LARGE SCALE CLASSROOM SCHEDULING

Edward L. Mooney
Industrial and Management Engineering Department
Montana State University
Bozeman, Montana 59717-0384

Ronald L. Rardin
School of Industrial Engineering
Purdue University
West Lafayette, Indiana 47907

W.J. Parmenter
Space Management and Academic Scheduling
Purdue University
West Lafayette, Indiana 47907

June 26, 1995

Contents

1 Introduction **1**

2 Course Scheduling at Purdue **2**

3 The Classroom Scheduling Problem **5**

4 Schedule Generation **8**

4.1 Search Model 8

4.2 Preprocessing 10

4.3 Local Search 11

4.3.1 Dynamic Biased Sampling 12

4.3.2 Strategic Oscillation 12

5 CHRONOS Implementation **14**

5.1 Data Input 14

5.2 Application Tests 15

5.3 Test Results 16

5.4 Large Lecture Rooms Experience 17

6 Conclusions **19**

Abstract

Classroom scheduling is an important part of course scheduling at Purdue University. The objective is to choose meeting rooms and times for each class that maximize student and instructor preferences without creating student, room or instructor schedule conflicts. An approach for solving classroom scheduling problems of practical size has been developed and implemented in CHRONOS, a scheduling support system developed at Purdue and described in this paper.

Requirements for CHRONOS derive from the course scheduling process at Purdue and are specified in a mathematical model of the classroom scheduling problem. Database, preprocessing, and search components provide computerized support to decision makers. Results obtained from preliminary tests and ongoing use scheduling 500 course sections in a set of 31 large lecture rooms are positive. Work is currently under way to implement the system in a client-server environment and improve the qualitative aspects of generated schedules.

1 Introduction

Scheduling is the "allocation of resources over time to perform a collection of tasks" [2]. Course scheduling is a familiar and difficult scheduling problem where the "tasks" to be scheduled are meetings. Resources include instructors, classrooms, and groups of students, or classes. In the broadest sense, course scheduling includes many related problems such as exam, conference, university, and school scheduling.

Effective course scheduling maximizes the likelihood that students can get desired courses while considering other goals and constraints. To date, the most successful computerized methods have been developed for special cases such as exam scheduling and timetabling (meeting time assignment subject to various constraints). Resource assignment (instructor, classroom or student section assignment) subproblems have also received significant attention. While progress has been made, solving large instances is difficult. Consequently, course scheduling problems requiring multiple resource and meeting time selection are usually solved by iterating between timetabling and single resource assignment subproblems.

For example, Ferland and Roy [6] and Aubin and Ferland [1] describe iterative room-time and class-time assignment systems. Hertz [10] describes tabu search algorithms for timetabling and student-section assignment (grouping) problems which presumably would be applied iteratively, given instructor and room assignments. Carter documents an iterative approach used at the University of Waterloo to develop course schedules [4]. A two phased approach is proposed by Tripathy [15] in which courses and rooms are grouped in the first phase and meeting times are selected in the second phase. Lagrangian relaxation is used to solve a smaller problem obtained by grouping.

Other course scheduling strategies have relied on interactive human aid. Mulvey [13] develops a multiple assignment scheduling approach based on the use of a general network model. The problem is modeled as the assignment of instructor-student *classes* to classroom-time *slots*. The approach relies on an interactive human interface designed to limit the size of the problem encoded for solution. Dinkel, Mote and Venkataramanan [5] report an application based on Mulvey's model involving 300 sections, 20 rooms, and 16 time slots. Solution was accomplished using a commercial integer programming code. A similar approach has recently been described by White and Wong [16] which allows incremental interactive time and room assignment using heuristic solution methods. Acceptable schedules were reported for department level use.

Nordlund [14] solved a special case of university classroom-time assignment using an enumerative, branch and bound scheme based on Bean's work [3]. While the approach worked well on some problems, difficulties finding feasible solutions for moderately constrained problems were observed. These results were consistent with Bean's experience using the approach for a different class of problems.

The approach developed at Purdue and reported here addresses classroom *scheduling*. Classroom scheduling is an example of single resource scheduling where classrooms and meeting times are selected concurrently for each course section, subject to classroom and other resource constraints. While any resource may be scheduled using the approach, classrooms were chosen because, with 75% average utilization at Purdue, they represent the course scheduling bottleneck.

2 Course Scheduling at Purdue

Each fall approximately 35,000 students pursuing degrees in 40 curricula enroll in over 3,000 courses offered by 79 academic departments at Purdue University. These courses are taught in roughly 8,000 sections by approximately 2,400 faculty members. Sections meet in classes of from 1 to 500 students, typically for 3 hours of lecture per week in one of 268 classrooms. A schedule that allows students to complete curricular requirements in a timely manner must be developed each semester.

Course schedules are developed at Purdue using a decentralized process. Academic departments schedule most of their course offerings in classrooms allocated to them. Each department uses its own criteria to assign instructors to sections and sections to rooms and times. Departments that consider instructor preferences subordinate those preferences to curriculum requirements and room capacity and other constraints.

