CourseScheduler

Tarek M. Sobh & Rik Cousens [1]

Abstract—*A typical problem in an academic environment 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 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 and earn a living.

The job of determining which courses to schedule (usually done by a department chairperson) requires information about 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 they are available to teach a given course. Instructors may have preferences as to what time of day they teach, or on which days of the week they would like to instruct the courses for which they are responsible. An instructor may not be available if they are already teaching too many courses, as defined by the individual learning-institution.

There are several difficulties in determining when to offer a given course. For example, consider a college Senior who has taken almost all of the courses required to satisfy the requirements for his/her degree. Because the student has nearly satisfied the degree-requirement, the student has a limited choice of courses that they can take. This is a problem for the department-chairpersons, since they ***must*** *offer appropriate courses to the student to ensure that they will have enough credits to graduate.*

This document will present a small set of software-components which will aid department-chairpersons in their scheduling process.

Introduction

In an institution of higher-education, 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 learned the information required by either having learned the course-material in an 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].

SKED

SKED evaluates courses already taken by students (transferred from other universities or previously taken at the current institution), and determines the "best" courses that the student can take, given:

1. courses already taken (or transferred) by the student.
2. the *prerequisite* courses for a given course (a prerequisite being a course that must be taken **before** another course, since the prerequisites give the student "foundation" information required to understand the concepts presented in the "next" course).
3. 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 the given-course, but will be helpful to the student, if learned at the same time).
4. 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.
5. and last, but not least, the courses required to satisfy the *major* for which the student desires a degree (a **major** being

1 Dept. of Computer Science, University of Bridgeport, Bridgeport, CT 06601

0-7803-6669-7/01/$10.00 ©2001 IEEE

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*, 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.

The **SKED** algorithm calculates a *requirement-cost* (the maximum number of prerequisite courses that must be taken before a given-course) for each course, as well as an *availability-cost* (which is the number of prerequisite, co-requisites and course-offerings in the next 2 semesters) [1] and generates a data-file 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.**

Trials (and "Errors") – A.K.A Algorithm-refinement

The initial implementation of this project attempted to use:
1. available instructors to teach the classes.
2. classroom size and availability.
3. time-slots of when classrooms were available and instructor availability.
4. students' preferences as to times courses were taught.

This attempt began with faculty ability/availability in generating all possible sets of classes for a given semester, and matching those sets with the student requirements (the dataset is 20 instructors and 19 data-files of student-requirements from **SKED**). It quickly became apparent that this was an *NP-hard* problem (a problem requiring an enormous number of resources and processing time). This version took 2-4 hours to generate a large number of sets that were **not** needed by the student population.

Given the poor performance of this version, it was abandoned without even attempting to address the classroom or student preferences.

[1] see the **SKED** paper, Algorithm section.

The second attempt tried to generate schedules which would maximize the number of:
1. students per class.
2. student-preferences satisfied by the schedule.

This scenario would have been ideal, had it worked out. In this way, we could have maximized the classroom utilization as well as the instructor utilization. The fundamental problem persisted: ***our algorithms didn't sufficiently address the students needs***.

Attempting to cater to the instructors preferences of *when* they wanted to teach courses, or trying to allow the students' desires on *when* they would prefer to take the class, did not work well.

The approach that was finally chosen was the following:
1. treat the student-requirements as a "set".
2. assign instructors to courses that "need-to-be-taught" from the student-requirements.
3. 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.

Algorithms

We found that we needed a special-case for some students: "Seniors" (a student at/near the end of a degree-program) who have satisfied nearly all of their degree-requirements typically have only a single course-set that *must* be followed, in order to graduate in the least amount of time.

It is these students that seem to cause the task of course-scheduling to be so difficult.

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[2].
3. At this point, the system has the following information:
   (a) all courses required by the student-body are possible, since there exist instructors capable of teaching them.

[2] This presents a significant problem should the "unavailable" instructor instruct a course needed by a student with only a single course-set.

