A Campus-wide University Examination Timetabling Application

Andrew Lim, Ang Juay Chin, Ho Wee Kit, Oon Wee Chong

School of Computing,
National University of Singapore, Singapore
{alim, angjc, howeekit, oonwc}@comp.nus.edu.sg

Abstract

The authors of this paper were tasked to create an automated campus-wide timetabling system, for both course and examination timetable scheduling, for the National University of Singapore. This paper explains the development and design of the exam-scheduling portion of the University Timetable Scheduler (UTTS) software. The preliminary results of the application of the AC3 algorithm on this problem are also shown, and indicate the tremendous potential benefits of such a system.

Introduction

Beginning in the 1993/1994 academic year, the National University of Singapore (NUS) introduced a modular academic course structure to give students greater control over the content of their course of study. This new structure has overall been a welcome change, but the students have largely been restricted to choosing courses within their own faculty. NUS is now following up by introducing cross-faculty modules, which are subjects that can be taken by students from various faculties. It is the intention of NUS to eventually offer program comprising of up to 30% cross-faculty modules. More information on the National University of Singapore and its course structure can be found at the NUS Website[1].

The introduction of cross-faculty modules greatly increases the difficulty in scheduling both the course and examination timetables. In particular, examination timetable scheduling (handled by the administration department for the entire university) is made much more difficult as cross-faculty modules must be placed in an available timeslot for students from several faculties. In view of this, NUS has tasked the authors of this paper to create an automated course and examination timetable scheduler, with the working title of University Timetable Scheduler (UTTS).

[1] http://www.nus.edu.sg

Aspects of the course-scheduling portion of UTTS are described in a separate paper (Lim et al. 2000a). This paper describes the development of the exam-scheduling portion. We will first give an account the way examination timetabling is done currently in NUS. We then describe the system design of UTTS, bearing in mind the possible conversion to a Client/Server application in the future. Finally, we will describe the current status of our program, with the results and statistics of our preliminary testing.

The Current System

At present, the University's policy is to schedule all examinations before student enrolment. Hence, it is the duty of all students to make sure that they choose courses with examinations that do not clash. Obviously, this is undesirable as it unnecessarily restricts the students' choices. It is therefore an aim of any timetable to have as few such potential clashes as possible.

The University's examination timetable scheduling is currently handled by the Administration department, which must organize and schedule all the examinations in a particular semester for each and every faculty. We interviewed the timetabling officer from the Administration department to find out their current timetabling process.

At the moment, this difficult process is still being done by hand. The process of creating the resultant timetable is as follows:

1. Each faculty puts forward a request for a certain number of days, timeslots and seats to the Administration Department.

2. The timetabling officer assigns each faculty a certain number of days, timeslots and seats. The number for each faculty is based on a combination of the requested number, enrolment figures, availability of resources and previous experience. History has shown that there are never enough resources to accommodate the wishes of all the faculties. The timetabling officer also reserves a number of "spare" slots for emergencies.

3. Each faculty then attempts to create a feasible examination timetable using the resources they have been assigned. Some faculties further break down these resources to departmental level, and produce a collated timetable of all the departments.

4. Inevitably, some faculties will find that the resources allocated to them are insufficient to create a feasible timetable. They then contact the Administration department to request for more resources. Depending on need and availability, the timetabling officer would then allocate more resources to the requesting faculty from the pool of "spare" slots.

5. This cycle of request/allocation continues until all the examinations have been scheduled satisfactorily.

In practice, this procedure has many disadvantages:

1. The initial allocation of days and slots is likely to be sub-optimal. This is because it is difficult to judge the effect of changes in student enrolment and registration.

2. Human nature and expediency dictates that the initial request for resources would be a value somewhat greater than what is strictly required. This is understandable, as the individual faculties would like some room to maneuver in case of unforeseen emergencies.

3. Due to the above points, the process is extremely time-consuming. The cycles of request and allocation (coupled with episodes of negotiation an compromise) can take several weeks to resolve.

4. The process is also error-prone, due to the large amount of data to consider. The act of verification is difficult, and there has been a case of a conflict that was overlooked until a very late stage, and its correction was awkward and troublesome.

The automated examination timetable scheduler aims to eliminate these problems. In particular, the UTTS Exam Scheduler would take as its most important inputs the set of examinations and candidates, and the set of constraints from each faculty. Hence the adequacy of resources could be more accurately determined, along with the task of their allocation. Furthermore, if the system proves successful, we can then experiment with scheduling the examinations only after student registration.