Room allocation and interdepartmental coordination is provided by a centralized Space Management and Academic Scheduling (SMAS) Department reporting to the Vice President for Academic Affairs. SMAS also has responsibility for assigning individual students to course sections following schedule development. This student scheduling approach contrasts with the first-come-first-served methods in use at many institutions. Finally, SMAS directly schedules sections requiring 31 large lecture rooms and other sections that are difficult for individual departments to schedule.

A schedule is built each semester for the following 16 week semester. Table 1 shows course scheduling activities, when they are performed in the scheduling cycle, and the Purdue organization responsible. The first step, course demand and room requirements determination, must be completed before the semester in which the schedule is built begins. Student scheduling is not completed until the first week in the following semester. In the interim, the course master schedule is continually refined as better course demand information becomes available.

Table 1: Purdue Course Scheduling Process

Activity	Organization	When
Determine course requirements	Departments	Prior semester
Allocate rooms to departments	SMAS	Prior semester
Request large lecture rooms	Departments	Week 1 of semester
Schedule common (large rooms)	SMAS	Weeks 2 through 5
Build master course schedule	Depts., SMAS	Weeks 6 through 7
Pre-register students for next semester	Registrar	Weeks 8-15
Assign students to sections	SMAS	Wks. 8-16, 1st wk. next semester

SMAS allocates rooms to departments to ensure that each department has enough capacity to schedule their courses while providing for satisfaction of institutional scheduling goals. This approach has contributed to classroom use that averages 37.7 hours per week, 75% of available hours, with a 67% station occupancy rate. Furthermore, departments must spread schedules throughout the day and week, facilitating student scheduling and increasing the chances for students to get requested courses. Currently 65% to 70% of students entering Purdue will earn a 4-year degree in 6 years or less. Statistics also show that 75% of the 1986 freshmen engineering students graduated within 8 semesters.

The scheduling process at Purdue has worked well. Decentralization means that departments can respond to changes in demand, instructor availability and other factors while coordination by SMAS provides a global perspective. Historically, the long planning cycle resulted in preliminary schedules based on incomplete information, and the difficulty of manual revision limited the number of schedule improvement iterations possible. As a result, the process terminated with schedules that could be improved given better information and

faster scheduling methods.

Recent process automation initiatives have addressed information and time constraints by shortening the planning cycle and making it possible to refine elements of the schedule more quickly. Student scheduling was the first process to be automated and has resulted in significant improvements in student schedules. This capability has also resulted in improved class schedules by allowing "what-if" questions regarding the impact of course schedule changes on student schedules. With automated student scheduling in place, classroom scheduling was identified as the best opportunity for improving course schedules. With computer assistance, scheduling can start later, with better information, and schedules can be generated and evaluated more quickly.

3 The Classroom Scheduling Problem

Classroom scheduling results in feasible classroom and meeting time assignments for each section of each course. At Purdue, good schedules are those that maximize the likelihood that students are able to schedule selected courses. Secondary goals include maximizing facility utilization and instructor preferences. Feasible schedules must not assign instructors or classrooms to more than one meeting simultaneously or to times when they are unavailable. Also, rooms may not have sections assigned which exceed their seating capacity.

Relations between sets of time patterns, classrooms, and sections are made explicit with a nonlinear 0-1 integer programming model of the classroom scheduling problem (CSP). A time pattern is specified by a set of days, a start time, and a total meeting time (per week) from which the number of meetings and the length of each can be deduced. Two patterns *overlap* if they have at least one day in common and one or more time blocks that overlap. Classrooms, time patterns, and sections are identified by integer-valued indexes. Now, given the following sets, subsets and notational conventions

$$\begin{aligned}
\mathcal{R} &= \{ \text{ classrooms, } r \in \mathcal{R}\} \\
\mathcal{T} &= \{ \text{ time patterns, } t \in \mathcal{T}\} \\
\mathcal{L} &= \{ \text{ slots, } l = (r,t) \in \mathcal{L}\} \\
\mathcal{L}_s &= \{ \text{ slots with feasible classroom and time pattern for section } s\} \\
\mathcal{T}_i &= \{ \text{ feasible time patterns for instructor } i\} \\
\mathcal{T}_r &= \{ \text{ feasible time patterns for room } r\} \\
t(l) \in \mathcal{T} &= \text{ the time pattern associated with slot } l \\
r(l) \in \mathcal{R} &= \text{ the classroom associated with slot } l \\
\mathcal{S}_i &= \{ \text{ sections to which instructor } i \text{ is assigned}\}
\end{aligned}$$

the classroom scheduling problem is modeled as the assignment of a slot, l, to each section, s after Mulvey [13]. That is, we wish to find optimal values for x_{sl} for each section s and slot, $l \in \mathcal{L}_s$ where

$$x_{sl} = \begin{cases} 1 & \text{if section } s \text{ is assigned to slot } l \in \mathcal{L}_s \\ 0 & \text{otherwise} \end{cases}$$

Assuming $\mathcal{T}$ contains only nonoverlapping time patterns, CSP can be formulated as a nonlinear, multiple objective integer program as follows. Let $\epsilon_{s_1 s_2}$ be the expected student

conflicts if sections s_1 and s_2 meet simultaneously (i.e., the expected number of students jointly enrolled in s_1 and s_2). And, ω_{sl} is the value of the instructor preference for meeting section s using the time pattern of slot l. Similarly, let $\delta_{s_1 l_1, s_2 l_2}$ be the value of instructor preference for a relation between section s_1 assigned slot l_1 and section s_2 assigned l_2 (e.g., preference for back-to-back courses) where s_1, $s_2 \in \mathcal{S}_i$. Finally, ρ_{sl} is the "room fit" computed by dividing enrollment in section s by the room capacity (stations) of room $r(l)$ and considering other criteria such as distance from the professor's office.

