assigns a random variable to each bit of a parent solution. Then,

based upon pre-defined probabilities, a bit is changed if a randomly-generated value of its corresponding variable meets certain criteria. Often times, the criteria are changed iteratively. Both cross over and mutation can create solutions *worse* than their parents. It is for this reason, that an objective function is necessary.

GAs use objective functions that measure the "fitness" (i.e. quality) of solutions. Again, the term "fitness" comes from "survival of the fittest" and describes the adaptability of a solution. These *fitness functions* are always problem-specific, but when applied to scheduling usually revolve around the number of conflicts generated by a certain schedule. Many children are created at each iteration and only the fittest survive to breed in the next generation.

Colorni applied a GA to the *Timetabling Problem* in 1990. The goal of his research was to schedule a section of classes in a high school environment. Given 10 classes, 20 teachers and a set of weekly teaching hours, solutions were constructed as 5-tuples appearing as (*H,T,A,R,f*) where:

1. $H = \{h_1, h_2, \dots, h_n\}$ is a finite set of hours in a school-day.
2. $T = \{t_1, t_2, \dots, t_m\}$ is a finite set of teachers.
3. *A* is a set of lessons to be taught.
4. *R* is an *m x n* matrix where $r_{ij} \in R$ signifies teacher *i* teaching lesson *j*.
5. *f* is the value of the tuple's fitness which is to be maximized.

In this case, *f* depends upon four variables: $p, q, u,$ and s. The variable p refers to the number of times a particular solution involves more than one teacher in the same class. This is minimized through the implementation of a separate parameter. The variable q represents a set of didactic goals and preferences. For instance, should Algebra meet once per day for 5 days, or 5 times in one day? Then variable u is a set of organizational goals. An example of this would

be always keeping a pool of teachers available in case substitutions need to be made. Lastly, s is a set of personal goals. These are generally not as important as the other goals but should be satisfied if at all possible. An example of this might be giving certain teachers a "light load" on particular days.

Colorni's algorithm only considers feasible solutions. That is, every class is matched with exactly one teacher in one room at one time. Interestingly, the teachers; movements are largely controlled by a variation of the *Traveling Salesman Problem.* Each $r_{ij} \in R$ represents a *gene* or trait that can be altered. These genes undergo cross over, mutation, and filtering operators throughout several iterations. Each iteration creates child solutions that are feasible and distinct from one another. At the conclusion of each iteration, the best group of solutions is chosen to be bred during the next iteration. The algorithm is terminated after a user-specified number of iterations has been completed. It was found that the algorithm always converges to a positive cost because of the way in which teachers' needs conflict with one another.

2.3.2-2C Simulated Annealing

Simulated Annealing is another popular metaheuristic. The term "annealing" refers to the effect of cooling atoms. When atoms are at high temperatures, their movements are erratic and seemingly random. While they cool, they begin slowing down and moving more purposefully until they "stop". Annealing has been used since the middle ages when blacksmiths cooled newly-molded swords by setting them in burning embers. As the embers died out, the sword cools in a way that makes it much stronger than if it had been cooled rapidly. This idea has been applied to optimization problems through the creation of simulated annealing algorithms.

One of the most exciting things about simulated annealing is its versatility across multiple fields of study and sectors of industry in the form of a general optimization technique. The methodology of annealing has also been applied to the *Traveling Salesman Problem* (Allwright 1989). Computer chip designers have utilized annealing in the development of the Versatile Placement and Routing algorithms (Betz 1997) which find near-optimal placement of components to save cost. Further application can be found in manufacturing through the scheduling of assembly machines to maximize throughput of product (Ben-Arieh 1992) and factory layout design (Souilah 1995).

Simulated annealing algorithms are based upon a neighborhood search similar to that of a tabu search. Initially, a solution *x* is randomly generated. Then the neighborhood *N(x)* is searched and a neighbor *x'* is selected. The quality of *x'* is measured by an objective function. The solution x' can be kept as the current candidate solution or thrown out. This decision is based upon the quality of *x'* and the present value of a probability p, which represents cooling of atoms.

The probability *p* is initially assigned a high value; usually between .95 and .99. It represents the likelihood that a move in a problem is accepted if it worsens the value of the objective function. After each iteration of the algorithm, *p* is lessened by a factor or "cooling rate" *r* so that the probability of accepting a bad move becomes less and less likely as the algorithm progresses. This cooling rate insures that the probability of making several bad moves in a row during the later stages of the algorithm is extremely rare but still occurs. Thus, simulated annealing algorithms avoid becoming stuck at local optima.

In regards to the Timetabling Problem, simulated annealing has been utilized in the production of sequential and parallel algorithms to manipulate time tables into feasible schedules (Abramson, 1991). Initially, tasks and resources are assigned randomly and in an impractical manner. The time slots in which resources are assigned to tasks are then moved and the change in a cost function is recorded. Abramson applied Simulated Annealing to TTP by first establishing a feasible solution in which tuples called *elements* consisting of a body of students, a room, a teacher and a particular lecture are randomly assigned throughout a time table. A cost function is established based upon the number of "class clashes" that are found. A class clash is defined as the scheduling of one of the *elements* in a way that forces teachers or students to be in 2 different places at once. Another type of clash would be *not* scheduling an element that needs scheduled in order to meet the requirements of that class. Note here that students do not play a role in the cost function.

Once the initial solution is constructed, a sequential simulated annealing algorithm randomly chooses a time-window or *period* P_i. An element E_k is then randomly chosen that is currently scheduled within that period and moved to another period P_j, also chosen at random. This move has a corresponding impact upon the cost function so the change in cost must be calculated using two components: the *from* cost and the *to* cost. The *from* cost refers to the bettering or worsening of the time table through the removal of E_k from P_i. The *to* cost then, refers to the change in cost through the insertion of E_t into P_j. This move is then accepted or rejected based upon the change in the cost function and the value of a random *unif(0,1)* number.

Costs are further broken down into class costs, teacher costs, and room costs. Each refers to the over / under assignment of each within a period. For instance, if the removal of an element E_k from P_i results in the number of occurrences its teacher T_k in P_i decreases however

remains greater than 0, the cost function is lessened by 1. Similarly, room costs and class costs are calculated based on the number of their occurrences within a period. This algorithm differs from classical simulated annealing in that elements are shifting positions and not being swapped with other elements. Because of this, each period can have a different number of elements assigned.

Abramson tested this method with varying numbers of rooms, teachers, and classes. when only 11 elements require scheduling, his algorithm finds a 0 cost solution in only 11 seconds. Furthermore, scheduling 600 elements with ample teacher and room options required only about 5 minutes. The most difficult schedule attempted contained 37 teachers, 24 rooms, 101 classes all totaling 757 elements. A 0 cost solution was found after 14 hours. From his results, it seems the greatest indicator of ease-to-solve is the number of teachers available for teaching as one experiment lessened the number of teachers from 30 to 20 and saw a change in the time to solve jump from 296 seconds to 5548 seconds. A parallel algorithm was also created in which multiple elements can be moved at the same time. This speeds up completion of the algorithm immensely as more processors are added.

The weaknesses of Abramson's technique can be found in the rigidness of its inputs. The user must input a room for a particular element which cannot be changed, even if an identical room is free nearby. Also, this technique requires a static, uniform time-constraint be placed on each element. In other words, every lecture of every class must take exactly 1 period to complete. Thirdly, this technique does not permit multiple classes to be scheduled in the same room in the case that teachers would like to combine classes. Lastly, this technique does not view one type of clash as more important than another type of clash. Basically, a teacher being over extended is equally important as a class being scheduled too few lectures. The lack of a

hierarchy of class clashes is important because most scheduling heuristics separate constraints into *hard (vital)* or *soft(desirable)* categories.

2.4 Summary

This section has discussed several popular and current approaches to scheduling theory and the TTP specifically. Each approach seeks to assign tasks to a set of resources in a way that minimizes cost while insuring all tasks are completed. Exact methods such as linear/integer programming have been effective in scheduling small sets of tasks but require impractical computation time for real world problems. For large instances, metaheuristics such as tabu search, genetic algorithms and simulated annealing have had greater success in constructing quality time tables in less time.

The next section presents a uniquely flexible system for scheduling classes in a large high school setting. This system permits teachers to design the structures of their own classes with regards to frequency, duration and location of their lessons. The freedom to design each class individually is highly valuable as it gives educators another tool in their effort to maximize student learning. Of course, such a progressive system produces unique challenges. It is for this reason that the next section describes a set of algorithms to schedule classes in an effective manner.

Chapter 3

The SMART Scheduler

This chapter presents this thesis' primary advancement, the SMART Scheduler. The SMART Scheduler is a computer program capable of scheduling large numbers of classes within the modular environment currently being used at Westside High School. This chapter describes modular scheduling both qualitatively and mathematically. This is followed by an explanation of the routines developed for the SMART Scheduler.

The chapter is broken into three sections. The first section gives a qualitative description of a modular class scheduling system. The second section contains a mathematical model of the system. The third section describes the steps taken to construct a computer program to schedule classes within this system for Westside High School. The final section gives some computational results.

As previously mentioned, Westside High School of Omaha, Nebraska utilizes a unique system for scheduling their classes. This system permits each teacher to construct the weekly structure of their classes. Each teacher controls the frequency and duration of each lesson within an allocated number of minutes per week. The freedom to design the structure of a class is almost unheard of in the education community. WHS's system allows their average graduate to attain a full semester's education greater than graduates of surrounding high schools (Hutchins

2009). The advantages of the system can also be seen in the amount of one-on-one time students experience with their teachers outside of the classroom during mutually open periods.

