Spring 4-13-2018

Modular Scheduling System for Westside School District

Tyler Bienhoff
University of Nebraska-Lincoln

Modular Scheduling System for Westside School District

An Undergraduate Honor Thesis
Submitted in Partial fulfillment of
University Honors Program Requirements
University of Nebraska-Lincoln

By
Tyler Bienhoff, BS
Computer Engineering, Mathematics, & Computer Science
College of Engineering & College of Arts and Sciences

April 13, 2018

Faculty Mentor:
Byrav Ramamurthy, PhD, Computer Science and Engineering
Jeremy Suing, MS, Computer Science and Engineering

Abstract

Westside School district offers a modular scheduling system for their high school that is more similar to a college schedule than the typical high school system. Due to the complexity of their master schedule each semester, there are no commercially available products that can assist in creating a schedule. Hence, this thesis discusses a scheduling algorithm and management system that was built specifically for Westside High School with the potential to be expanded for use by other interested schools. The first part of the paper is focused on gathering input from students and faculty for which courses and how many sections of each class should be offered per semester. This is more complicated with a modular system since each class meeting is inputted and scheduled independently. Previously thousands of pieces of paper were used to gather information about student course interests and what courses would be offered by each department, but this has been completely transitioned to use the new system. Finally, the algorithm and courses constraints are discussed in detail, describing how restrictions imposed on class meetings to produce a valid and well-constructed solution. Many of the constraints are simple, such as making sure only one class is in a room at a time. However, other restrictions enforce that teachers have a lunch break and are not overworked as well as enforcing that that schedule solutions can quickly be found.

Key Words: computer science, scheduling algorithms, modular scheduling

Acknowledgements

I would like thank Dr. Byrav Ramamurthy and Jeremy Suing for providing me guidance in writing and completing this thesis, in addition to their hard work in helping with the project.

I am grateful for Westside School District providing the opportunity for this challenging task and always being supportive. Further, I appreciate the Raikes Design Studio for facilitating the project and helping with administrative support.

Finally, and most importantly, I want to thank Camille Hoover for encouraging me to complete the thesis and for always putting up with my many complaints. Without her support I never would have had to motivation to complete this paper.

Contents

1 Introduction

1.1 Background

Westside School District in Omaha, Nebraska uses a modular schedule system in their high school which is similar in style to a university schedule. Instead of the traditional six to eight period days where students have each class every day at the same time, with the modular system, each day is split into 14 time periods called “mods” of approximately 20 or 40 minutes each. Depending on the course, it can meet one or more times throughout the week at different times and for different lengths of time (40, 60, or 80 minutes). Additionally, a class can be set to have “large group” meetings where all students attend a single, large lecture and “small group” meetings where a student is schedule for one of several possible meetings with about 15 to 20 total students. These smaller meetings provide more intimate and interactive group discussions with peers as well as closer interaction with faculty members. For example, a physics class may have a single lecture each week for all students, six small group classes which each student is scheduled to only attend one, and six additional small group labs that are 80 minutes long to provide additional time for hands-on experience.

While most high schools continue to use the traditional period day, Westside has highlighted some key advantages in their modular design that they continue to use since its adoption in 1967. This style allows students to take more classes, specifically elective courses such as fine arts or industrial technology that they are interested in and allows for students and teachers to have time during the day to meet on a one-to-one basis. As the modular style is more similar to college course schedules, students are able to be involved in making decisions

with their use of time and are more prepared for life as a university student. Finally, most students who have used both the traditional and modular schedules tend to prefer the modular systems.

One major con that schools have when attempting to adopt a modular course schedule is the complexity in creating a master schedule of courses for each semester. While many systems exist for creating the traditional schedule, there are not commercially available products that create a modular master schedule which meets the needs of interested high schools including Westside. For the past 30+ years, Westside has used a DOS program with direct support of the original developer to create each schedule which each take more than a week to generate. Thus, Westside is interested in obtaining a new state of the art and efficient system that can generate modular schedules with the possibility to be generalized for other schools who are looking at switching to the modular system. To go along with the course generation algorithm, Westside is also interested in a modern website that can be used by students, teachers, and administrative faculty members to view current schedules and course catalogs, sign up for classes, and generate master schedules for each semester.

