Adv. in Comput. Sci. & Eng. **1**(1) (2007), 25-46

COURSE SCHEDULER: AN AUTOMATED SCHEDULE GENERATOR

TAREK SOBH, RIK COUSENS and SAROSH PATEL

Department of Computer Science and Engineering
University of Bridgeport
Connecticut, U. S. A.
e-mail: sobh@bridgeport.edu

Abstract

A typical problem in a college of engineering is trying to find the best set of courses to offer students in a given semester, taking into consideration which courses are needed by students, and the availability of instructors capable of teaching those classes. The optimal solution, of course, is to offer all courses that will allow all students to graduate in the minimum number of semesters. This allows students to finish their course-work quickly so that they can enter the work force.

The job of determining which courses to schedule (usually done by a department chairperson) requires information about what the student-body needs and information about instructor availability/capability. The most difficult part of this process is to determine which instructors should teach which courses, and in which time-slots they should teach them, and still be able to satisfy all of the students.

Selection of instructors is determined by the fact that an instructor knows the course material and that he/she is available to teach a given course. Instructors may have preferences as to what time of day to teach, or on which days of the week they would like to instruct the courses for which they are responsible. Instructors may not be available if they are already teaching too many courses, as defined by the individual learning-institution.

2000 Mathematics Subject Classification: 65, 68Wxx.

Keywords and phrases: optimization, scheduling, automated schedule generation, accreditation, course scheduling.

Received January 5, 2007

There are several difficulties in determining when to offer a given course. For example, consider an engineering major senior who has taken almost all of the courses required to satisfy the requirements for his/her degree. Because he/she has nearly satisfied the degree-requirement, the student has a limited choice of courses to study. This is a problem for the department chairperson, since they must offer appropriate courses to the student to ensure he/she will have enough credits to graduate. This document will present a set of algorithms and software components which will aid a department chairperson in the scheduling process.

1. Introduction

In an institution of higher-education; for example, a college of engineering, there are students who wish to acquire a degree in a given discipline.

A *student* is a person who wishes to acquire a degree in a given discipline. By attending lectures, reading textbooks, and completing projects, the student assimilates the information, increasing their knowledge about the discipline. In order to obtain the degree, the student must take a prescribed set of courses in order to satisfy the degree-requirements of the institution.

The information that the *student* must learn (as dictated by the learning-institution) is disseminated to the students by an *instructor*. The *instructor* has already acquired the information required by either having learned the course-material in an engineering educational setting, or acquired it by themselves by having worked in that field. Furthermore, each *instructor* is typically capable of instructing more than one course.

In order to enable *students* to satisfy their degree-requirements, a schedule must be defined. A *schedule* is a set of courses, each with an *instructor* to teach it, whose responsibility it is to disseminate the course-information to the *students*.

In order to generate a schedule of classes the **CourseScheduler** application (one of the tools in this project) requires the following information:

(1) A data-set of courses needed by the students. Preferably, this data-set should contain a “tree” of possible ways that a given student could graduate in the minimum number of semesters.

(2) A data-set of instructors' abilities. This data-set should indicate which courses a given instructor is capable of teaching to students.

Luckily, the first criterion for **CourseScheduler** is available by using the SKED program[1] [4].

2. SKED

SKED evaluates courses already taken by students (transferred from other Universities or previously taken at the University of Bridgeport), and determines the 'best' courses that the student can take, given:

- courses already taken (or transferred) by the student.
- the *pre-requisite* courses for a given course (a *pre-requisite* being a course that must be taken before another course, since the *pre-requisite* gives the student 'foundation' information required to understand the concepts presented in the 'next' course).
- the *co-requisite* courses (courses that may be taken *at the same time* as the given-course, since the information obtained in the *co-requisite* is not dependent upon then given-course, but will be helpful to the student, if learned at the same time).
- the maximum number of courses allowed by the university. This 'restriction' exists to make sure that students are not over-burdened by taking too many classes. This is supposed to guarantee that students have sufficient time to concentrate on their homework, lab-work, exams, etc.
- and last, but not least, the courses required to satisfy the *major* for which the student desires a degree (a *major* being an area of concentration in which the student is interested. Universities require a student to take a sufficient number of courses [*credit-hours*] in a given discipline so that the student is conversant with many aspects of the desired *major*).

There also exist certain courses that may not be taken by students until they have reached a certain 'year' (i.e., *freshman, sophomore, junior,*

[1] Also developed at the University of Bridgeport.

or *senior*: these terms indicate the number of credit-hours successfully completed by the student). This 'restriction' exists to 'protect students from taking courses for which they may have insufficient 'background' information to complete successfully[2].