Functions of the Automated System

When we started work on the UTTS Examination Scheduler, we strove to achieve the following aims:

- To create an examination timetable that schedules all examinations, invigilators, registrar staff and required equipment. The most important criterion is that two examinations taken by the same student should not be scheduled at the same time. The preferences of invigilators and registrar staff are relatively less important than the ability of a candidate to sit for his registered examination. Equipment required for an examination can be treated simply as an attribute for the examination, and should have little or no effect on the actual timetable scheduling.

- To drastically reduce the time taken to schedule the examination timetable, while at the same time satisfying (as much as possible) the constraints imposed by the user. Aside from the obvious benefits of saved time, an automated timetable scheduler that can produce a timetable in minutes rather than days or weeks opens up possibilities of simulating policy changes.

- To minimize the total number of days taken for the entire examination period. The total number of days occupied is used to judge the "goodness" of a timetable solution.

- To reduce the number of examinations held at the IMM Building, which would result in a definite monetary saving (see the Problem Domain section below).

- To produce a candidates' seating plan. This is the least important aim, as it is just a tool of convenience for the users. To the scheduling engine, where a particular student sits within an examination venue is irrelevant.

Problem Domain

In order to test our eventual system, we obtained student registration data from the NUS Computer Center for both semesters of the academic years 1997/1998 and 1998/1999, as well as for the first semester of academic year 1999/2000. The data that was obtained from the Computer Center consisted of a set of text files, each containing the list of student-examination tuples. We converted the text files into *Microsoft Access*™ files, which enabled us to make use of SQL queries for the data.

As an example of our problem domain, we present the

statistics for Semester I of the 1998/1999 academic year. This consists of 21607 students, each taking one or more of 1561 papers. Some relevant facts include:

- The number of candidates sitting for an examination ranges from as few as 1 (152 papers) and as many as 1283 (Mathematics A, EG1401).
- Each examination requires a number of invigilators to be present, of which at least one must be a Chief Invigilator.
- Most examinations also require one or more members of the Registrar's Office to be present.
- Some of the examinations are labeled "open-book" examinations, meaning that the candidates are allowed to bring reference materials into the examination venue. The remaining are considered "closed-book", where no reference materials are allowed.

These examinations are scheduled to the following venues:

Alias	Venue	Capacity
IMM	IMM Exhibition Hall	1600
GYM	Gymnasium	312
MPH1	Multi-Purpose Sports Hall 1	750
MPH2	Multi-Purpose Sports Hall 2	850
CH	Competition Hall	396

Table 1: List of Examination Venues

The IMM Building is a commercial building that is not owned by NUS. The venues in the IMM Building must be rented by the University for the purpose of holding the examinations.

Using the current manual system, the timetable that was produced started from 27 October 1998 to 28 November 1999, comprising 47 sessions.

System Design

The UTTS system design is based on the 3-Tier architecture that is commonly used when building Client/Server applications. It keeps distinct the GUI, object oriented and data storage portions of our program. By separating the system into 3 tiers, they can be worked on independently (Reese, 1997).

UTTS is divided into the following 3 tiers. The *View* tier involves the graphical user interface. The *Application* tier is composed of the modules in an object-oriented paradigm that manipulate the objects in the system. This includes the scheduling engine, the printing modules and the report generator. Finally, the *Persistence* layer consists of the actual database access. Figure 1 shows the UTTS system design.