A previous Raikes Design Studio team began development on this product and completed most of the preliminary features of the web interface for the project. However, the website was not feature complete and required constant support to fix bugs and improve different components during the first year of live use by Westside. Furthermore, some basic steps had been completed for the framework of the scheduling algorithm, but it had never been tested with actual data, contained numerous bugs, and was lacking the majority of its functionality required to be used to create master schedules for Westside.

1.2 Contributions

This project continues the previous development of the Westside modular scheduling algorithm as well as portions of the web interface which interact directly and indirectly with that system. The department course request page was updated to improve data input handling and allow preprocessing of the data. The scheduling algorithm was essentially completely rewritten along with thorough testing to verify the accuracy of the program. The system is well documented to allow for continued development by future team members.

1.3 Thesis Organization

This paper breaks down the components of the system that are used to create a modular master schedule. This begins with the web interface where school administrators can enter information about each course to be scheduled for the next semester called Department Course Requests. Next, the implementation of the algorithm is investigated with how it models the problem and imposes necessary restrictions to create a well-formed schedule. Finally, the interface used for creating jobs and viewing the results of previously runs is examined in detail.

2 Department Course Requests

One of the most important parts for the overall scheduling system is efficiently gathering information from students, teachers, and administrators about courses that should be scheduled for the coming semester. Previously, all this information was handled through thousands of papers that had to be manually processed, but now it is all completed much more efficiently through the new website. The process begins with students filling out information on a website for the courses they are interested in taking. From this information, department

heads see how many students are interested in taking each class, for example, regular Physics compared to AP Physics. The school can offer a varying number of classes to accommodate the interests of the students and create the department course request for each class.

2.1 Course Request Phases

Each department course request is split into phases that represent each unique time a course meets, and each phase can be split into multiple sections (possibly with different teachers) which means the course is offered multiple times. A student is required to be in exactly one section in each and every phase but doesn't necessarily need to be in the same section of each phase. A typical class with have 3-5 phases and 1 to 6 or more sections to accommodate enough classes of students. Similar to a university course structure, for example, Physics has 5 phases: a single large group lecture that every student enrolled must attend, 3 small group phases each with 6 sections, and a lab phase with 6 sections that is double the length to accommodate more hands-on experiment time.

A sample phase from the Physics course is shown in Figure 1. The important information for the scheduling algorithm includes the mod length for how long the class should be where 40 minutes is the typical length and 60 or 80-minute options are available for longer lab type meetings. Additionally, for each section one or more teachers must be chosen to lead the course, and a room must be selected which are vital in creating the master schedule to avoid overbooking teachers and rooms. Finally, there are optional fields to select the exact day and time or simply just the day of the week when a class should occur. This is imperative for certain

courses which must meet at a certain time, and it also helps later during scheduling to have some course times to build the schedule around and reduce the solution search size.