0-7803-6669-7/01/$10.00 ©2001 IEEE

(b) 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.

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

(a) *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).

(b) *default.courses* – an "index" file containing the ordinal number of the courses actually **needed** by the student-sample.

(c) *student.XXX.map* – generated for each student containing all possible required-schedules for the student. A file is **not** generated if one student has the *same exact* requirements as another student in the system.

Each of the required-schedules has been "reduced", or "factored". Each line of the file contains the ordinal "index-values" into *course.map*.

The **CourseScheduler** application does its processing in 4 steps:

1. **CourseScheduler** makes a special-case for students with only a single course-set. For those 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), a list of "required-courses" is built. These "required-courses" must be taught. **CourseScheduler** then removes the "required-courses" from all other students. Then, a unique list of distinct requirements are made from the course-sets left after the "factoring". This is done so as to reduce the number of permutations that must be generated in Step 2.

2. At this point, **CourseScheduler** iterates through all students with multiple course-sets, generating a list 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). 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.

3. **CourseScheduler** examines the "required-course" set and the new minimal set of courses required by all students and verifies that there are available/qualified instructors to teach each course, or displays an error message. If an instructor teaches no courses that are "required" by the student-sample, **CourseScheduler** displays this fact as well.

4. **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 (instructors in the same department can teach the same course). For example, there is usually more than one instructor within the **Math** department that is capable of instructing *CS 227* (Discrete Math) (a prerequisite course at University of Bridgeport for many courses). Depending upon the availability of the instructor, and student-requirements, the course may need to be split into multiple sections. Instructors are limited to a certain number of courses that they can teach each semester (so instructors have 'free" time to prepare exams, attend faculty meetings, etc.).

The University of Bridgeport 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:

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

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) = $\frac{5!}{3!\bullet(5-3)!} \Rightarrow \frac{5!}{3!\bullet 2!} \Rightarrow \frac{120}{6\bullet 2} \Rightarrow \frac{120}{12} \Rightarrow 10$ 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, individual instructors or the "adminstrator" must allocate a time-slot to each course required by the students' needs.

IApplet – "Instructor applet"

IApplet allows instructors or an "administrator" to set/modify *when* instructors teach the classes that are required. There is no algorithm behind this applet, per se. It simply allows an instructor/administrator to maintain time-slot information (textual representation of *when* courses are 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.

Each instructor is responsible for maintaining their 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.

AdmValidatorApplet – "Administrator's schedule validation applet"

This applet takes the course permutations (assuming all instructors have completed their time-slot/course selection) and attempts to generate a list of schedules in which all students will be able to take at least one permutation of their schedule.

The applet permutes the instructor-sets[3] and iterates through all student-requirements (by re-reading the **SKED** output files and cache-files from Step 2 of CS.). A **viable** schedule is one for which *at least* one student/requirement permutation for **EVERY SINGLE STUDENT** exists for the given instructor/course/time-slot combination.

So, for the purposes of this project, we're interested in only *viable* solutions. **AdmValidatorApplet** gives visual feedback as to whether or not all student-requirements are satisfied by each and every instructor/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 students' needs. This situation typically arises when there is a time-slot conflict between the "required-courses" (from **CourseScheduler**) that must be given in order to satisfy the students' single-schedule needs. 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.

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

Software package

This software package is broken up into three pieces:

1. **CourseScheduler** – a process which generates all possible schedules for all instructors, based upon the courses needed by all students. This particular application is written in C++. It has been tested and debugged using both **GCC** (under *Linux* and *Solaris*), as well as **Microsoft Visual C++ v5.0** (under *Windows NT*).
2. **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 (JRE1.3) available from Sun Microsystems, and has been tested and debugged on *Linux*,*Solaris*, and *Windows NT*.
3. **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 time-slots chosen by the instructor, and how well they meet the students' needs. This piece of software uses the Java Runtime Environment (JRE1.3) 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:

Software execution

The following snapshots show:

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 from 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 $\frac{1}{2}$.