In order to produce this data-set, **SKED** requires the following information:

- courses previously satisfied.
- courses being offered next semester.
- courses being offered the semester after next.
- prerequisite courses to a given course.
- co-requisite courses to a given course (if any).
- the maximum number of courses that each student is allowed to take.

The **SKED** algorithm calculates a *requirement-cost* (the maximum number of pre-requisite courses that must be taken before a given-course) for each course, as well as an *availability-cost* (which is the number of pre-requisites, co-requisites and course-offerings in the next 2 semesters)[3] and generates a data-file[4] containing a 'tree'. This 'tree' contains all possible schedules (list of courses in a semester-by-semester format), describing which courses, taken in which semester, would allow the student to satisfy the degree-requirements in the minimum number of semesters. **SKED** is written in Microsoft Visual Basic, and uses Microsoft Access as its data-source.

The **CourseScheduler** application manipulates the output-files from **SKED** and provides a simple infrastructure to solve an arduous problem: determining *which* courses to offer in a given semester that will allow all students to have *at least one* of their 'required' schedules (as determined by **SKED**) satisfied, given student requirements and instructor availability.

[2] The above-mentioned terms are more clearly defined in the **SKED** [4] paper, Introduction section.

[3] See the **SKED** paper, Algorithm section.

[4] for a single student

2.1. Trials (and 'Errors') – A.K.A. algorithm refinement

In the initial phase of this project the goal seemed simple:

- determine which courses to offer given
- available instructors to teach the classes
- classroom size and availability
- time slots of when classrooms were available and instructor availability
- student's preferences as to times courses were taught

The development effort began with this goal and these requirements in place. It soon became apparent that this was an *NP-hard* problem, meaning that the process of trying to find a solution with this number of variables would require an enormous number of resources and processing time [3].

This first approach attempted to generate a set of schedules based upon faculty ability and availability. The schedule-generation and the matching of students needs to those schedules took between two to four hours, since many of the schedules generated contained 'unnecessary' courses according to the student requirements.[5] Given the poor performance of this version it was abandoned without even attempting to address the classroom or student preferences.

The second attempt was to try to generate schedules which we would:

- maximize the number of students per class
- maximize the number of student who would graduate early
- maximize the number of student preferences satisfied by the schedule

[5] The data-set used for this set of tools contains 20 instructors (Computer Science staff at University of Bridgeport) and 19 data files containing student requirements as generated by **SKED**.

This scenario would have been ideal had it worked out. In this way, we could have maximize the classroom utilization as well as the instructor utilization. The fundamental problem persisted: *our algorithms simply did not address the student's needs.*

Attempting to cater to the instructors preferences of when they wanted to teach courses, or trying to allow the student's desires on when they would prefer to take the class. This scheme did not work well. The approach that was finally chosen was the following:

- treat the student requirements[6] as a 'set'
- assign instructors to courses that 'need to be taught' from the student requirements
- allow instructors to assign a desired time slot to each of the courses that they teach

By using these rules, courses that the student requires in order to graduate early is an integral part of the scheduling process, and not left to chance. It also allows instructors some lee-way in determining when they want to teach (i.e., morning/afternoon or weekends).

In the event that an instructor does not have a full class load, the department chairperson may have that instructor teach an elective course[7].

3. Algorithm

It is important to note the following: when a student first begins their degree-program, they have some flexibility as to which courses that they can choose. It is desirable for the student to take courses that are prerequisite to other's so as not to delay their graduation. Once the major prerequisite courses are taken, the student has the most amount of flexibility in taking both required courses and/or electives. Towards the end of their program ('Senior year'), they are likely to have significantly

[6] As determined by **SKED**.

[7] Assuming the instructor wishes to teach additional courses.

less flexibility, since they MUST take certain courses to graduate, and have likely taken most of their electives.

The reason to note this is that **SKED** generates ALL possible permutations of courses that a student needs to take in order to graduate 'in the least amount of time'. There are several cases where, for students who are at the end of the program, there exists only a single permutation of classes that they need in order to graduate [in the least amount of time].

It is these students who have only a single course-set that must be offered which seems to make this task of course-scheduling so very difficult.

3.1. CourseScheduler – a batch process

The primary algorithm for the course-scheduling process is to

(1) gather all permutations of student requirements from Sked-output files. From the resultant information, courses that exist in all students schedules are gathered and saved. The usage of this information will be discussed later.

(2) gather instructor information - verify that all courses that are 'required' by the students can be offered. There may be circumstances that prohibit a course from being offered, such as an instructor being on sabbatical[8].

(3) At this point, the system has the following information:

- the fact that all courses required by the student-body can, in fact, be scheduled (at least initially) given the fact that there exist instructors who are capable of teaching all courses.
- a set of courses that are common to all students (may be an empty-set). At the very least, the system knows what the students require.