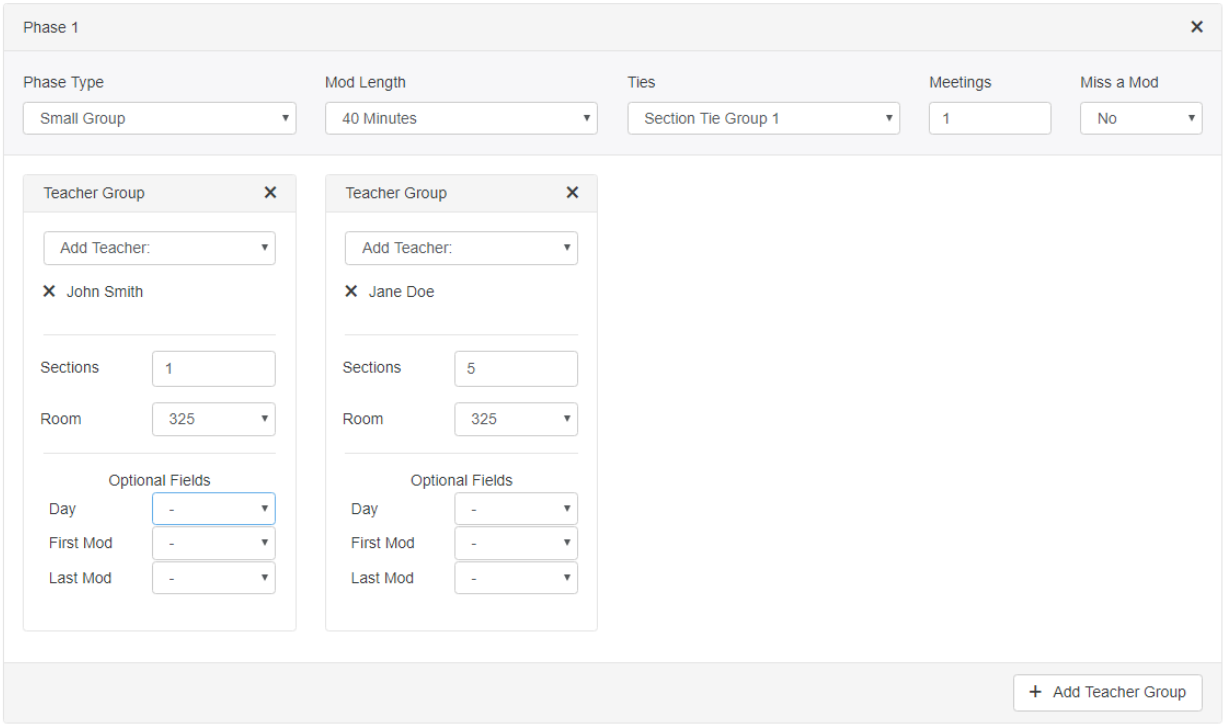

Figure 1. Example Department Course Request

2.2 Course Request Data

One major problem encountered with using this online form for data input is the high likelihood of invalid or incorrect information being included. For example, it is easy to accidentally input 10 sections instead of 1 section or select the incorrect room from the dropdown menu. As will be discussed in more depth later, the scheduling algorithm requires certain requirements to be met (e.g. only one class per room at a time) to produce a valid schedule. If these requirements are not met, the algorithm will not output a solution; however, it will often take several hours to determine this result, and the error is typically not obvious.

To help combat this issue, the course request page was updated to prevent invalid input from being included with regards to rooms and teachers. If a room (or teacher) is already booked during a certain mod as specified by the optional fields, then no other section with that room (or teacher) can be scheduled at that time, and a friendly error message will be presented. Furthermore, a preprocessing step is included that should be run before the full algorithm is executed. This step checks for errors over the entire list of course requests for things such as a room (or teacher) being scheduled more than the possible mods in a week. In addition, it provides warnings about possible typos in the data for courses that might be overscheduled with a large number of sections. While it is not necessary to use this preprocessing step, it helps to identify possible problems in the course requests that may prevent a valid schedule from being produced.

2.3 Schedule Complexities

Much of the complexity in creating in a modular scheduling algorithm for Westside is that course requirements drastically change between Fall and Spring semesters as well as between years. While some courses like English are based mainly on the number of students requiring the course for graduation and can easily add or remove sections depending on the semester, other elective courses are structured on student interest level that may differ drastically in the number of sections needed per semester. Even a course that only has one section and meets the same number of times each semester, such as Jazz Band with four class times a week, will have considerably different meeting times from semester to semester. Thus, not only do students have to be adaptable to the modular system, but the scheduling algorithm

must be robust enough to accommodate these changing requirements from semester to semester in addition to major multiyear shifts in course developments.