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** courses but only **r** at a time is given by $C(n,r)=\frac{n!}{r!(n-r)!}$. So, if we look at Professor Eigel below (who is capable of instructing 5 different courses), we have

$C(5,3) = \frac{5!}{3!(5-3)!} \Rightarrow \frac{5!}{3!\bullet 2!} \Rightarrow \frac{120}{6\bullet 2} \Rightarrow \frac{120}{12} \Rightarrow = 10$ 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/can not be offered as electives. For example, according to the instructor input-file, *Professor Liu* is needed to instruct **CPE410**. He is also capable of teaching **CPE498**

[3]This applet uses the same permutation algorithm as **CourseScheduler**: the only exception is that the applet does it in **Java**.

and **CS536X**, but these courses have been removed since the needs of the student-sample does not require either of these 2 classes.

Once **CourseScheduler** knows which courses are needed, it then finds 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.
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.

Once the minimal-set of required-courses is generated by **CourseScheduler**, 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.

Each faculty member would use a web-browser to maintain their preferences (by clicking a hypertext-link to **IApplet** on a web-page exclusively for faculty activities.). A login-panel

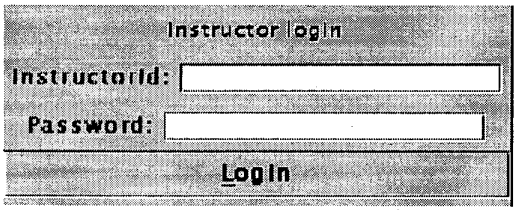

Fig. 1. Login-panel during instructor login.

appears (Figure 1) and the staff-member would fill in their user-id and password, and begin the process of choosing when (during the day) they would like to instruct their classes. After all courses (in all permutations) have a time-slot associated with them, the instructor would click the "Save" button.

In Figure 2, Professor Eigel has 2 permutations of classes. He must teach MATH227 and MATH323. Time-slots must also be allocated for both MATH109 and MATH112. Also in Figure 2, we see that Professor Eigel has begun his election-process. MATH227 has been allocated Time-Slot 8, which translates to 10AM-12PM Tuesday morning.

In order to complete the process, Professor Eigel must continue to select time-slots for the 5 other courses. If he so

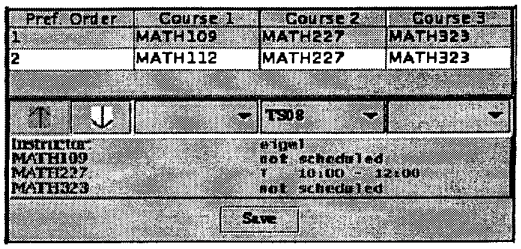

Fig. 2. Professor Eigel's preferences.

desires, he can indicate his preference as to which of the two course-sets he wants to teach (meaning he may prefer to teach MATH112 instead of MATH109 next semester). He would use white arrows (between the course-permutations and "translation" section) to move course-sets up or down. The higher the course-set, the more the instructor prefers the given course-set. In Figure 2, the current row may only be moved downwards. This preference mechanism is used in **AdmValidatorApplet** when determining schedule-viability.

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".

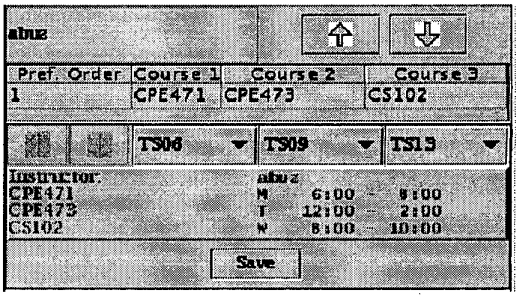

Fig. 3. Administrator viewing instructor with a single course-set.