[8] This may have dire consequences for students with only a single schedule. If the 'unavailable' instructor taught a required-course, and if no other instructor in the department who could teach the required-course, this process would not succeed.

(4) Assuming that the above steps are successful, the batch-process now creates data-files to be read by the scheduling-applets.

- course-map – A list of all possible courses required by the student-body. This file contains the mnemonic names of the courses (i.e., MATH227, CS102, etc.)
- default-courses – An index-file containing the ordinal number of the courses actually needed by the student-sample.
- Student.XXX.map – one of these files will be generated for almost every single student in the input student-sample. A file will NOT be generated if Student-F has the same exact requirements as Student-D, for example. Each of these files contains all possible required-schedules. Each of the required-schedules has been 'reduced', or 'factored'[9]. Each line of the file contains the ordinal 'index-values', into *course.map*.

The **CourseScheduler** application does its processing in 3 steps:

(1) **CourseScheduler** makes a special-case for students with only a single course-set. For these students, all courses that they require are considered **extremely** important, since failing to cater to these specific needs will fail to attain the goal of minimizing the graduation time for all students.

For those students (if any) who have only a single course-set, a list of required-courses is built. These required-courses must be taught. CourseScheduler then iterates through all other students, removing the required-courses. This is done so as to reduce the number permutations that must be generated in Step 3. So, the required-courses are removed from all other students that have multiple schedules, leaving those students with only those courses that are required by the individuals above and beyond the required-set. In many cases, by removing the required-courses, many duplicate requirements appear for individual students, and are removed.

[9] as described later

At this point, CourseScheduler iterates through all students with multiple course-sets, and generates a schedule of courses required by all students. The number of permutations is significantly reduced by the factoring-out of the required-courses (from 33,000,000 to 600,000) for the current data-set. For example, in a 2 student scenario: if Student 1 has 10 permutations, and Student 2 has 3 permutations (after the reduction of the required-courses), there will be 30 (10*3) possible schedules. Under certain conditions, the complete course-set may be a duplicate of a course-set previously calculated. In this case, the newly generated course-set is not added to the list of possible schedules (since it is a duplicate). All unique schedules are written to disk for later evaluation. Given our current data-set, out of 604,800 permutations of all student requirements, only 24 are unique. Each of these 24 schedules must be evaluated against the courses that our instructors are capable of teaching.

(2) **CourseScheduler** examines the required-course set and the new minimal set of courses required by all students and verifies there are, in fact, instructors capable of teaching those courses, and displays errors messages if any courses are missing. Also, if there is an instructor who teaches courses that are not required this semester, it displays this fact, and removes the instructor from the instructor list.[10]

(3) **CourseScheduler** now knows which courses must be taught and which instructors teach them. It then permutes the instructors and the courses that they can teach[11]. Instructors are limited to a certain number of courses that they can teach each semester.[12]

[10] describe **ELECTIVES** here.

[11] This permutation is done due to the fact that there is typically overlap in course-teaching ability among the faculty. For example, there is usually more than one instructor within the Math department that is capable of instructing CS 212 (Discrete Math), which is a prerequisite course at U.B. for several courses. Depending upon the availability of the instructor, and student-requirements the course may need to be split into multiple sections.

[12] This restriction exists so that instructors will have sufficient time to: prepare for teaching their classes, correct exams, meet with students, participate in faculty meetings, etc.

U.B. allows instructors to teach 3 courses per semester. If a given instructor is capable of teaching only 3 courses that are needed (as determined by CourseScheduler), no permutation is required. Otherwise, the number of permutations possible for a given instructor is defined by the combination formula

$$C(n, r) = \frac{n!}{r!\,(n-r)!},$$

where n is the number of courses for which the instructor is capable of teaching and is required by the students, and r is the number of courses per semester that an instructor can teach. So, for example, if an instructor teaches 5 'needed' courses, but allowed to teach only 3 at a time, we have

$$C(5,\ 3) = \frac{5!}{3!\,(5-3)!} = \frac{5!}{3! \bullet 2!} = \frac{120}{6 \bullet 2} = \frac{120}{12} = 10 \textit{ distinct combinations}$$

Once the instructor permutations are complete, a data-file is written containing all instructors, and the courses that are needed by the student-body. At this point, the individual instructors or a department chairperson must now associate time-slots for each of the courses that each instructor needs to teach.

3.2. IApplet – "Instructor applet"

It allows instructors or an administrator to set/modify when instructors teach classes that are required. There is no real algorithm behind this particular applet, per se. It simply allows an instructor administrator to maintain time-slot information.[13]

Each instructor is responsible for maintaining his/her preferences as to when they wish to teach their courses. Unallocated (unspecified) time-slots courses will prohibit valid schedule-generation.