2.4 Student Scheduling Algorithm

As previously mentioned, the course requests are partially based off the number of students interested in taking a course, and these requests are the direct input into the scheduling algorithm. However, individual student course interests are not considered when creating the master schedule. There is a separate student scheduling algorithm that takes as input a master schedule and student course preferences and produces schedules for each student. This separate algorithm is not discussed in detail in this paper.

3 Master Schedule Algorithm

This section examines the representation of courses and the modular schedule system in code and how the system works to create a valid master schedule. Additionally, the constraints or restrictions imposed on the schedule are investigated which prevent, for example, rooms and teachers being overbooked and allow a valid solution to be created.

3.1 Schedule Modeling

One of the unique quirks of the Westside modular schedule is the difference in mod lengths. Mods 1 to 3 and 12 to 14 are each approximately 40 minutes in length while mods 4 to 11 are each roughly 20 minutes long to allow for some classes of 60-minute durations in the middle of the day. This difference in length makes the problem hard to model in code; thus, in the scheduling algorithm, the six mods of 40 minutes are split into two separate mods, as display in Figure 2. Therefore, the day is split into 20 time blocks each of about 20 minutes, and

the school week contains 100 time blocks which will be referred to simply as mods when dealing with the scheduling algorithm. Each of the 100 mods in the code is represented by an integer between 0 and 99, with, for example, Tuesday being represented by mods 20 through 39. This allows the algorithm to simply assign integer values for the time of each course meeting, such as assigning 22 and 23 to a class during Mod 2 on Tuesday, as shown in Figure 2.

	Monday	Tuesday	Wednesday	Thursday	Friday
Mod 1	0	20	40	60	80
	1	21	41	61	81
Mod 2	2	22	42	62	82
	3	23	43	63	83
Mod 3	4	24	44	64	84
	5	25	45	65	85
Mod 4	6	26	46	66	86
Mod 5	7	27	47	67	87
Mod 6	8	28	48	68	88
Mod 7	9	29	49	69	89
Mod 8	10	30	50	70	90
Mod 9	11	31	51	71	91
Mod 10	12	32	52	72	92
Mod 11	13	33	53	73	93
Mod 12	14	34	54	74	94
	15	35	55	75	95
Mod 13	16	36	56	76	96
	17	37	57	77	97
Mod 14	18	38	58	78	98
	19	39	59	79	99

Figure 2. Real mods compared to algorithm mods.

3.2 Algorithm Basics

Each meeting of a course is scheduled independently, but restrictions are imposed to properly structure the meetings of each course. Initially meetings with a specified time are placed into the schedule as these courses must be at the given time, and no conflicts should exist from the preprocessing step. Then the algorithm goes through each classroom and specifies that each remaining course must meet during the available time slots in the room. Classes can only begin at certain mods, since some time blocks are fractions of reals mods. However, this actually simplifies the process as 40-minute courses can only start at each 40-minute interval which are represented by even mods in the code. Similarly, 80-minute courses can only begin at each 80-minute interval which in each day are mods 0, 4, 8, 12, and 16. Further, 60-minute classes must begin or end with a 20-minute mod (real mod 4 to 11), and therefore in the algorithm begin with any mod between 4 and 15, inclusive. Additionally, this has the added benefit of preventing courses from being scheduled during mods on consecutive days (e.g. the last mod on Wednesday and the first mod on Thursday).

3.3 Room Constraints

The first constraint the system imposes is that each course in a room must meet at a different time and cannot overlap with another course (with a few exceptions). This is easily implemented in the code by preventing the start and end mods of two classes from overlapping. For example, if a class meets beginning at mod 24 and lasts until mod 27, every other class in the room must end before mod 24 or begin after mod 27. Nevertheless, there are some exceptions to this “one class per room” constraint that must be taken into account. For instance, “Out” is considered a room in the system but it used as a miscellaneous placeholder

room for teacher meetings or for courses that are not held at the school (e.g. college courses at UNMC that students can take). Obviously, this "room" needs the ability to have multiple courses scheduled in it at the same time; thus, the constraints must ignore this room. Additionally, the gym is capable of accommodating two separate classes at the same time provided they are taught by different instructors. Therefore, two courses (but no more) can be scheduled in the room, but constraints must be enforced to prevent a third class at the same time.