Currently, WHS employs one of their former mathematics teachers to schedule their classes. A semester's schedule requires more than 7 weeks of full-time work to construct using a program developed in FORTRAN®. Clearly, this is a heavy burden to the scheduler but also a huge cost to the school. As technology progresses, it has become less and less practical to schedule classes in this way.

This thesis develops a mathematical model of WHS's unique *modular scheduling* system and constructs a heuristic that schedules their classes in significantly less time and cost. A scheduling program written in c and an interactive website utilizing HTML and PHP languages as well as SQL-queries to a database are contributions of this research. First, a detailed explanation of modular scheduling is necessary.

3.1 Qualitative Description of Modular Scheduling

Each day of the week is broken into 14 time windows called *mods.* These are similar to the fixed "periods" of other high schools. A unique feature of this scheduling system, however, is that mods do not all have the same length. Mods 1, 2, 3, 12, 13 and 14 are each 40 minutes while mods 4, 5, 6, 7, 8, 9, 10, 11 are 20 minutes. This is a necessity when considering that lessons at Westside do not all have the same duration. Furthermore, some students use 20, 40 or 60 minutes each day for lunch because their schedules are able to reflect this individual need.

Initially, every class offered is allocated a certain number of minutes that teachers can spend in shaping their class. This number usually corresponds to the number of credit hours that

the class is worth. Then, decisions regarding several aspects of the weekly structure are made. Each teacher's section is independent of other sections. A section of a class first breaks the week into *phases*. Each phase contains a set of activities in which all students must participate before they have completed that phase. The teacher is allowed up to seven phases. The structure of each phase is independent of other phases with a unique set of control parameters. The control parameters are as follows:

- **Number of *groups*** – The pool of students belonging to a teacher are divided into a number of groups. For instance, suppose an Algebra teacher has 120 students. A phase containing 4 groups would ideally split the students into 4 groups with 30 students in each group.
- **Number of *meetings*** – Each phase contains a specified number of lessons that each student must attend before the phase is complete.
- ***Duration*** – The duration of each meeting is assigned a length of time. This can be 20, 40, 60 or 80 minutes.
- ***Class Type*** – The class type of each phase is a *large group*, a *small group* or a *lab*.

  Large groups are instances where all of a teacher's (or group of teachers') students meet at the same time. These meetings require special rooms (i.e. auditoriums) to handle the influx of students.

  Small groups denote meetings where a teacher's students are broken into multiple groups. Each meeting of a small group takes place in a non-specialized classroom where the student population does not exceed 30.

  Labs are meetings that require a classroom with specialized equipment.

Instances of lab meetings are visits to computer labs, science labs, and industrial technology rooms.

- ***Specified* and *Alternate Rooms*** – If multiple rooms of a certain type are available, a teacher might request a specific room for each of the meetings in a phase. The structure also allows for alternate rooms to be assigned in case the specified room is not available.
- ***Tie Type*** – Each phase is assigned a tie type which governs the rules of student grouping and re-grouping within and between phases. There are three different tie types: Section-tied, Teacher-tied and Neither-tied.

  Section-tied classes are the most restrictive in that they require the exact same group of students to meet together for each of a phase's meetings. Most high schools use this type of scheduling approach in which students become familiar with the same 20-30 students every day.

  Teacher-tied classes are less restrictive in that they permit students to attend *any* of their phase's groups for each meeting as long as it taught by the same teacher. This is a unique feature of Westside's approach to scheduling. Classes using this type of tie assign students to a meeting of a particular meeting without regard to the group they were in for a different meeting. In this way, a pool of students is divided differently for each meeting.

  Neither-tied classes are less restrictive still. A phase that is neither-tied permits students to see any group of *any teacher* of that class. These types of meetings are usually reserved for regular assessments where the teacher

administering an assessment is not as important as the student being able to attend.

- ***Day / Mod Constraints*** – A phase may be assigned day and mod constraints as a way of restricting when meetings of a phase may be scheduled during the week. For instance, if a phase contains 3 meetings, a teacher may restrict these meetings to be held no sooner than Wednesday Mod 6 and no later than Friday Mod 4. Furthermore, day constraints can be specified in a way that *wraps* the weekend. In other words, a teacher may request that five meetings of a phase meet between Thursday and the following Wednesday. This is important in sequencing classes considering that students may attend a particular lesson in different groups on different days.

Teachers may also specify less important parameters such as whether they want a particular phase to take place on a single day with no meetings of others phases taking place on that day. For instance, a teacher may request that a lab phase be the only thing scheduled on its day. A teacher may also request that a particular phase be the *sequencing point* where all other phases are scheduled afterward allowing wrapping of the weekend. Lastly, a teacher may allow each of their students to "*miss a mod*" from a week's lessons. This is utilized when a class is especially hard to fit into all of its students' schedules such as band, orchestra and theater classes that take students from all grades and skill levels. Students of these classes are allowed to miss one 40 minute meeting during the week without being counted absent.

Another unique feature of modular scheduling permits multiple classes to meet in the same room at the same time. This ability to *fuse* classes together allows the collaboration of multiple teachers to impact how lessons are delivered because they are often "team-taught". For example,

a set of Physics teachers may desire to set aside one meeting of each week in which all of their students meet in a large group at the same time. This large group would most likely meet in a large auditorium where lessons are delivered by one teacher or a group. An advantage of this type of meeting is that students experience an environment similar to that which they will see again in college.

Fusing is also used as a way of collaborating across departments. For example, two chemistry teachers may team-up and create a curriculum that intertwines both of their teaching styles. They are also able to supervise several classes at one time during hands-on activites.

3.2 A Mathematical Model for Modular Scheduling

This section focuses on the development of a mathematical model to describe modular scheduling. This model is developed using techniques that lend themselves to heuristics such as the ones described in chapter 2. An original approach to finding a solution to this problem is presented later in this chapter.

Recall for input, let $C = \{c_1, c_2, \dots, c_n\}$ be a set of classes, $T = \{t_1, t_2, \dots, t_m\}$ be a set of teachers and $R = \{r_1, r_2, \dots, r_p\}$ be a set of rooms. Let *D* be the set of days and *N* be the set of mods. Let τ_j and υ_j be the teacher's first and last mods in school for each day for each $t_j \in T$. Let $P_j^i = \{p_{j1}^i, p_{j2}^i, \dots, p_{jl}^i\}$ be a set of phases belonging to $c_i \in C$, $t_j \in T$. Each phase $p_{jk}^i \in P_j^i$ has several attributes to consider. Let α_{jk}^i denote the number of groups, β_{jk}^i denote the number of meetings and γ_{jk}^i denote the duration of each meeting. Let δ_{jk}^i denote the class type (Large Group, Small Group, Lab) and ε_{jk}^i denote the tie type (Section-tied, Teacher-Tied, and Neither-Tied. To govern when a meeting is scheduled, let ζ_{jk}^i denote the first day and η_{jk}^i denote the first

mod of day ζ_{jk}^{i} on which the first lesson of phase *k* may be taught. Also, let θ_{jk}^{i} denote the last day and ϑ_{jk}^{i} denote the last mod on day θ_{jk}^{i} in which a lesson may be taught.

Furthermore, let $G_{jk}^{i} = \{g_{jk1}^{i}, g_{jk2}^{i}, \ldots g_{jk\alpha_{jk}^{i}}^{i}\}$ be the set of groups in the k^{th} phase of teacher $t_j's$ section of class $c_i \in C$. Let $M_{ijkl} = \{m_{ijkl1}, m_{ijkl2}, \ldots m_{ijkl\beta_{jk}^{i}}\}$ be the set of meetings belonging to the l^{th} group within the k^{th} phase of teacher $t_j's$ section of class $c_i \in C$. Let $\lambda_{jk}^{i} \in R$ denote the desired room of the school in which the lesson is to be scheduled.

To form an integer programming model, it is assumed that each class must be scheduled in the desired room. Let x_{ijklq}^{de} take on a value of 1 if meeting m_{ijklq} is occurring during day *d* during mod *e* and a 0 otherwise. Let b_{ijklq}^{de} take on a value of 1 if meeting m_{ijklq} is begins on day *d* during mod *e* and a 0 otherwise.