Once all instructors have indicated their preferences, the validation-applet may be run by the administrator.

[13] The "time-slot" are simply textual representations of when courses may be taught. In the current implementation, they are simply 2-hour slots. In a "real" implementation, day-of-week logic should be used. The file **sched.tools.MyComboModel** contains hard-coded time-slots, which can easily be changed.

3.3. AdmValidatorApplet – "Administrator's schedule validation applet"

This applet takes the course permutations[14] and attempts to generate a list of schedules in which all students will be able to take at least one permutation of the schedule.

The applet permutes the instructor-sets[15] and iterates through all student-requirements (again).[16] A viable schedule is one for which at least one student requirement permutations for **EVERY SINGLE STUDENT** exists for the given instructor course time-slot combination [1, 6].

The importance of a viable solution can be illustrated as follows:

If student is taking classes during their senior year, he or she has presumably taken almost all required courses with the exception of their senior-project, and will probably have only electives to take. The number of courses that they can take is limited by the fact that they have taken everything that they actually need.

Conversely, a freshman has taken no courses, and has a great deal of flexibility in what courses they can take. Of course, it is in the best interests of the freshman student to take courses that are prerequisite courses, so as to give them greater flexibility towards which courses they can take in the middle of their degree-program.

Since it is so difficult to determine which courses to offer, a decision was made during the development of this project to use the concept of viability.

So, for the purposes of this project, we are interested in only viable solutions. **AdmValidatorApplet** gives visual feedback as to whether or not all student-requirements are satisfied by each and every instructor

[14] This step assumes that all instructors have filled in the desired time-slots as to when they wish to instruct their classes.

[15] This applet uses the same permutation algorithm as **CourseScheduler**, the only exception is that the applet does it in Java, of course.

[16] Instead of opening and re-reading the original output-files from SKED, it uses cached files, generated during Step 2 of the execution of **CourseScheduler**.

time-slot permutation. The larger the percentage of success, the better the fit of instructor time-slot mappings to the students needs.

Circumstances may exist where none of the schedules can satisfy the student's needs. This[17] situation typically arises when there is a time-slot conflict between the required-courses that must be given in order to satisfy the students who single-schedule needs[18]. These students "by design" must have their needs satisfied, if the goal of graduating these students [in the minimum number of semesters] is to be achieved.

Conflict resolution can be done by having both applets visible[19] and modifying the instructor time-slots in one window, and using the Reload feature of the **AdmValidatorApplet**. The administrator would usually be the one responsible for this task. He or she would use the Validate-button. For those schedules that indicate lack of viability, he/she would click the **Show Conflicts** button, and click "Validate" again. This would enable verbose display of the **AdmValidatorApplet** indicating for each student (within each instructor time-slot permutation) why the conflict occurs. The output within the applet indicates (for a given student) which course cannot be scheduled, given the list of courses that have already been scheduled for the student[20]. At this point, the administrator can (using the **IApplet** page) move a conflicting course from its current time-slot. "Save" the instructor/time-slot information, switch to **AdmValidatorApplet**, use the "Reload" and "Validate" functions.

This step may need to be done several times, in order to afford all students to have a viable schedule.

[17] The "required-course" list is generated by **CourseScheduler**

[18] Again, this is typically the "seniors" who have a much smaller selection of possible courses, since they have taken most or all of their required courses.

[19] This can be accomplished by having a single Web browser open and using the "New Browser" feature.

[20] The order of courses being scheduled for the student depends upon the instructor preferences in the case where the instructor is "required" to teach this course, and this course occurs in more than one permutation of courses that he/she must teach.

Once a viable schedule is found, it can be displayed by selecting the “Show Schedules” checkbox, and once again, clicking the “Validate” button. The resultant output indicates (in alphabetic course-order) which classes are taught by which instructors and in which time-slot to be able to satisfy the student requirements.

4. Software Package

This software package is broken up into three pieces:

CourseScheduler – A process which generates all possible schedules for all instructors, based upon the courses needed by all students. This particular application is written by using C++ [2]. It has been tested and debugged using both GCC (under Linux and Solaris) as well as Microsoft Visual C++ v5 (under Windows NT).

IApplet – An applet which allows instructors to indicate when they would prefer to teach the courses that are needed by the student-body. This piece of software uses the Java Runtime Environment (JRE 1.3) available from Sun Microsystems, and has been tested and debugged on Linux, Solaris, and Windows NT.