3.4 Teacher Constraints

Constraints on teachers are very similar to room constraints but more restrictive. Clearly, a teacher cannot lead two different classes at the same time, so the same no overlap room constraint is applied for teachers. Additionally, to spread out a teacher's day and week, no one is allowed to teach more than three 40-minute or two 80-minutes classes in a row, and each teacher is required to have a lunch break at a reasonable time in the day. This cannot be done by simply preventing three 80-minute courses in a row in the algorithm as it does not explicitly distinguish which mods are on each day. For example, it is allowable for a teacher to end a day with two 80-minute classes and begin the following day with an 80-minute course even though the algorithm interprets the teacher working three courses in a row. Thus, the constraints must be created so that only within the mods of each day are restrictions placed to prevent overloaded schedules.

3.5 Additional Schedule Constraints

Several more complicated constraints are used both to form a good schedule and impose restrictions on the problem to reduce the large search space. The first constraint is that meetings within a course phase must be grouped together. That is, if a course has two small group phases (denoted phase 1 and phase 3) each with five sections and one large group lecture (denoted phase 2), then all five sections in phase 1 must come before the large group lecture followed by all five sections in phase 3 later that week. This is aided by the fact that when filling out the department course requests, the phases are generally listed in the order during the week they should occur. For instance, if a Physics course wants the large lecture early in the week and lab sections at the end of the week, the large group will be phase 1, the labs will be the last phase, and other small group meetings are placed in the remaining middle phases. Similar to the previous constraint, a class will typically only allow one phase per day (unless noted on the course request), so that students do not have the same course multiple times in a day. Of course, multiple sections of a course will be on the same day, but each student will only be in one of these sections.

3.6 Efficiency Constraints

Finally, a few additional constraints were included to improve the runtime of the scheduling algorithm without removing any possible solutions. The primary enhancement included is if two class meetings are exactly the same, then the algorithm arbitrary selects one to be meeting to precede the other one. For the meetings to be the same, they must have the same teacher(s), meet in the same room, and be part of the same phase of a course. Since the meetings are the same, they are interchangeable in the master schedule, and there is no

practical difference for which one comes first or second. Nevertheless, the inclusion of this constraint allows for the search size for solutions to be cut in half for every pair of course meetings it applies to. While not completely necessary, this efficiency helps to identify solutions or determine that none exist much more quickly and is extremely beneficial since it can take several hours up to a day for the scheduling algorithm to fully execute.

3.7 Schedule Creation Interface

The interface for generating master schedules is one of the many administrator only pages on the website and is very simple to use as shown in Figure 3. There are three inputs to the page: the semester to schedule courses for, the maximum number of hours to run the scheduling algorithm for, and the number of computation nodes that should be used. The website is hosted on Microsoft Azure, so generating a schedule provisions separate computation nodes in Azure to be used exclusively in executing the algorithm. A typical schedule can be generated in approximately a few hours, and the algorithm will continue creating new schedules until either it checks every combination of class meeting times or the much more common case of the timeout being reached. The page displays a list of previously run jobs with their status of “Running” or “Completed.” Clicking on a completed job shows a list of potential master schedules and allows an administrator to apply a schedule to the online system for the specified semester.