The problem can then be modeled as:

$$\max \sum_{d:d\in D} \sum_{e:\, e\in N} \sum_{m_{ijklq}\in M_{ijkl}} b_{ijklq}^{de}$$

s.t.

$$\sum_{e\in N}\sum_{d\in D} b_{ijklq}^{de} \leq 1 \;\; \forall c_i \in C, t_j \in T, p_{jk}^{i} \in P_j^{i}, g_{jkl}^{i} \in G_{jk}^{i}, m_{ijklq} \in M_{ijkl} \qquad (1)$$

$$b_{ijklq}^{d\,e} = x_{ijklq}^{d\,e+w} \forall c_i \in C, t_j \in T, p_{jk}^{i} \in P_j^{i}, g_{jkl}^{i} \in G_{jk}^{i}, m_{ijkl2} \in M_{ijkl}, d \in D, w \in \{0, \ldots, \gamma_{jk}^{i}\} \qquad (2)$$

$$x_{ijklq}^{de} = 0 \; \forall c_i \in C, t_j \in T, p_{jk}^{i} \in P_j^{i}, g_{jkl}^{i} \in G_{jk}^{i}, m_{ijkl2} \in M_{ijkl}, d \leq \eta_{jk}^{i},$$

and if $d = \eta_{jk}^{i}$ then e $\leq \zeta_{jk}^{i}$ (3)

$$x_{ijklq}^{de} = 0 \; \forall c_i \in C, t_j \in T, p_{jk}^i \in P_j^i, g_{jkl}^i \in G_{jk}^i, m_{ijkl2} \in M_{ijkl}, d > \theta_{jk}^i,$$

and if $d = \theta_{jk}^i$ then $e \leq \zeta_{jk}^i$ (4)

and if $d = \theta_{jk}^i$ then $e \leq \zeta_{jk}^i$ (4)

$$x_{ijklq}^{de} = 0 \; \forall \; \mathrm{e} \leq \tau_{\mathrm{j}}, c_i \in C, t_j \in T, p_{jk}^i \in P_j^i, g_{jkl}^i \in G_{jk}^i, m_{ijklq} \in M_{ijkl}, d \epsilon D \quad (5)$$

$$x_{ijklq}^{de} = 0 \; \forall \; \mathrm{e} \geq \upsilon_{\mathrm{j}}, c_i \in C, t_j \in T, p_{jk}^i \in P_j^i, g_{jkl}^i \in G_{jk}^i, m_{ijklq} \in M_{ijkl}, d \epsilon D \quad (6)$$

$$\sum_{m_{ijklq} \in M_{ijkl}} x_{jklq}^{ed} \leq 1 \; \forall e \in N, d \in D, t_j \in T \quad (7)$$

$$x_{ijklq}^{ed} + x_{i'j'k'l'q'}^{ed} \leq 1 \; \forall e \in N, d \in D, t_j t_{j'} \in T$$

such that $m_{ijklq} \in M_{ijkl}$ and $m_{i'j'k'l'q'} \in M_{i'j'k'l'}$ utilize the same room. (8)

$$x_{ijklq}^{ed} \epsilon \{0,1\} \; \forall e \in N, d \in D, c_i \in C, t_j \in T, p_{jk}^i \in P_j^i, g_{jkl}^i \in G_{jk}^i, m_{ijklq} \in M_{ijkl} \quad (9)$$

$$b_{ijklq}^{ed} \epsilon \{0,1\} \; \forall e \in N, d \in D, c_i \in C, t_j \in T, p_{jk}^i \in P_j^i, g_{jkl}^i \in G_{jk}^i, m_{ijklq} \in M_{ijkl} \quad (10)$$

The first set of constraints insures that every meeting has a beginning time and thus is scheduled. The second set of constraints forces a meeting to be held for the entire duration once a beginning time has been established. The third and fourth constraints insure that all meetings will be held within their corresponding phase's day and mod constraints. The fifth and sixth constraints restrict meetings to be held during times within the teacher's daily availability. The seventh constraint insures that a teacher cannot be involved in more than one meeting at a time. The eighth constraint permits rooms to host no more than one meeting at a time. The ninth and tenth constraints restrict the variables that govern when a meeting is taking place to binary values. Finally, the objective function seeks to maximize the total number of meetings scheduled.

While the model presented above covers the majority of parameters that are considered in creating a modular schedule, there are several aspects that were not covered. Things such as sequencing classes across weekends, fusing classes together and scheduling labs in adequately equipped rooms were not mentioned. This is partially a result of the complexity of describing such parameters mathematically but also due to the fact that this model is not what was used to solve the problem. Given that there are 70 mods per week and over 3,000 meetings to schedule, the IP described above would require over 420,000 variables making a solution computationally intractable. The size of this problem is well beyond the technological limitations of the average secondary school. Therefore, another system for scheduling classes within the modular environment is necessary.

3.3 A Computer Program to Schedule Classes within a Modularized Environment

It is difficult to find a cohesive class schedule that satisfies all teachers' structure requests as they often conflict with each other. For this reason, a candidate class schedule must be constructed and meet faculty approval before students can be scheduled. This section outlines an approach to scheduling modularized classes with regards to teacher and room usage. The general philosophy for the scheduler is to start with the most highly constrained (i.e. most difficult to schedule) classes or sets of classes and work toward the easiest. Not all of the 8,000 lines of code that lie within the SMART Scheduler are explained but all important routines will be covered in depth.

When scheduling class meetings, the goal is to find a teacher, room and a time that is compatible with other meetings taking place. That is, no un-fused classes will meet in the same room at the same time nor require a teacher to be in two distinct places at once. Furthermore, the parameters of modular scheduling require unique inputs in order to prioritize classes to be scheduled. Once inputs are described, the routines that the SMART Scheduler utilizes are discussed.

3.3.1 Inputs

The first set of routines reads-in all of the data necessary to schedule classes in the form of Excel files. The first file to be addressed catalogs all of the school's teachers. Each teacher's Identification number, name, and department is stored to the computer's temporary memory. Because some teachers are part-time and others work at multiple schools, it is important that each teacher's unique availability is stored as well. This availability is stored as preferred starting and ending mods.

The second data gathering routine stores information about the rooms in the school. Each room has a capacity, a primary department, a secondary department, and a room "type". The room types can be one of: small group, large group, or lab.

Data concerning the student population and their class requests is stored next. This information consists of each student's identification number, name, projected year of graduation and class requests stored as class I.D. numbers.

Lastly, data concerning classes is stored. General information such as a class's name, department and desired students-per-meeting is stored first. Then, the SMART Scheduler begins

storing class structures on a phase by phase basis. This data contains the important phase attributes described in section 3.1 as well as requested class fuses.

3.3.2 Improved Prioritizing of Classes

In order to make a balanced schedule, the number of students to expect per teacher and per group in a teacher's class is calculated. Students cannot be randomly assigned a teacher as it would affect the balancing of classes. In other words, it is unfair to assign 50 students to one teacher and 150 to another. The SMART Scheduler is able to remedy this. Each teacher is assigned a number of sections for each of their classes. The number of sections is used to determine the correct proportion of students that should be assigned to that teacher. Mathematically this can be calculated by letting DS_{ij} represent the desired number of students assigned to teacher $t_j \in T$ in class $c_i \in C$. Also, let $\pi_{\mathrm{j}}^{\mathrm{i}}$ be the number sections of class $c_i \in C$ assigned to teacher $t_j \in T$. Lastly, remember that S^i represents the set of students requesting class *i*.

$$DS_{ij} = \frac{\pi_{\mathrm{j}}^{\mathrm{i}} * \sum_{i:c_i \in C} S^i}{\sum_{j:t_j \in T} \pi_{\mathrm{j}}^{\mathrm{i}}} \; \forall c_i \in C, t_j \in T$$

For example, suppose teacher t_1 has 3 sections, t_2 has 2 sections and t_3 has 1 section of Algebra. Thus, 6 sections of Algebra are being offered total. Also, suppose that 240 students have requested Algebra for the following semester. The SMART Scheduler would assign $\frac{3*240}{6} = 120$ students to t_1, $\frac{2*240}{6} = 80$ to t_2, and $\frac{1*240}{6} = 40$ to t_3.

Now, it is possible to calculate the desired number of students per group within a particular phase of a teacher's class. Let DSG_{ijk} denote the desired number of students per group of the k^{th}

phase of class *i* belonging to teacher *j*. Also, remember that α_{jk}^{i} is the number of groups teacher *j* has divided their students into for phase *k* of class *i*. The number of groups is calculated by:

$$DSG_{ijk} = \frac{DS_{ij}}{\alpha_{jk}^{i}} \ \forall c_i \in C, t_j \in T, p_{jk}^{i} \in P_j^{i}$$

An instance of this could be illustrated expanding the example given above. Suppose 120 students are assigned to t_1. Also, suppose t_1 has designed phase 1 to divide their students into 4 groups. Each meeting would then contain $\frac{120}{4} = 30$ students.

Once the number of students per teacher and students per group are calculated, the SMART Scheduler determines the window size of each phase. A phase's window size refers to the number of 20-minute intervals that have been allocated to complete all meetings. Let μ_{jk}^{i} be the window size of the k^{th} phase of class *i* belonging to teacher *j*. Also, let λ_m be the number of 20 minute intervals contained in mod *m*. Note that there are 20 such intervals in a day. Remember from the model that ζ_{jk}^{i} is the first day and η_{jk}^{i} is the first mod on ζ_{jk}^{i} that a meeting from phase *k* may be scheduled. Also, θ_{jk}^{i} denotes the last day and ϑ_{jk}^{i} is the last mod on θ_{jk}^{i} that a meeting from phase *k* may be scheduled. Because the sequencing of classes can be done across the weekend, there are three cases to consider.

The first case addresses instances when the first day of phase *k* is less than the last. Here, the class is not wrapping a weekend. An example would be if the first scheduable day of a phase is Monday (day 0) and the last is Friday (day 4). If $\zeta_{jk}^{i} < \theta_{jk}^{i}$ a phase's window size is calculated by

$$\mu_{jk}^{i} = \sum_{\eta_{jk}^{i}}^{14} \lambda_m + 20\left(\theta_{jk}^{i} - \zeta_{jk}^{i} - 1\right) + \sum_{1}^{\vartheta_{jk}^{i}} \lambda_m .$$

The calculation is made by first adding the number of 20 minute intervals that are schedulable on the first day. The number of 20 minute intervals is then calculated for the number of days between ζ_{jk}^{i} and θ_{jk}^{i}. Lastly, the number of 20 minute intervals is added for the last day.