AdmValidatorApplet – An applet which allows a department chairperson to view and validate instructors’ selections as to when they teach their courses. This tool gives the chairperson an indication as to how successful, a given schedule is, according to the chosen time-slots of chosen by the instructors, and how well they meet the student’s needs. This piece of software uses the Java Runtime Environment (JRE 1.3) available from Sun Microsystems, and has been tested and debugged on Linux, Solaris, and Windows NT.

The overall approach to finding a solution (“the best” schedules to offer in a given semester, so that the student-body is able to graduate in the least amount of time) occurs in the following steps:

5. Software Execution

Step 1: Find courses required by all students in the system.

duplicate requirements: 's20'.

```
Student   's22'  has 1   combinations : <CPE210 CPE387 MATH109 MATH323 PHYS112>
Student   's26'  has 1   combinations : <AD101 CPE387 CS102 ENGR300 HUMC201 MATH323>
Student   's13'  has 1   combinations : <CPE387 CPE410 CPE447 CS102 EE235 MATH112>
Student   's8'   has 1   combinations : <CS102 ENGR111 MATH112 MATH227 PHYS111>
Student   's1'   has 1   combinations : <AD101 CPE315 HUMC201 MATH323 PHYS111>
Student   's25'  has 1   combinations : <AD101 CPE387 EE235 HUMC201 MATH323 SSCC202>
Student   's4'   has 1   combinations : <AD101 CPE315 CPE387 CPE471 PHYS111>
Student   's21'  has 1   combinations : <CHEM103 CPE315 EE235 HUMC201 MATH323 SSCC201>
Student   's17'  has 1   combinations : <CPE315 CS102 ENGLC101 MATH323 PHYS112>
--------------------------------------------------------------------------------
Student   's3'    has 3     combinations
Student   's14'   has 3     combinations   (reduced to 2)
Student   's2'    has 4     combinations   (reduced to 2)
Student   's6'    has 6     combinations   (reduced to 4)
Student   's16'   has 6     combinations   (reduced to 4)
Student   's23'   has 6     combinations   (reduced to 2)
Student   's5'    has 7     combinations
Student   's7'    has 17    combinations   (reduced to 15)
Student   's15'   has 36    combinations   (reduced to 15)
Need 33312384 combinations (reducible to 604800)
```

Note that Students 22-17 all have only 1 possible course-set. This implies that in order for these students to graduate in the minimum number of semesters, the appropriate courses **MUST** be offered. **CourseScheduler** displays the reduction information. In several cases, no reduction is possible (meaning that the “required-courses” as generated by the students having only a single possible schedule could not be “factored-out”) for several students with multiple possible schedules.

One noteworthy exception is Student s15 whose number of permutations is decreased by more than ½.

Step 2: Determination of instructor availability and coverage. At the University of Bridgeport, faculty members are allowed to teach (at maximum) 3 courses per semester. The number of combinations for an instructor who is capable of teaching **n** but only **r** at a time is given by $C(n, r) = \dfrac{n!}{r!\,(n-r)!}$. So, if we look at Professor Eigel below (who is capable of instructing 5 different courses), we have

$$C(5, 3) = \frac{5!}{3!\,(5-3)!} = 10 \textit{ permutations.}$$

No needed courses for 'rigia' who teaches <CS200>

Instructors (for courses needed by students):

eigel	Edwin Eigel	<MATH109, MATH112, MATH112, MATH227, MATH323>
mahmood	Ausif Mahmood	<CPE387, ENGR111, ENGR111, ENGR300>
abuz	Abdel Abuzneid	<CPE471, CPE473, CS102>
ee-guy	elect-eng-guy	<EE235, EE443, ENGR300>
grodzinsky	Stephen Grodzinsky	<CPE315, CPE448, CPE489>
guerra	Deborah Guerra	<MATH109, MATH112, MATH215>
phys-guy	physics-guy	<CHEM103, PHYS111, PHYS112>
art-guy	artie-the-art-guy	<AD101, CAPS390>
dlyon	Douglas Lyon	<CPE210, CPE387>
engl-guy	english-guy	<ENGL100, ENGLC101>
healey	Stephen Healey	<SSCC201, SSCC202>
human-guy	humanities-guy	<HUMC201, HUMC202>
multi-guy	multi-discipline-guy	<FREELEC1, TELEC1>
romalis	Natalia Romalis	<CPE447, CPE489>
sobh	Tarek Sobh	<CPE315, CPE460>
v_der_kroef	Justus van der Kroef	<SSCC201, SSCC202>
dichter	Julius Dichter	<CS102>
elleithy	Elleithy	<CPE210>
liu	Gonhsin Liu	<CPE410>

In the above list, one should note that certain courses have been removed, as they are not "needed" by the student-body for this semester (this is not to say that they should not/cannot be offered as electives[21]. For example, according to the instructor input-file, *Professor Liu* is needed to instruct **CPE410**. He is also capable of teaching **CPE498 CS536X**, but these courses have been removed since the need of the student-sample does not require either of these 2 classes.