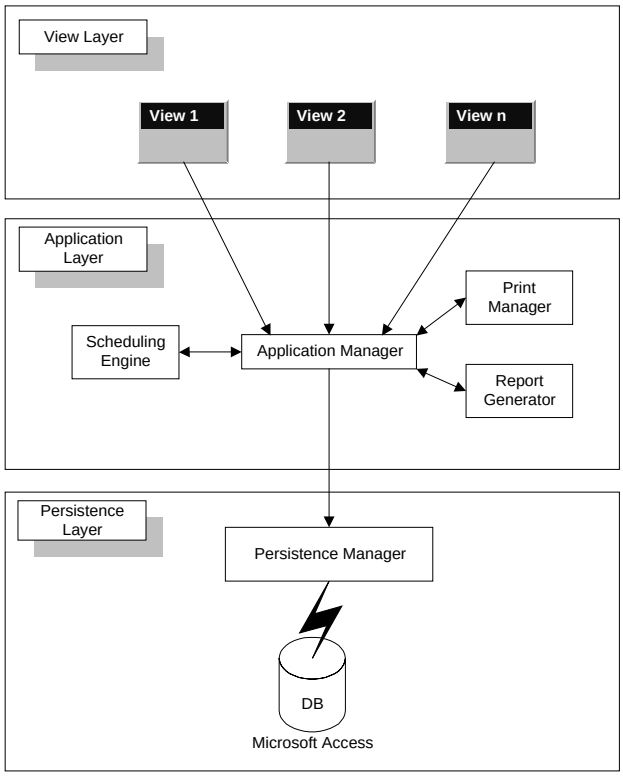

Figure 1: UTTS System Architecture

When deciding on our system design, we had to balance the factors of program speed and memory use. One naïve implementation would be to load all data into the main memory during program load time. However, this would take up an unnecessary amount of memory, since it is unlikely that all the information stored in the databases would be required. The starting load time would also increase. In our design, we read information into memory on an "as-needed" basis. We keep a `MasterList` in the *persistent* layer that retains the list of objects read from the database, and the actions performed on them. This `MasterList` is also useful for undoing actions. For example, when the user requests for information on a particular student, the information flow is as follows:

1. The GUI requests for the student information from the *application server* layer by calling `server.getStudent(studentIndex).`
2. The *application server* layer uses the *persistence manager* class to retrieve this information, calling `persistent.get(studentIndex).`

3. The *persistence manager* class first checks if the wanted student can be found in the `MasterList`. If it can, the correct student object is returned straight from the `MasterList`. If not, it asks the *student* class to load the required student object.
4. The *student* class delegates its *studentPeer* class to retrieve the appropriate information from the database. The student object is then returned to the GUI, and that student object is updated to the `MasterList`.

In this way, both database access and memory usage are minimized.

Scheduling Engine

The Examination Timetabling Problem is both a constraint satisfaction problem (CSP) as well as an optimization problem. In the exam-timetabling problem, we are typically given a set of both hard and soft constraints. Two *hard* constraints that must be satisfied are:

- No student is required to be present for two or more examinations in the same time slot.
- There number of seats at a venue is sufficient to accommodate all the students scheduled to take an examination there.

In addition, we would also like to handle the following constraints if possible (*soft* constraints):

- *Staff S* is required to invigilate at least *x* sessions and at most *y* sessions.
- Separate all open and closed-book examinations.
- Separate all examinations with different duration.
- Spread all examinations of a student over the examination period as much as possible.
- Any 2 papers of a student should be placed minimally *x* sessions or *y* days apart.
- *Paper A* be placed *x* days away from *Paper B*
- *Paper A* to be held before *Paper B*
- *Paper A* and *Paper B* to be held simultaneously
- *Paper A* and *Paper B* are not to be held simultaneously
- *Paper A* to be held as early/late as possible in the examination period
- *Paper A* is to be held in session *s*
- *Paper A* is to be held on/before/after date *d*
- *Paper A* is to be held within period *(d1, d2)*
- *Paper A* must not be held during period *(d1, d2)*
- *Paper A* is to be held in week *n*.
- *Paper A* is to be held at venue *v*.
- *Paper A* is to be held at a venue belonging to venue group *g*.

The user can assert any or all of the above soft constraints. Our objective in any case is to generate a conflict-free timetable, which minimizes the number of time slots used. In consideration of the nature of our problem and the variety of constraints that we have to handle, we have combined our main scheduling algorithm together with a consistency algorithm.

For our first attempt, the main scheduling algorithm used is based on a weighted sum of three measures. Each paper has a weight computed as follows:

- Measure 1 is based on the number of candidates taking this paper
- Measure 2 is based on the constraint degree of this paper (i.e., the number of other papers affected by the scheduling of this paper)
- Measure 3 is based on the number of slots that cannot be used for scheduling this paper, due to one or more constraint conflicts

$PaperWeight = \alpha Measure1 + \beta Measure2 + \gamma Measure3$ where $\alpha + \beta + \gamma = 1$

With this weighted scheme, the main algorithm can be described as follows:

1. Let Q be a Priority Queue of papers sorted by *PaperWeight*
2. While Q is not empty
   Dequeue paper p in Q
   Find the first available time slot for paper p
   If the time slot is found then
   Assign paper p to the found time slot
   Else
   Return Failure
3. Return Success

We model our problem as a CSP and derive a constraint graph. Each paper corresponds to a vertex of the constraint graph with its associated domain. Each constraint relation P(X, Y) corresponds to the two arcs (X, Y) and (Y, X) in the graph.