The second case addresses instances when all meetings of a phase are held on the same day. If $\zeta_{jk}^{i} = \theta_{jk}^{i}$ a phase's window size is calculated by $\mu_{jk}^{i} = \sum_{\eta_{jk}^{i}}^{\vartheta_{jk}^{i}} \lambda_m$. The number of 20 minute intervals is summed from the first schedulable mod to the last one on day ζ_{jk}^{i}.

The last possibility requires wrapping across a weekend. An example of this would be a phase with a first schedulable day of Thursday and a last day being the following Tuesday. Here, $\zeta_{jk}^{i} = 3$ (Thursday) and $\theta_{jk}^{i} = 1$ (Tuesday). If $\zeta_{jk}^{i} > \theta_{jk}^{i}$, a phase's window size is calculated by

$$\mu_{jk}^{i} = \sum_{\eta_{jk}^{i}}^{14} \lambda_m + 20\left(4 - \zeta_{jk}^{i}\right) + 20\left(\theta_{jk}^{i} - 1\right) + \sum_{1}^{\vartheta_{jk}^{i}} \lambda_m .$$

This calculation is made by first summing all schedulable mods on day ζ_{jk}^{i} starting with mod η_{jk}^{i}. Next, the number of 20 minute intervals from day $\zeta_{jk}^{i} + 1$ until Friday (day 4) is added. Then, the number of 20 minute intervals from Monday until day θ_{jk}^{i} is added. Finally, the number of schedulable mods on day θ_{jk}^{i} is added.

Next, let ξ_{jk}^{i} be the number of 20 minute intervals that is required to complete all meetings of the k^{th} phase of class *i* belonging to teacher *j*. This can be calculated by:

$$\xi_{jk}^{i} = \alpha_{jk}^{i} * \beta_{jk}^{i} * \gamma_{jk}^{i}$$

Finally, the ratio $\frac{\mu_{jk}^{i}}{\xi_{jk}^{i}}$ is calculated and stored as the phase's *constrained time window*. This ratio serves as the standard measure of how constrained a phase is. This provides guidelines so that the highest constrained classes are scheduled first.

3.3.3 Main Routines

Three routines are used widely throughout the scheduling of classes. The first, denoted *AvailableRoom( )* identifies the compatibility of assigning a class to a particular location at a specific time . The second, *AvailableTeacher( )* identifies the compatibility of assigning a teacher to a class at a specific time. The third, *BuildWindow( ),* calculates the window of time in which a meeting must take place so that all meetings of the class are spaced in a reasonable way.

When deciding where a class should meet, it is important to consider several attributes of both the class and the room. *AvailableRoom( )* initializes with an input class meeting and a potential day and starting mod for the meeting. First, the routine identifies whether or not the class could feasibly be scheduled beginning with the starting mod provided. If a meeting is 60 minutes for instance, it may not begin during mod 1 because both mods 1 and 2 are 40 minutes in length. Therefore, mod 1 would be rejected as a candidate starting mod. However, mod 3 would be a compatible starting time for a 60 minute meeting as mod 3 is 40 minutes and mod 4 is 20 minutes long.

Secondly, the capacity of the room and the projected students-per-meeting must be compared. The starting mod is rejected if students are not able to fit into the room. The status of the room during a particular time is investigated next. If scheduling the given meeting at the

potential time would force two classes to meet in the same place at the same time, the starting mod is rejected.

Once a room has been identified as a potential location for a class, the availability of the class's corresponding teacher is inspected. *AvailableTeacher( )* is initialized with a given teacher's I.D. number, a class meeting and a candidate starting mod. If the teacher is not working at the school during one of the mods that would be affected by scheduling the class during the candidate starting mod, the starting mod is rejected.

Next, *AvailableTeacher( )* identifies whether a teacher is busy with another activity during the mods that are taken up by scheduling the meeting for the starting mod. The starting mod is rejected if the teacher is busy. It is important for every teacher to be given a 40 minutes window of time to serve as lunch time. Since the cafeteria is only open during certain times, *AvailableTeacher( )* ensures every teacher has a lunch break.

Lastly, the routine *BuildWindow()* establishes a desirable time frame for every meeting by considering the day/mod constraints, number of groups and additional meetings of a phase. The goal is to assign a "preferred" start day and mod as well as a preferred end day and mod in which the meeting must be scheduled to keep meetings balanced across the time-window of the phase. *BuildWindow()* initializes with a single meeting *i* of a class belonging to a certain teacher. First, a search is made for the nearest meeting scheduled before *i* and the nearest meeting scheduled after *i* to act as the boundaries of the current meeting. If no such meetings have been scheduled, The SMART Scheduler uses the phase's day/mod constraints as boundaries for that end of the time-window. This is called the *meetings gap* and represents the number of days in which some meetings must be scheduled. Once the meeting gap is found, there are three possible scenarios:

the number of days in the meeting gap is less than, equal to, or greater than the number of meetings needing to be scheduled within it.

If the meeting gap is less than the number of meetings remaining to be scheduled, each meeting will occur on a different day and at least one day won't have a meeting. For example, if the meeting gap is 3 and only 2 meetings remain to be scheduled during that time, one day would not contain a meeting.

If meeting gap is equal to the number of meetings that must take place, each meeting will take place on a different day and every day will have a meeting. This is the standard case when a class is broken into 5 meetings, each taking place on a different day.

Lastly, if the meeting gap is greater than the number of meetings, at least two meetings will have to be held on the same day. The start mod of meeting *i* depends on when previous meetings have been scheduled. If a meeting was scheduled in the morning, meeting *i's* time window will begin in the afternoon of the same day. However, if the closest previous meeting was scheduled in the afternoon, meeting *i*'s time window will begin mod 1 of the next day and be limited to the first 7 mods.

3.3.4 A Sequential Approach to Scheduling Classes

Once relevant data has been saved and preliminary calculations have been made, The SMART Scheduler is ready to begin scheduling classes. Since teachers are permitted to request specific windows of time for each of the phases of their class, these windows can be as small as a single 20 minute mod. Such a restricted window can present difficulties if scheduling has commenced and another class has been scheduled in that spot. For that reason, the SMART

Scheduler begins by scheduling the most constrained meetings first. The level of restriction is measured by the phase's constrained time window, which was calculated in the program's initialization phase.

The SMART Scheduler begins by scheduling phases with a constrained time window equal to 1. This is equivalent to starting with phases that have been allocated exactly the number of mods required to fit all meetings of the phase. The program applies the *SchedulePhase( )* function in a way that the first slot mutually available between the teacher and the desired room of a class is chosen as the starting mod and the class is scheduled. If a phase has multiple groups or meetings but has been given just enough time to schedule each, the result is usually a set of consecutive mods in which the teacher and room is busy. Teachers at Westside have requested that they be scheduled no more than 4 mods consecutively in order to give teachers time to prepare. However, the program ignores this requirement if several meetings must take place in a small amount of time. Thus, this is a soft constraint.

After all phases with a constrained time window equal to 1 have been scheduled, the SMART Scheduler iteratively moves to the unscheduled phases with constrained time window less than or equal to 1.1, 1.2, and so on up until 1.3. When classes were scheduled that had constrained time windows more than 1.3, it was observed that classes became unnecessarily clustered and teachers overused. An example would be 4, 40 minute meetings that must take place between and including mods 2 and 9 of the same day. These meetings will require 8, 20 minute intervals to complete and have been allocated 10. Thus the constrained time window of this phase would be $\frac{10}{8} = 1.125$ and would be scheduled along with other phases having constrained time windows between 1.2 and 1.3.

Next, the SMART Scheduler schedules classes that contain phases that require all meetings of a phase to complete on the same day. This is another instance of highly constrained classes needing to be addressed in the early stages of scheduling. Usually phases of this type require the same room and teacher for large sections of the day. The program searches for phases marked with the "all in one day" attribute and schedules them next.

The next section of the SMART Scheduler addresses a unique characteristic of the system. This is the ability to *sequence* classes across weekends. For example, a teacher may wish to have a large group on Wednesday of each week and then meet with their students 3 more times between Wednesdays. Let phase 1 govern the large group and phase 2 control the set of small groups. The program considers this instance by scheduling the Wednesday large group first and then adjusting the day constraints of the other phases accordingly. To address wrapping across the weekend, the SMART Scheduler lets days 0-4 correspond to Monday-Friday of the first week while days 5-9 correspond to Monday-Friday of the following week. The Wednesday large group in the example would be scheduled on day 2 and adjust the day constraints of phase 2 to between days 3 and 6. When phase 2 is eventually scheduled, *SchedulePhase( )* subtracts 5 from any starting and ending day of the phase's time window greater than 5. In other words, the adjusted day constraints are addressed modulo 5. Phase 2 of our example would have day constraints forcing the 3 meetings to occur between day 3 (Thursday) and day 1 (Tuesday).

In strategizing the scheduling of classes, it was agreed upon that the location and time of as many meetings as possible should be meaningful to the cohesiveness of the schedule. Randomly assigning meetings can be useful in creating a rudimentary class schedule but fails to satisfy students' class requests.