Once **CourseScheduler** knows how which courses are needed, it then find all combinations for every instructor (from the courses that are needed, and the fact that the instructors are only allowed to teach 3 courses). Once all of the combinations are calculated, permutations are generated for all instructor-combinations.

```
eigel          :  10 combinations.
mahmood        :   4 combinations.
abuz           :   1 combinations.
ee-guy         :   1 combinations.
grodzinsky     :   1 combinations.
```

[21] At the discretion of a department-chairperson if he/she feels it would be in the student's best interests.

```
guerra          :  1 combinations.
phys-guy        :  1 combinations.
art-guy         :  1 combinations.
dlyon           :  1 combinations.
engl-guy        :  1 combinations.
healey          :  1 combinations.
human-guy       :  1 combinations.
multi-guy       :  1 combinations.
romalis         :  1 combinations.
sobh            :  1 combinations.
v_der_kroef     :  1 combinations.
dichter         :  1 combinations.
elleithy        :  1 combinations.
liu             :  1 combinations.
total of 40 instructor-combinations.
permute 19 out of 20 instructors.
```

One may notice that *Professor Rigia* has been removed from the list, as she teaches courses that are not required by the students' needs. This does not mean, of course, that she will not be teaching: a department-chairperson may decide to offer courses she teaches as an elective.

Once the minimal-set of courses-required is generated by \CS, instructors or an administrator should begin to fill in the “time-slots” as to when instructors should teach the courses to the students. This is accomplished by using IApplet.

Presumably, the learning-institution would have a web-site exclusively devoted to faculty activities. This would be the ideal place for the IApplet to be placed.

For each faculty-member, they would click on a URL in some kind of “maintenance” page, which would prompt them with a login panel (Figure 1):

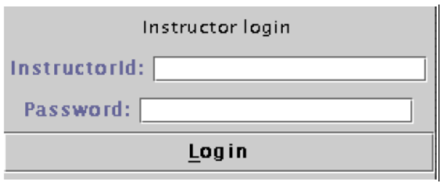
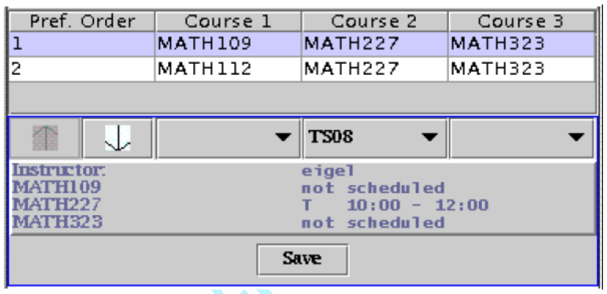


Figure 1. Login-panel during instructor login.

They would fill in their user-name and password, which would then show the classes they would need to teach in the next semester. They would need to select from a list of time-slots as to when they would like to instruct the courses, and click the "Save" button. As an example, Professor Eigel logs on, and maintains his preferences. In Figure 2, one can see that Professor Eigel is needed to teach at least MATH223 and MATH323. We notice that these two courses are common to both combinations of student-required courses. For the courses MATH109 and MATH112. In Figure 2, Professor Eigel has just begun his scheduling. He has selected his first combination, and has elected to instruct MATH227 in TimeSlot 8, which translates to Tuesday morning from 10AM until 12PM[22].



Pref. Order	Course 1	Course 2	Course 3
1	MATH109	MATH227	MATH323
2	MATH112	MATH227	MATH323

TS08

```
Instructor:  eigel
MATH109      not scheduled
MATH227      T    10:00 - 12:00
MATH323      not scheduled
```

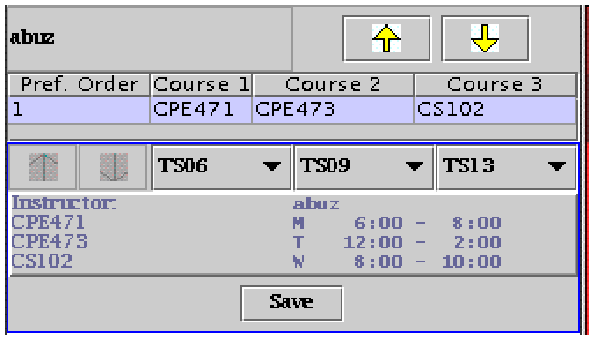
Figure 2. Professor Eigel's preferences.