In Figure 3, we see the administrator viewing Professor Abuzneid's scheduling-preference. We know that this is the administrator due to the presence of the Instructor-ID and two yellow arrows, top-right. The arrows allow iterating through the instructors in order to view/change their course/time-slot information. In Figure 4, Professor Mahmood has three possible combinations. The more able [4] the professor, the more course-sets need to be filled-in.

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) [5], the administrator runs **AdmValidatorApplet**.

Using **AdmValidatorApplet**, the administrator examines the results of the scheduling process. He/she can view reasons

[4] the instructor is capable of instructing many courses, and the courses are needed by the student-population.

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

for poor viability, and fine-tune the results by having **IApplet** and **AdmValidatorApplet** both visible.

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. To view this, the administrator checks the *Show Conflicts* button, then re-clicks *Validate*.

Figure 6 shows (verbosely) how Student "S5" had 6 out of 7 possible schedules satisfied. We see that MATH109 could not be scheduled since the student also needs CPE471; both classes are being offered in Time-Slot 8. Since the student cannot participate in 2 classes at the same time, this schedule is not 100% viable for this student.

Should the administrator wish to fix this conflict, they 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.

Limitations and Future enhancements

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).

Future Enhancements

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

1. **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++. 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 tool-set for programmers to access in several ways.
2. **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.
3. **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.

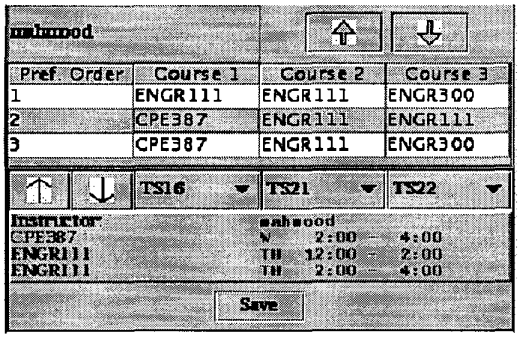

Fig. 4. Administrator examining instructor with multiple preferences.

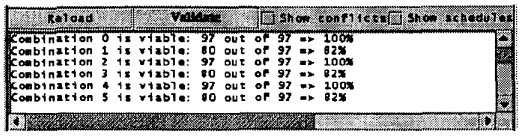

Fig. 5. Examining viability of generated schedules.

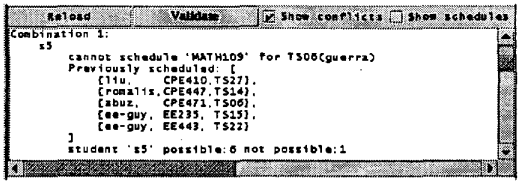

Fig. 6. Examining reasons for poor viability.

Conclusions

Using **CourseScheduler**, **IApplet**, and **AdmValidatorApplet** together as a suite of tools 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 educational community.

References

[1] Raul Mihali et al. *SKED: A Course Scheduling and Advising Software.*
[2] Ellis, Margaret A. and Stroustrup, Bjarne. *The Annotated C++ Reference Manual.* Addison-Wesley Publishing Co., New York, 1990. ISBN: 0-201-51459-1

[3] Knuth, Donald E. *The Art of Computer Programming: Volume 3, Sorting and Searching.*,Second edition., Addison Wesley Longman, Reading, Massachusettes, 1998. ISBN: 0-201-189685-0.
[4] Satir, Gregory and Brown, Doug. *C++: The Core Language.* O'Reilly & Associates, Inc., Cambridge, 1995. ISBN: 1-56592-166-X.
[5] William H. Press et al. *Numerical Recipes in C*, Cambridge University Press, New York, New York, 1992. ISBN: 0-521-43108-5.
[6] Thomas A. Cormen et al. *Introduction to Algorithms*, The MIT Press, Cambridge, Massachusettes, eighth printing, 1992. ISBN: 0-262-03141-8 (MIT Press), 0-07-013143-0 (McGraw-Hill).