Let R be an *n x n* matrix where $r_{ij} \in R$ denotes the number of students who requested class *i* who also requested class *j*. The matrix *R* is utilized to construct *swim lanes* so that classes that are often requested together are scheduled so as not to overlap. Briefly, this is done by selecting a class *i*. Then searching row *i* of matrix R for the top 4 values. Each column *j* where one of these values corresponds to a class *j* with a large number of students requesting both class *i* and *j*. The program then schedules all meetings of one group from each phase for all 5 classes sequentially. Each time a class is scheduled, the program stores the days and mods that were used and blocks the other classes from scheduling during them. In this way, a group from every phase of every class is scheduled away from a group of each of its most mutually requested classes. This gives students a much higher chance of finding a group from each phase of each of their requested classes that works in their schedule.

Once classes with high numbers of mutual requests have been scheduled, the SMART Scheduler begins scheduling classes according to a restricted greedy algorithm. Unscheduled meetings of each class are scheduled by constructing arrays of all mutually free mods among the desired room and teacher. The program then selects one of these available times at random and schedules the class. The restriction occurs when the SMART Scheduler attempts to schedule a teacher for their 6^{th} 20-minute increment of the day. Scheduling meetings of a class equally across the full time window allotted helps balance the class load put onto a teacher. By forcing the program to search for open slots across the full time window and not simply choosing the first available, we help ensure the full time window is used well.

The next several routines help schedule any meetings that haven't been addressed and clean up any haphazard assignments. A secondary greedy algorithm is implemented next that does not

consider the load of a teacher on a given day. If a slot is found within a phase's time window in which the meeting can be scheduled in the desired room, it is scheduled.

At this point in the program, teacher and room schedules become dense and available mods are hard to find. A routine was created to help open new slots for classes to be scheduled. This is done by first picking an unscheduled meeting and a slot that fits into the teacher's schedule. The desired room is inspected during this time and if another class is using it, the SMART Scheduler tries to find another time in which the conflicting class could meet. If one exists, the conflicting class is moved and the slot becomes available for the unscheduled class.

Some rooms are requested so frequently that it is impossible to schedule all classes in them. At this point, the program begins searching for other rooms within a class's department that satisfy the class's needs and capacity requirements. If a room is found open during an open mod of the teacher, the meeting scheduled.

As is the case in several schools, rooms are shared by departments. With schools becoming more and more crowded, buildings are becoming so small that teachers from different departments often share rooms. The SMART Scheduler's final attempt to schedule meetings involves searching all rooms that have been approved for multi-departmental use for open mods that fit into a teacher's schedule. If any such mutually open slots are found, class meetings are scheduled accordingly.

The final routine of the class scheduler acts as an error corrector. It is desirable that meetings are taught in a sequential order. That is, all meeting 1s are completed before any meeting 2s are taught. The SMART Scheduler utilizes a sequencer that exchanges searches for and exchanges meetings that are out of order. Once a meeting $i+1$ of a phase is found to be scheduled before an

instance of a meeting *i*, the SMART Scheduler exchanges the scheduled times of meeting *i+1* with the last instance of meeting *i*. This way all groups of meeting *i* are completed before a teacher encounters meeting *i+1*.

If meetings of classes are left unscheduled after all routines have terminated, the program was unable to find a feasible time or location for them. This could be the result of the overuse of a room or teacher. It might also be caused from infeasible parameters such as nonsensical day/mod constraints or phases with more than 5 meetings each requiring its own day. To schedule these meetings, the user will need to identify the cause of each unscheduled meeting by investigating the class' structure and the output schedules of the assigned rooms and teachers. If mistakes are found, the user needs only to correct them and rerun the program.

3.4 Computational Results

Westside High School was gracious enough to provide scheduling data from the most recent semester, Spring 2011. The data governs parameters for 203 classes, 196 teachers, and 108 rooms. The classes were broken into 653 phases containing 1,366 groups and 3,036 meetings to schedule. In addition, there are 15,282 class requests from 2007 students that are addressed in designing swim lanes. The SMART Scheduler was able to schedule 98.12% of the meetings in 5.1 seconds.

The run time of the SMART Scheduler is very satisfactory given the enormous amount of data that is being input and the complexity of parameters. Recall in section 2.3.2-2C that Abramson's program was able to schedule 770 meetings in 14 hours. Granted, Abramson was using available technology in 1991. However, recall the Abramson's method was simulated annealing which requires a neighborhood search of multiple solutions. The SMART Scheduler,

on the other hand, does not rely upon a neighborhood search but instead attempts to make a quality schedule the first time.

Consider as well that previous class schedules have been implemented in class scheduling systems that require a class to meet at same time each day. Thus, when one meeting of a class is scheduled, all are scheduled for the same time every day of the week. A school that offers 208 classes only needs to schedule 208 meetings to complete a schedule. Modular scheduling treats each meeting individually. Therefore, scheduling 208 classes that make up 3,036 meetings is the same as scheduling 3,036 classes; a much greater accomplishment.

Chapter 4

An Interactive Website

This chapter describes the extensive effort that went into the development of an interactive website to aid students, counselors and administrators access scheduling information. This website transforms the SMART Scheduler from a research topic to a useful tool that can replace the current method that WHS uses to schedule their classes. Furthermore, it can be easily modified for other schools who desire to implement modular scheduling.

The website also provides capabilities for counselors to input their students' class requests. Furthermore, administrators are able to add, delete and modify the inputs of the scheduling program. The website is especially valuable because Westside has been deemed a "one-to-one" school under a federal grant. The grant provides laptop computers to all students and faculty for use in school or at home. This added internet accessibility enables students to check for changes in their schedules at any time. Additionally, parents/guardians are able to access their children's schedules from home.

After schedules are created by the program, they are uploaded to a mysql® database that is accessed by the website. The website has been partitioned into four password-protected areas: students, teachers, counselors and administrators. Each partition contains capabilities unique for the type of user.

4.1 Student Website Capabilities

When the student section of the website is accessed, the user is taken to a page containing the student's schedule, as shown in Figure 4.1. Having regular access to one's schedule is important as classes and assignment times change from day to day. Most schools permit teachers to request that students meet with students before/after school but a combination of Westside's modular scheduling and access to this website makes it easy for teachers and students to meet during the school day. Students are assigned to certain locations for multiple reasons. Most commonly, a student is assigned to a teacher's desk during a mutually open mod. At this time, the student catches up on missing work or engages in one-on-one instruction with the teacher. Counselors and Administrators may assign students to activities taking place outside of school. These activities range from supplemental instruction at the Westside Career Center to job training programs taking place throughout the city. Being able to check one's own schedule as it changes prepares students to deal with fluctuating schedules in the real world as they attend college or start careers.

Students are also able to view teachers' schedules. After entering the identification number of a teacher, they are taken to a page presenting that teacher's schedule. Students are not able to see the names of students assigned to that teacher as it would violate confidentiality agreements and cause potential problems. If the student does not know the teacher's I.D. number off hand, they can click the "find teacher information" button. This prompts a window to open in which the student can enter all or part of a teacher's last name. The page displays the names, ID numbers and email addresses of all teachers whose last name contains what the student had entered.

Schedule for Alonzo Carr					
	Monday	Tuesday	Wednesday	Thursday	Friday
Homeroom 8:00-8:15 (NO HR Wednesdays)	Home Room Stegman 385	Home Room Stegman 385	Home Room Stegman 385	Home Room Stegman 385	Home Room Stegman 385
Mod 1 8:20-9:00 (Wed. 8:00-8:40)		Choir-Concert Avery Choir Room	Geometry Sem 2 LeFebvre 308	Literature 2 Richardson 320	Earth & Space Science Bergman 305
Mod 2 9:05-9:40 (Wed. 8:45-9:20)	Assigned Mazgaj my desk	Spanish 2 White 142		Literature 2 Richardson 320	Geometry Sem 2 LeFebvre 308
Mod 3 9:45-10:20 (Wed. 9:25-10:00)			Personal Finance Leehy 233	Geometry Sem 2 LeFebvre 308	Spanish 2 White 142
Mod 4 10:25-10:40 (Wed. 10:05-10:20)	Earth & Space Science Bergman Lecture Hall		Assigned Mazgaj my desk	Choir-Concert Avery Choir Room	
Mod 5 10:40-11:00 (Wed. 10:25-10:40)	Earth & Space Science Bergman Lecture Hall			Choir-Concert Avery Choir Room	Soph Group Guidance Hansen Lecture Hall
Mod 6 11:05-11:22 (Wed. 10:45-11:02)		Personal Finance Leehy 233	Spanish 2 White 142		Soph Group Guidance Hansen Lecture Hall
Mod 7 11:22-11:44 (Wed. 11:07-11:24)		Personal Finance Leehy 233	Spanish 2 White 142		
Mod 8 11:49-12:06 (Wed. 11:29-11:46)					
Mod 9 12:06-12:28 (Wed. 11:51-12:08)					
Mod 10 12:33-12:50 (Wed. 12:13- 12:30)	Spanish 2 White 142	Earth & Space Science Bergman 305	Geometry Sem 2 LeFebvre 308	Spanish 2 White 142	Personal Finance Leehy 233
Mod 11 12:50-1:10 (Wed. 12:35-12:50)	Spanish 2 White 142	Earth & Space Science Bergman 305	Geometry Sem 2 LeFebvre 308	Spanish 2 White 142	Personal Finance Leehy 233
Mod 12 1:15-1:50 (Wed. 12:55-1:30)		Choir-Concert Avery Choir Room			Strength & Conditioning 2 Secora Fitness
Mod 13 1:55-2:30 (Wed. 1:35-2:10)	Strength & Conditioning 2 Secora Fitness	Strength & Conditioning 2 Secora Fitness		Spanish 2 White 142	Strength & Conditioning 2 Secora Fitness
Mod 14 2:35-3:10 (Wed. 2:15-2:50)		Strength & Conditioning 2 Secora Fitness	Choir-Concert Avery Choir Room	Personal Finance Leehy 233	Literature 2 Richardson 320

Figure 4.1 – A Sample Student's Schedule

4.2 Teacher Website Capabilities

The teachers' section of the website contains functionality similar to that of the students with some unique additions. First of all, teachers can look up their own schedules in the same manner as students. Once the teacher enters his/her ID number and clicks the "lookup" button, they are taken to a page containing their schedule. In addition to a listing of the teacher's class

schedule, they can also view a list of students assigned to them in each mod. This helps the teacher be aware of who is to be expected during open mods.