In order to complete the process, Professor Eigel must continue to select time-slots for the 5 other courses[23]. If he so desires, he can indicate his preference as to which of the two courses he wants to teach (meaning he may prefer to teach MATH112 instead of MATH109 in the next semester). In the case where an instructor has multiple course-sets instructor would select the course-set that they wished to move, and the white arrows would highlight, indicating in which direction the course-set could be moved. In Figure 2, the current row may only be moved downwards. This preference mechanism is used in **AdmValidatorApplet** when determining schedule-viability.

[22] Again, this concept of time-slots is completely arbitrary. Two-hour time-slots have been chosen for this project to simplify processing. In the "real-world" implementation, date-time logic would need to be used to make sure that classes did not overlap.

[23] He has completed the selection for MATH227 in the current combination, the other two are not yet scheduled, and he has not filled in the other three courses for combination \#2.

If **IApplet** is being run by an administrator (typically reached via a protected URL), no login-panel is required, as this version of the applet is not “public”.



abuz

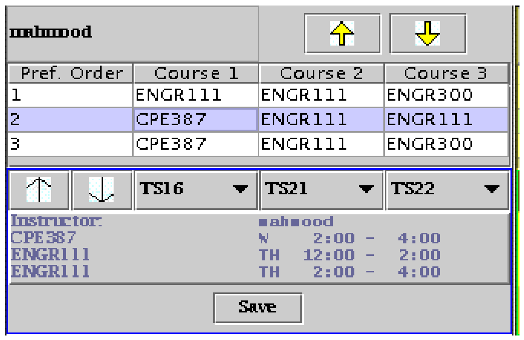
Pref. Order	Course 1	Course 2	Course 3
1	CPE471	CPE473	CS102

TS06 TS09 TS13

```
Instructor:            abuz
CPE471                 M     6:00 -  8:00
CPE473                 T    12:00 -  2:00
CS102                  W     8:00 - 10:00
```

Figure 3. Examining viability of generated schedules.

In Figure 3, we see the administrator viewing Professor Abuzneid’s scheduling-preference. One may notice that in the top-left corner, we see the InstructorID. This is visible only in “Adminitrator-mode”. The two arrows to the right of the InstructorID allow the administrator to step-through all instructors. Notice that Professor Abuzneid has only a single course-set. In Figure 4, Professor Mahmood has three possible combinations. Again, the reason for an instructor having multiple course-sets is that the courses required by the student body and the number of courses that a particular instructor is capable of teaching.



mahmood

Pref. Order	Course 1	Course 2	Course 3
1	ENGR111	ENGR111	ENGR300
2	CPE387	ENGR111	ENGR111
3	CPE387	ENGR111	ENGR300

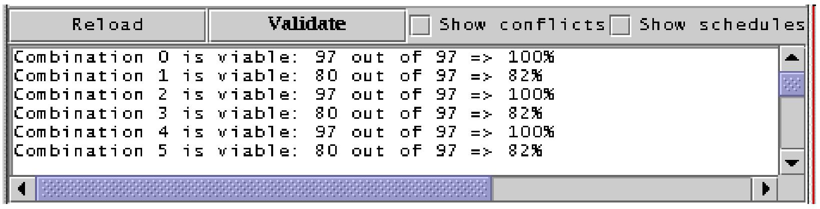
TS16 TS21 TS22

```
Instructor:            mahmood
CPE387                 W     2:00 -  4:00
ENGR111                TH   12:00 -  2:00
ENGR111                TH    2:00 -  4:00
```

Figure 4. Administrator examining instructor with multiple preferences.

Once all instructors have specified their preferences as to when they desire to teach their courses, and which particular course-set is more interesting to them (if applicable)[24], the administrator runs **AdmValidatorApplet**.

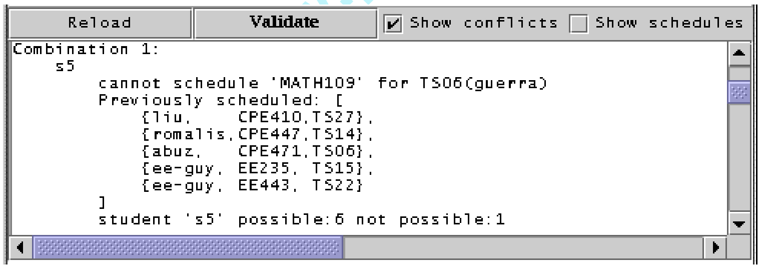
Using **AdmValidatorApplet**, the adminstrator examines the results of the scheduling process. He/she can view reasons for poor viability, and fine-tune the results by having IApplet and **AdmValidatorApplet** both visible.



```
Combination 0 is viable: 97 out of 97 => 100%
Combination 1 is viable: 80 out of 97 => 82%
Combination 2 is viable: 97 out of 97 => 100%
Combination 3 is viable: 80 out of 97 => 82%
Combination 4 is viable: 97 out of 97 => 100%
Combination 5 is viable: 80 out of 97 => 82%
```

Figure 5. Examining viability of generated schedules.