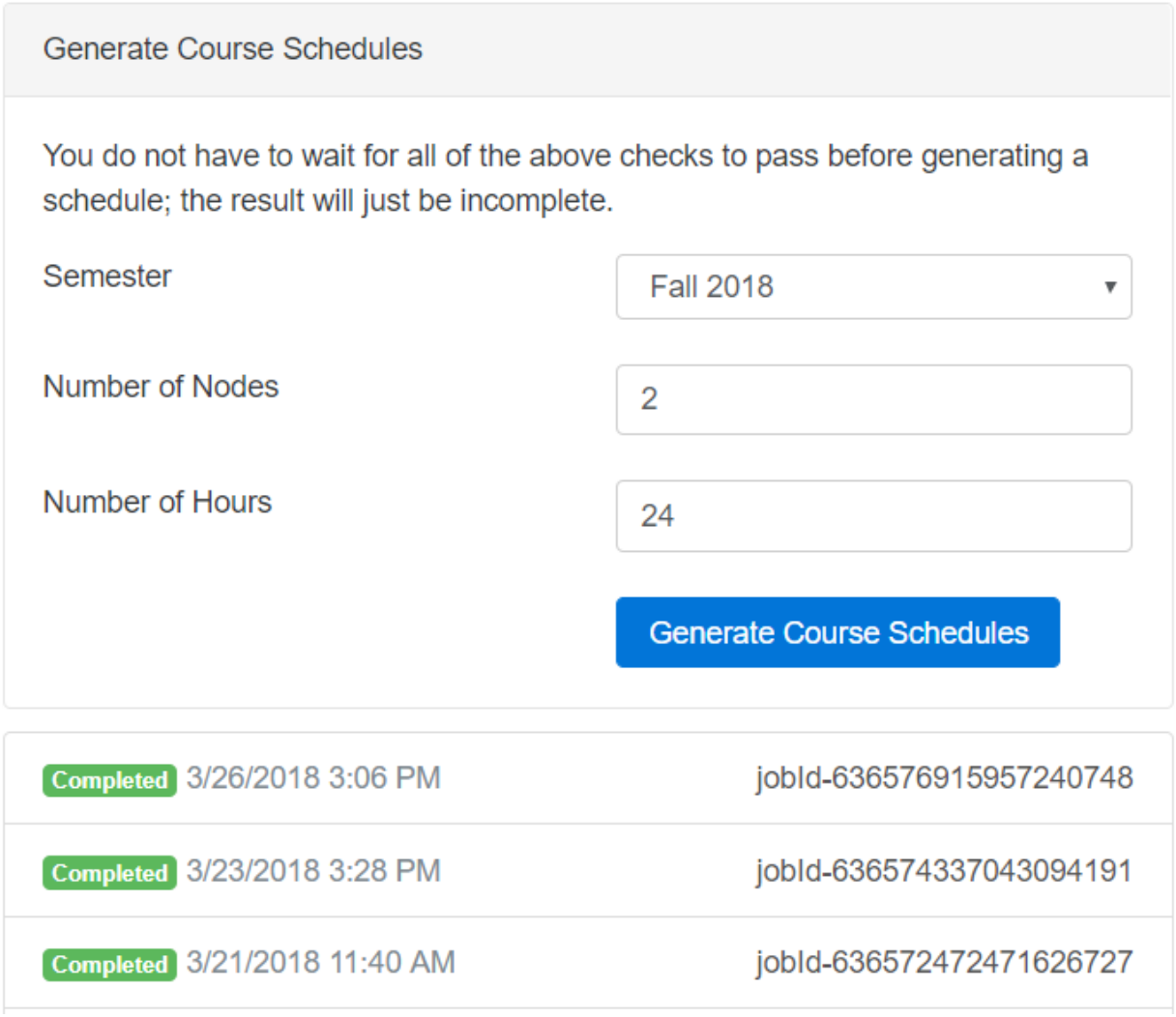

Figure 3. Web interface for generating schedules.

4 Conclusion

In this thesis, the concept of modular scheduling in high schools, specifically at Westside High School was introduced. The department course requests were described that are used for the input into the algorithm to create the unique schedule for each semester. Additionally, the ideas and constraints used in the scheduling were thoroughly examined.