Teachers at WHS often participate in extra supervisions during the school day. Examples of supervisions might be monitoring a study center or cafeteria. Other supervisions include spending time with students who have special instructional needs. The website permits teacher's to input their own supervision schedules. Once supervisions are entered, they appear on the teacher's schedule when students or other faculty access it.

Teachers can also look up the schedules of students using the "Student Schedules" page. The teacher needs only to enter the I.D. number of the student of interest and click "lookup". The teacher is able to view the class and assignment schedule of the student. It is from this page that teachers may assign students to specific locations during open mods which is illustrated in Figure4.2. On the right side of the student's schedule there is an area containing three text fields and a slew of checkboxes. The textbox containing the student's ID number is prefilled and cannot be changed without first looking up another student's schedule. The next text box is for the teacher's I.D. number. The final text field is for the location that the student should report to during their assignment. The checkboxes below are for the teacher to designate which mods the student is to be assigned. When the teacher is satisfied, they simply click the "Submit Assigned Mods" button and the assignment is added to the student's and the teacher's schedules. A message for each assignment appears in the middle of the screen. If there were no conflicts with the student's schedule, the assignment is approved. Otherwise, a message appears telling the teacher that the student is already busy during that mod.

Figure 4.2 Teachers Assign Mods to Students

Teachers are also able to release students from assigned mods. In the top-right area of the page containing the student's schedule, the teacher finds two text fields. After entering the student's and the teacher's I.D. numbers, the teacher clicks the "Release Student" button. At this point, the database deletes records corresponding to all assignments concerning the student and the teacher. If the teacher wishes to reassign the student for a different mod, they can do so in the same manner as the original assignment.

4.3 Counselor Website Capabilities

Counselors have more privileges still. While they are able to look up student schedules and assign students in the same way that teachers can, counselors may also view the master class schedule and enrollment information for each class. The master class schedule contains the times and locations for every lesson in the school. From here, counselors can track the sequence of lessons in a given week. The enrollment section permits counselors to find the attendance list for each lesson.

The website offers a new way for counselors to submit the class requests of their students as shown in Figure 4.3. Previously, counselors sifted through a catalogue of 300 classes to pick the ones best suited for a student. This requires manual calculations of credit hours and the mods per week for the student to enroll in the class. Additionally, counselors had to know which classes were being offered in first, second and summer terms.

The “Student Sign-up” section of the website allows counselors to firs tenter a student’s I.D. number and then select classes from columns corresponding to the first and second semester of the school year. Each column contains drop-down menus. The first six drop downs menus in each column are prefilled with classes belonging to specific departments. These departments were chosen by counselors because they contain the majority of core courses that students take almost every semester. The remaining drop-down menus in each column contain a list of every class offered at Westside. They are alphabetically ordered by department name, sub-ordered by class name and appear only in the drop down menus belonging to the semester of the class. The number of credit hours and mods required per week of the classes that the student is requesting is updated and displayed with each selection.

When the counselor is finished with the student's class requests, they push the "Send Requests" button. This button takes the counselor to a confirmation page where each of the class requests are listed. If a mistake is found, the counselor can click the "Back to Class Requests" button return to the requests page. If the requests are found to be satisfactory, the counselor clicks the "Submit Class Requests" button. This submits the class requests to the database and sends a schedule confirmation email to the student. The email can then be shared with parents at home. If a student's class requests should change, the counselor can simply go to the "Edit Class Requests" section of the page to modify them.

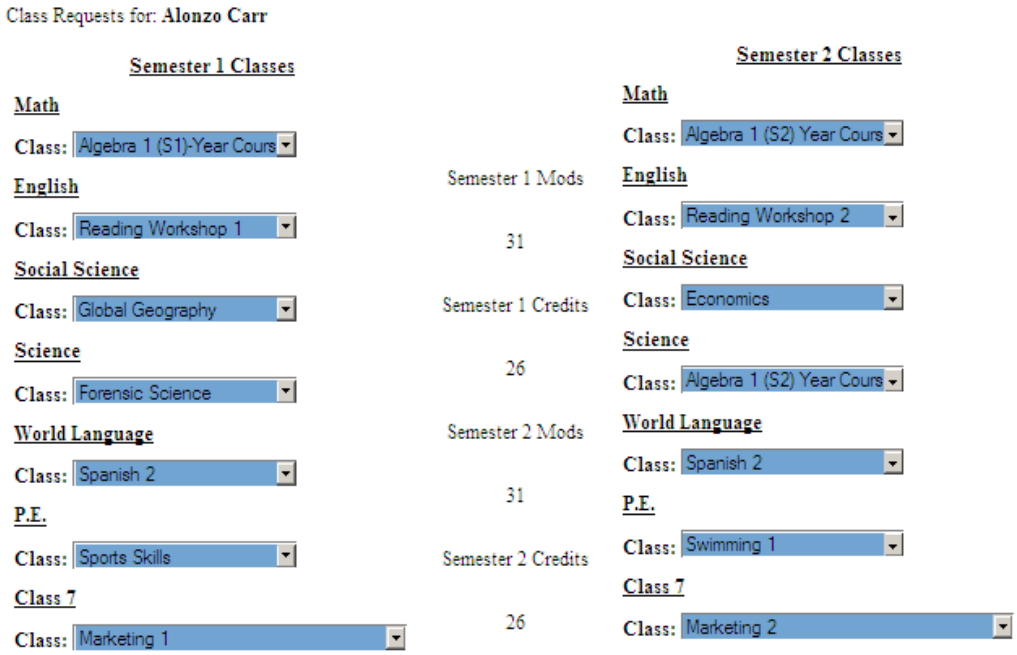

Figure 4.3 – Students' Class Requests

Counselors are also responsible for adding and dropping classes from a students' schedule. The counselor can click the "Add/Drop Classes" button on the left toolbar. If the counselor wishes to remove a student from one of their classes, the counselor simply enters the student's ID number and the ID number for the class and clicks "Remove Class from Student's

Schedule". Once the button is clicked, all instances of the class are removed from the student's schedule.

To add a class to a student's schedule, the counselor enters the student's I.D. number and selects the class they wish to add from a drop-down menu. They are taken to a page where they can select a specific teacher and view the structure of their class as shown in Figure 4.4. The counselor makes selections regarding the phases, groups and meetings in which to enroll a student . If a particular meeting does not fit into the student's schedule, the counselor is alerted. Before the website, all class additions are done by searching each section of a class schedule by hand for a set of meetings that satisfy the class structure and work for the student. This is aggravating for counselors. The website makes this process quicker and much easier.

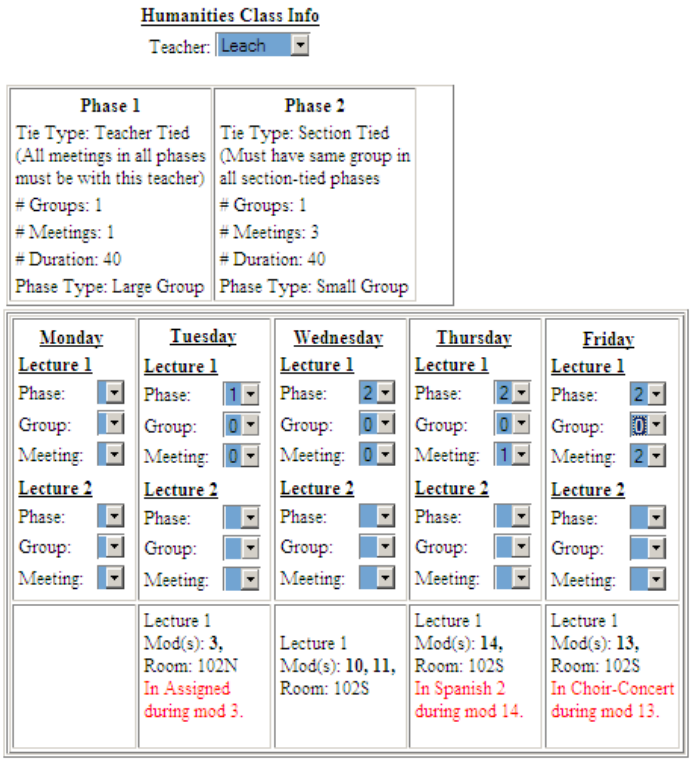

Figure 4.4 – Student Sign-up

4.4 Administrator Capabilities

Administrators have the most privileges when it comes to the website. The main page of the administrators' section contains functionality to lookup student, teacher, class and room schedules, in addition to viewing enrollment information. Like counselors, administrators are also able to add and remove classes from a student's schedule. The majority of the additional functionality in the administrator's section pertains to the control of profile data. This information is vital for scheduling as it contains personnel availabilities as well as class structure information. Specifically, administrators control data regarding students, teachers, rooms and classes.