We apply the arc consistency algorithm, AC-3 (Mackworth 1977), to perform domain reduction at the start of the scheduling algorithm, as well as during each assignment of a paper to a time slot. The AC-3 algorithm basically makes the entire graph arc-consistent by considering a set of potentially inconsistent arcs. While the queue is not empty, an arc is removed from the queue and considered. If it is not consistent, its domain is revised and made consistent. As a result, all other consistent arcs that could have become inconsistent are inserted back into the queue.

Exam Data Information	1998/99 semester 1			1998/99 semester 2		
No. of Candidates	21607			21591		
No. of Candidates_Paper	101197			93693		
No. of Time Conflicts	23751			24424		
Max Degree of Time Conflicts	277			285		
Connectivity of Constraint Graph	1.95%			2.05%		
Comparison	**Papers**	**# Slots**	**Time(s)**	**Papers**	**# Slots**	**Time(s)**
Manual System	1282	47	-	1248	48	-
UTTS (M1=0.1, M2=0.8, M3=0.1)	1561	30	381	1545	29	368

Table 2: UTTS test results

One advantage of performing domain reduction using AC-3 is that we can handle the various constraints easily. In addition, with the use of AC-3, our main scheduling algorithm requires a minimum effort in finding the first available slot for each paper. We could also determine the number of slots that are being eliminated for a given paper (as used in Measure 3) easily.

When using AC-3, we need to perform an arc-consistency check whenever a paper is assigned a slot. However, this overhead is relatively cheap compared to naïve implementation of computing the availability of the all time slots for every unscheduled paper. Thus, making use of AC-3 helps to improve the efficiency of our scheduling algorithm.

Test Results

The data we acquired from the computer center contained some anomalies of students taking an illogical number of examinations. In particular, there were instances of students taking more than 20 papers in the semester. We suspect that this is due to the inclusion of non-examinable and/or exempted papers in the database. As there was no convenient way to remove these cases, they remained in our test data.

Another discrepancy between our test data is the number of examinations to be scheduled. Our test data includes non-examinable subjects, which as previously stated cannot be conveniently extracted. Furthermore, some examinations in the manual timetable were scheduled in small classrooms and laboratories, and these alternative sites are not used in our simulation. Hence, our results are based on more papers, to be scheduled in fewer venues, and may be worse than in an actual implementation.

Table 2 shows the results of running the UTTS program on our sets of test data, and also compares the actual timetables that were created by the manual process and the timetables generated by our program. In these results, we have only taken into account the 2 hard constraints.

As can be seen, despite having to schedule more papers in a smaller capacity, our heuristic produced much better results than the actual manual system. For Semester I of the 1998/1999 academic year, UTTS produced a timetable that makes use of a mere 30 slots, compared to 47 for the manual system. Similarly, 29 slots were used compared to 48 for Semester II of the same academic year. These results were obtained in around 6 minutes per test case.

Since our tests only take into account the 2 hard constraints, we cannot view these results as ironclad. Nonetheless, it is obvious that our automated system can potentially result in a timetable with a shorter duration than the traditional manual system.

Our tests were performed on a Pentium-500 PC with 128MB RAM. The system was coded in JDK 1.1 and JFC 1.03 using *IBM VisualAge™ for Java*.

Future Directions

The development of an automated examination-timetabling program is a large project, and there will be several cycles of development, testing, user feedback and implementation to be done before the final product is deployed. The major aspects include the handling of more constraints, the advancement to the Client/Server architecture, and better scheduling algorithms. Efforts have already been made in the direction of improved scheduling algorithms by the UTTS team (Lim and Fu, 2000).

UTTS is an ongoing project, with improvements and refinements to be made as the program undergoes actual use.

Conclusion

This paper details the development process and techniques of an automated examination-timetabling program. We show our system design based on the 3-tier architecture, which is appropriate for our purposes. We also show the results of implementing the AC-3 algorithm on our constraint-propagation scheduling engine, and note that the results are at least comparable to those achieved by the current manual system, and in a much shorter time, when implemented on the basic hard constraint set.

We believe that our system shows the great potential benefits of automating the examination-timetabling task in a large university like NUS.

References

The National University of Singapore Website, http://www.nus.edu.sg

A. Lim, W. C. Oon, J. C. Ang, W. K Ho, *Development of a Campus-wide University Course Timetabling Application: Input Issues*, forthcoming.

A. Lim, Z. Fu, *The Examination Scheduling Problem*, forthcoming.

A. K. Mackworth, Consistency in Networks of Relations, *Artificial Intelligence* 8 (1977): 88-119

G. Reese, *Database Programming with JDBC and Java*, O'Reilly 1997.