In Figure 5, we see that Instructor combination 0 has 100% viability, while combination 1 has only 82% viability. The administrator may want to know exactly why only 82% of student-requirements are satisfied. In order to view this information, the administrator would simply check the *Show Conflicts* button, and re-click the *Validate* button.



```
Combination 1:
    s5
        cannot schedule 'MATH109' for TS06(guerra)
        Previously scheduled: [
            {liu,    CPE410,TS27},
            {romalis,CPE447,TS14},
            {abuz,   CPE471,TS06},
            {ee-guy, EE235, TS15},
            {ee-guy, EE443, TS22}
        ]
        student 's5' possible:6 not possible:1
```

Figure 6. Examining reasons for poor viability.

Figure 6 shows, verbosely, exactly why we do not have a perfectly viable schedule. We see that student “s5” has 6 valid schedules, but that one of them has a conflict. We see that we cannot schedule MATH109 in TimeSlot #6, since “s5” is already taking CPE471 in this time-slot. Since the student cannot participate in 2 classes at the same time, this schedule is not 100% viable for this student.

[24] this applies only to instructors who have multiple course-sets to manage.

Should the administrator wish to fix this conflict, he/she can move either of the conflicting courses to a different time-slot. **N.B.** one must be careful when doing this, since the success/viability of other students' schedules may depend upon the current time-slot allocations.

6. Limitations and Future Enhancements

6.1. Limitations

Clearly, this system is limited in a couple of ways:

(1) **time-slot** values are currently fixed-value. A much more dynamic solution would be desirable. Specifically, one that handles date/time issues.

(2) **a guaranteed solution** does not always exist. Success of this process is primarily determined by instructor-selection of desired teaching times. A better approach would be to generate the time-slot information based upon the needs of the students (i.e., which classes exist that cannot be scheduled at the same time).

6.2. Future enhancements

Future enhancements are subject to the approval and interest in the results of this project. Some ideas:

CourseScheduler engine – This would contain a set of functions/objects which could be utilized through other programming languages to allow increased flexibility. This would obviate the need for much of the *Java* processing, which is inherently slower than C or C++ [4]. Extensions for COM/DCOM (Microsoft) or RPC (remote-procedure-call - available under most flavors of Unix) are possible, and probably desirable. This would also remove redundant program-code, and provide a single, cohesive toolset for programmers to access in several ways.

time-slots – This is really necessary for this product to function in the real-world. At the very least, the time-slots should be maintainable by the administrator.

time-slot generation – Should really be generated from the student-data. This seems to be the "best" solution, given that individual instructors have insufficient information as to when other "core" courses are being offered, and they may attempt to schedule their own "core" courses at the same time.

7. Conclusions

Using **CourseScheduler**, **IApplet** and **AdmValidatorApplet** functions together as a suite of tools which will help department chairpersons in the course-scheduling process. One of this suite's strong points is that it removes a lot of guess-work from the scheduling process by providing:

(1) immediate feedback and visual cues allowing for quick conflict-resolution.

(2) sampling of the student-requirements, which minimizes the problem of the instructor having to guess as to which courses to offer.

(3) simple and easy-to-understand controls user-interfaces.

While this suite does not address classroom-allocation or class-size issues, we believe that it can be an enormously beneficial set of tools to members of the engineering education community.

References

[1] Thomas A. Cormen et al., Introduction to Algorithms, The MIT Press, Cambridge, Massachusetts, eighth printing, 1992. ISBN: 0-262-03141 (MIT Press), 0-07-013143-0 (McGraw-Hill).

[2] Margaret A. Ellis and Bjarne Stroustrup, The Annotated C++ Reference Manual, Addison-Wesley Publishing Co., New York, 1990.

[3] Donald E. Knuth, The Art of Computer Programming, Volume 3, Sorting and Searching, Second edition, Addison-Wesley Longman, Reading, Massachusetts, 1998.

[4] R. Mihali, T. Sobh and D. Vamoser, SKED: a course scheduling and advising software, J. Comput. Appl. Engg. Edu. 12(1) (2004), 1-19.

[5] Gregory Satir and Doug Brown, C++: The Core Language, O'Reilly and Associates Inc., Cambridge, 1995.

[6] H. Press William et al., Numerical Recipes in C, Cambridge University Press, New York, 1992.

■