The ability to add, edit and remove students from the database is utilized when a student transfers to/from Westside after a semester has already begun. To add a student, the administrator selects the "Edit Students" button from the toolbar, and clicks "Add a Student". A set of textboxes containing relevant information such as name, I.D. number and counselor is presented so that information can be entered. The administrator simply clicks the "Add Student" button and the data is inserted into the database.

Conversely, if a student is to be deleted from the database, the admin clicks the "Delete a Student" button, enters the student's ID number and clicks the "Delete Student" button. Once this button is clicked, the student is deleted from the database along with their schedule and enrollment information.

Editing students' information is easy as well. The administrator may enter a student's I.D. number in the "Edit a Student" section of the website. Once the submit button is clicked, the

database is queried for profile information and displays its results. The administrator can change any field and when the update button is clicked, the information is changed in the database.

Adding, editing and removing teachers is similar to adding, editing and removing students. From the "Edit Teachers" section of the website, data such as name, I.D. number, department and availability can be modified.

Profile information for rooms may also be altered from the website. Although new rooms do not appear out of nowhere, the capacity, room type (lecture, lab, etc.) and corresponding department may change from time to time. In this case, the administrator may add, edit and remove room data from the database.

Editing class information is the most intensive responsibility of the administrator. When adding or editing a class, the administrator is prompted by a page containing a large form as illustrated in Figure 4.5. At the top of the form, the administrator inputs general information about a class including name, I.D. number, department, credits, etc. Below, the administrator enters specific information for each section of the class. Each row corresponds to a specific teacher. The administrator decides upon the number of sections assigned and the main room in which the class meets. To the right, the administrator finds dropdown menus and textboxes that help shape the structure of each section of the class. The page starts in Phase 1 by default. Administrators choose the number of groups of, number of meetings, durations of each meeting, as well as room and day/mod constraints.

Once the first phase of a class is entered, the admin clicks the "Add Class/Phase" button and the class information is sent to the database. The administrator has the option to add another phase to the class or to go back and add another class. The administrator can also delete classes in the same manner as deleting students and teachers.

A major benefit of this section of the website is the ability to save class structures from semester to semester. Previously, an administrator would manually collect structure data from teachers in the form of paper forms to be filled out. This led to disorganization, wasted paper and forced an administrator to input data repetitiously every semester. The website requires about two work days to input class structures the first time. Structures from the previous semester are stored in a mysql® database for retrieval in future semesters. Administrators will need to make only minor changes from one semester to the next which requires one work day.

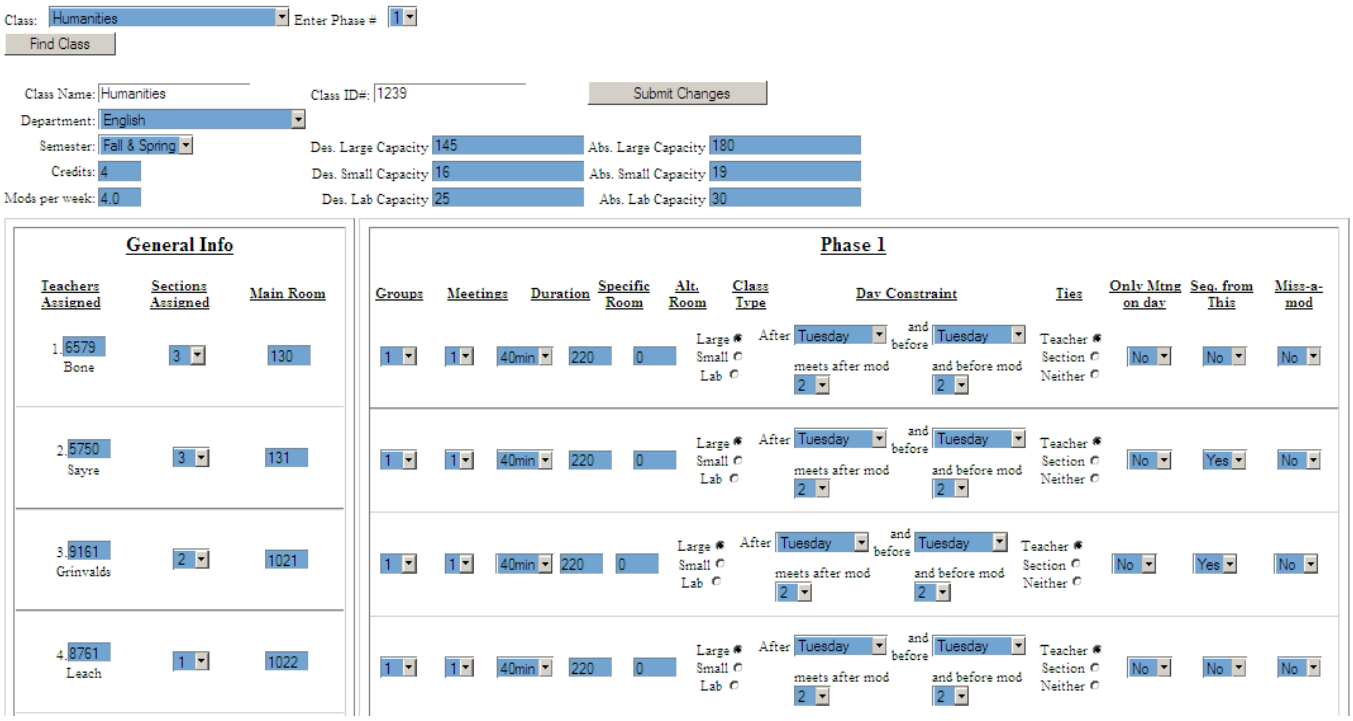

Figure 4.5 – Input Class Structures

4.5 Summary

In summary, this section introduces a website so that students, teachers, counselors and administrators can interact with the inputs and outputs of the scheduling program described in chapter 3. This website allows students and faculty to view each other schedules as well as add supervisions and assigned mods. In addition, counselors are able to process students' class requests in an organized and aesthetically pleasing manner. Finally, administrators are able to edit inputs of the program quickly and precisely.

Overall, the website acts as a valuable complement to the scheduling program described in Chapter 3. It acts as an organizational tool to plan one's day by allowing unprecedented access to student and faculty schedules. Users are able to find when/where they are supposed to be at any time by simply logging into their account. Furthermore, teachers are made aware of students' availabilities for supplemental instruction and homework help. Finally, the data management features of the website will save administrators large amounts of wasted time when modifying student/teacher profiles and class structure information.

Chapter 5

Conclusion

Westside High School utilizes a novel class scheduling system called modular scheduling. Modular scheduling gives schools the ability to affect student learning in a new way, by giving teachers the freedom to design the structure of their classes. Teachers are also able to fuse their classes together and team-teach in a way that was previously impossible. Furthermore, teachers will be able to supplement classroom instruction with individual attention to failing students during their open periods.

The Modular scheduling gives students the ability to get extra help from teachers without missing after school activities. In addition, students will learn to be flexible with their schedules as they change day to day. Time management is learned implicitly as students plan out their week's work ahead of time. These skills will hopefully result in responsible, knowledgeable students who are ready for the real world.

The drawback of modular scheduling is the laborious, time-intensive process of creating cohesive class schedules. This thesis presented the SMART Scheduler, an efficient computer program that schedules classes within a modular environment. Running the SMART Scheduler requires about two work days to input the necessary data and 5.1 seconds to schedule 3,036 lessons. This is only 5.7% of the seven weeks that is currently required to make a semester's schedule at WHS. Furthermore, updating class structures from semester to semester does not

require fresh data to be input. The accompanying database saves data from one semester to another, enabling following semesters' schedules to be made in less than a day.

5.1 Future Research

Future research will be able to improve the SMART Scheduler through the efficient scheduling of student schedules using the SMART Scheduler's class schedules. This will be necessary before the can be considered a complete package. Once students' schedules can be constructed, the SMART Scheduler will be available for use by other schools who desire to implement modular scheduling.

Improvements can be made to the class scheduler by balancing teachers' daily work-loads. Currently, the program will over assign a teacher on certain days if the class is found to be highly constrained by its day/mod constraints. An example would be a class that requires 8 40-minute meetings be scheduled in a 2 day period. The SMART Scheduler will automatically overload the first day to avoid not having enough time.

It would also be interesting to integrate one of the metaheuristics discussed in chapter 2. Originally, this approach was considered for the SMART Scheduler but was decided against based upon the difficulty of constructing a single class schedule, let alone entire neighborhoods. A metaheuristic would most likely increase computation time but might produce higher quality schedules.

Further improvements such as better website interface can also be made. In the long-term, the SMART Scheduler will need to be re-constructed using a modern language of the day.

It would be preferable if this language was entirely web-based as it would allow administrators to schedule classes remotely.

Works Cited

Abramson, D. "Constructing School Timetables using Simulated Annealing: Sequential and Parallel Algorithms." *Management Science* (1991): 98.

Akkoyunlu, E. A. "A Linear Algorithm for Computing the Optimum University Timetable." *The Computer Journal* 16.4 (1973): 347.

Al Yakoob, S. M. "A Mixed-Integer Programming Approach to a Class Timetabling Problem: A Case Study with Gender Policies and Traffic Considerations." *European Journal of Operational Research* 180.3 (2007): 1028.

Allwright, J. R. A. "A Distributed Implementation of Simulated Annealing for the Traveling Salesman Problem." *Parallel Computing* 10.3 (1989): 335.

Beligiannis, G. N. "Applying Evolutionary Computation to the School Timetabling Problem: The Greek Case." *Computers Operations Research* 35.4 (2008): 1,265.

Ben-Arieh, David. "Annealing Method for PCB Assembly Scheduling on Two Sequential Machines." *International Journal of Computer Integrated Manufacturing* 5.6 (1992): 361.

Betz, V. “Versatile Placement and Routing: A New Packing, Placement and Routing Tool for FPGA Research.” *Lecture Notes in Computer Science* 1304 (1997): 213.

Bidot, J. "A Theoretic and Practical Framework for Scheduling in a Stochastic Environment." *Journal of Scheduling* 12.3 (2009): 315.

Burke, E. K. "A Tabu-Search Hyperheuristic for Timetabling and Rostering." *Journal of Heuristics* 9.6 (2003): 451.

Busacca, P. G. "Multiobjective Optimization by Genetic Algorithms: Application to Safety Systems." *Reliability Engineering System Safety* 72.1 (2001): 59.

Campbell, J. F. "Integer Programming Formulations of Discrete Hub Location Problems." *European Journal of Operational Research* 72.2 (1994): 387.

Carlier, J. "An Algorithm for Solving the Job-Shop Problem." *Management Science* (1989): 164.

Chen, Haoxun, Jurgen Ihlow, and Carsten Lehman. "A Gentetic Algorithm for Flexible Job-Shop Scheduling." *Proceedings of the 1999 IEEE International Conference on Robotics & Automation* (2000).

Clark, John, and Derek Allen Holton. *A First Look at Graph Theory*. Singapore: World Scientific Publishing Company, 1991.

Colorni, A. *Genetic Algorithms and Highly Constrained Problems: The Time-Table Case.* Springer Berlin / Heidelberg, 1991.

Colorni, Alberto, et al. "Ant System for Job-Shop Scheduling." *Belgian Journal of Operations Research* (1994).

Costa, D. "A Tabu Search Algorithm for Computing an Operational Timetable." *European Journal of Operational Research* 76.1 (1994): 98.

Dai, J. G., Gideon Weiss. "A Fluid Heuristic for Minimizing Makespan in Job Shops." *Operations Research* 50.4 (2002): 692.

Datta, D. “Multi-Objective Evolutionary Algorithm for University Class Timetabling Problem.” *Studies in Computational Intelligence*, 2007.

Davis, Lawrence. "Job Shop Scheduling with Genetic Algorithm Research." *Proceedings of an International Conference on Genetic Algorithms* (1985).

Della Croce, F. "A Genetic Algorithm for the Job Shop Problem" *Computers Operations Research* 22.1 (1995): 15.

Dell'Amico, M. "Applying Tabu Search to the Job-Shop Scheduling Problem." *Annals of Operations Research* 41.3 (1993): 231.

DePaul, A. *Survival Guide for New Teachers: How New Teachers can Work Effectively with Veteran Teachers, Parents, Principals, and Teacher Educators*, 2000.

Dowsland, K. A. "Solving a Nurse Scheduling Problem with Knapsacks, Networks and Tabu Search." *Journal of the Operational Research Society* 51.7 (2000): 825.

Easton, K. “Solving the Travelling Tournament Problem: A Combined Integer Programming and Constraint Programming Approach.” *Lecture Notes in Computer Science*, 2003.

Even, S. *On the Complexity of Time Table and Multi-Commodity Flow Problems*, 1975.

Fagerholt, K. "Designing Optimal Routes in a Liner Shipping Problem." *Maritime Policy and Management* 31.4 (2004): 259.

Fortemps, P. "Jobshop Scheduling with Imprecise Durations: A Fuzzy Approach." *IEEE Transactions on Fuzzy Systems* 5.4 (1997): 557.

Gajda, R. "Evaluating and Improving the Quality of Teacher Collaboration." *NASSP Bulletin* 92.2 (2008): 133.

Garey, M. R. "The Complexity of Flowshop and Jobshop Scheduling." *Mathematics of Operations Research* (1976): 117.

Gaspero, L. D. "A Composite-Neighborhood Tabu Search Approach to the Traveling Tournament Problem." *Journal of Heuristics* 13.2 (2007): 189.

Gaspero, L. D., *Tabu Search Techniques for Examination Timetabling*, 2001.

Geismar, H. N. "The Integrated Production and Transportation Scheduling Problem for a Product with a Short Lifespan." *INFORMS Journal on Computing* 20.1 (2008): 21.

Gendreau, M. "A Tabu Search Heuristic for the Vehicle Routing Problem." *Management Science* 1276 (1994).

Goddard, Y. "A Theoretical and Empirical Investigation of Teacher Collaboration for School Improvement and Student Achievement in Public Elementary Schools." *Teachers College Record* 109.4 (2007): 877.

Goldberg, D. E. "Genetic Algorithms and Machine Learning." *Machine Learning* 3.2/3 (1988): 95.

Goncalves, J. F. "A Hybrid Genetic Algorithm for the Job Shop Scheduling Problem." *European Journal of Operational Research* 167.1 (2005): 77.

Gotlieb, C. C. "The Construction of Class-Teacher Timetables." *Proceedings of the IFIP Congress* (1962)

Graham, R. L., E. L. Lawler, and Lenstra, J.K. Rinnooy Kan, A.H.G. "Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey." *Annals of Discrete Mathematics*.5 (1979): 287.

Haidar, A. "Genetic Algorithms Application and Testing for Equipment Selection." *Journal of Construction Engineering and Management* 125.1 (1999): 32.

Hamiez, J. P. "Solving the Sports League Scheduling Problem with Tabu Search." *Lecture Notes in Computer Science*, 2001.

Hanen, C. "Study of a NP-Hard Cyclic Scheduling Problem: The Recurrent Job-Shop." *European Journal of Operational Research* 72.1 (1994): 82.

Hurink, Johann, Bernd Jurisch, and Monika Thole. "Tabu Search for the Job-Shop Scheduling Problem with Multi-Purpose Machines." *OR Spektrum* 15 (1994): 205.

Hutchins, Pat. *Principal, Westside High School*, 2009.

Jagielska, I. "An Investigation into the Application of Neural Networks, Fuzzy Logic, Genetic Algorithms, and Rough Sets to Automated Knowledge Acquisition for Classification Problems." *Neurocomputing* 24.1-3 (1999): 37.

Kelchtermans, G. "Teacher Collaboration and Collegiality as Workplace Conditions - A Review." *Zeitschrift Fur Padagogik* 52.2 (2006): 220.

Kelley Jr, J. E. *Critical-Path Planning and Scheduling*, 1959.

Khazzaka, J. "Comparing the Merits of a Seven-Period School Day to those of a Four-Period School Day." *The High School Journal* (1997): 87.

Lewis, R. "A Survey of Metaheuristic-Based Techniques for University Timetabling Problems." *OR-Spektrum* 30.1 (2007): 167.

Lin, Shyh-Chang, Erik D. Goodman, and William F. Punch III. "A Genetic Algorithm Approach to Dynamic Job Shop Scheduling Problems." *Proceedings of the Seventh Conference on Genetic Algorithms* (1997)

Linkens, D. A. "Genetic Algorithms for Fuzzy Control. 1. Offline System Development and Application." *IEEE Proceedings: Control Theory and Applications* 142.3 (1995): 161.

Mares, C. "An Application of Genetic Algorithms to Identify Damage in Elastic Structures." *Journal of Sound and Vibration* 195.2 (1996): 195.

Neufeld, G. A. "Graph Coloring Conditions for the Existence of Solutions to the Timetable Problem." *Communications of the ACM* 17.8 (1974): 450.

Nowicki, E. "A Fast Taboo Search Algorithm for the Job Shop Problem." *Management Science* (1996): 797.

Qu, R. "A Survey of Search Methodologies and Automated System Development for Examination Timetabling." *Journal of Scheduling* 12.1 (2009): 55.

Rahmat Samii, Y. *Electromagnetic Optimization by Genetic Algorithms*, 1999.

Rossi Doria, O. *A Comparison of the Performance of Different Metaheuristics on the Timetabling Problem*. 2740 Vol. , 2003.

Sanchez Arroyo, A. "Determining the Total Colouring Number is NP-Hard." *Discrete Mathematics* 78.3 (1989): 315.

Shin, K. S. "A Genetic Algorithm Application in Bankruptcy Prediction Modeling." *Expert Systems with Applications* 23.3 (2002): 321.

Souilah, A. "Simulated Annealing for Manufacturing Systems Layout Design." *European Journal of Operational Research* 82.3 (1995): 592.

Taillard, E. "A Tabu Search Heuristic for the Vehicle Routing Problem with Soft Time Windows." *Transportation Science* 31.2 (1997): 170.

Tan, C. C. R., and J. E. Beasley. "A Heuristic Algorithm for the Period Vehicle Routing Problem." *OMEGA International Journal of Management Science* 12.5 (1984): 497.

Wong, Y. "Critical Path Analysis for New Product Planning." *Journal of Marketing* (1964): 53.

Xia, W. "An Effective Hybrid Optimization Approach for Multi-Objective Flexible Job-Shop Scheduling Problems." *Computers Industrial Engineering* 48.2 (2005): 409.

Yamanaka, H. "Application of Genetic Algorithms to an Inversion of Surface-Wave Dispersion Data." *Bulletin of the Seismological Society of America* 86.2 (1996): 436.