Now, it is desired to

$$\text{minimize} \quad Z_1 = \max_{\substack{s_1, l_1, \\ s_2 \neq s_1, \\ l_2 : t(l_2) = t(l_1)}} (\epsilon_{s_1 s_2} x_{s_1 l_1} x_{s_2 l_2}) \tag{1}$$

$$\text{maximize} \quad Z_2 = \sum_s \sum_{l \in \mathcal{L}_s} (\omega_{sl} + \rho_{sl}) x_{sl} + \sum_{\substack{s_1, s_2 \in \mathcal{S} \\ s_2 \neq s_1}} \sum_{l_1, l_2 \neq l_1} \delta_{s_1 l_1, s_2 l_2} x_{s_1 l_1} x_{s_2 l_2} \tag{2}$$

(CSP) subject to:

$$\sum_{\substack{s \in \mathcal{S}_i, \\ l \in \mathcal{L}_s : t(l) = t}} x_{sl} \leq 1 \qquad \text{for all } i, t \in \mathcal{T}_i \tag{3}$$

$$\sum_{s : l \in \mathcal{L}_s} x_{sl} \leq 1 \qquad \text{for all } l = (r, t \in \mathcal{T}_r) \tag{4}$$

$$\sum_{l \in \mathcal{L}_s} x_{sl} = 1 \qquad \text{for all } s \tag{5}$$

$$x_{sl} = 0 \qquad \text{for all } s, l \notin \mathcal{L}_s \tag{6}$$

$$x_{sl} \in \{0, 1\} \qquad \text{for all } s,\ l \tag{7}$$

Equations (1) and (2) reflect current management practice at Purdue and were developed through a series of interviews with decision makers involved in the master scheduling process. They are not exhaustive, but represent the most widely applied scheduling criteria. Equation (1) indicates that a primary goal is to minimize the maximum expected common enrollment between two sections scheduled at the same time. A minimax objective reflects the fact that a few conflicts in several sections (which might preclude a few students from getting all their courses) are preferable to keeping the average (or total) conflicts small but allowing an arbitrarily bad worst case. The minimax objective is preferred because a bad enough worst case may prevent affected courses from being offered at all.

Equation (2) contains terms related to secondary scheduling objectives: maximizing in-

structor time preferences, room fit and paired section and slot preferences such as requests for sections to be scheduled back-to-back. The linear term equally weights instructor room and time preferences and the quality of the room fit. Including a measure of how well the number of stations in a room match the section requirement in the room fit may at first seem inconsequential. If unused rooms are included when computing aggregate room utilization, all solutions which feasibly schedule all sections will have the same utilization. However, using capacity utilization as a secondary room selection criteria should lead to qualitatively better schedules.

The quadratic term in (2) models second order instructor preferences. Common preferences include the desire to teach courses with a one hour break, on the same day, and back-to-back. If preferences are specified by an instructor, relations are generated for all pairs of that instructor's sections. As implemented, coefficient values and weights are controlled by the decision maker to reflect the relative importance of various preferences. The value of second order preferences, $\delta_{s_1 l_1, s_2 l_2}$, may also be a function of the difference between a desired relation and that represented by the slots. For example, if back-to-back sections are desired, the farther apart the times for the slots are, the less value the paired choices are given.

Set-packing constraints (3) and (4) restrict room and instructor assignments to available capacity (preclude resource conflicts). Note that (3) and (4) are valid instructor and room conflict constraints only if time patterns, $t \in \mathcal{T}$, do not overlap. While nonoverlapping time patterns are assumed here for exposition, the solution approach described in the next section relaxes this assumption. Equation (5) is a multiple-choice constraint that requires exactly one slot (room-time) assignment for each section. Finally, equation (6) restricts room assignments to those that meet or exceed section seating and other requirements and time pattern choices to those that match meeting time requirements for each section.

4 Schedule Generation

While CSP models the essential elements of the Purdue classroom scheduling problem, efficient solution is complicated by the multiple, nonlinear objective functions and binary decision variables. Also, while assuming nonoverlapping time patterns simplifies exposition, many of the time patterns in use at Purdue do overlap. Therefore, rather than solving CSP directly, a two-phase approach is used. This approach efficiently solves CSP while relaxing the nonoverlapping time pattern assumption.

The two phases of the solution approach consist of preprocessing followed by a computerized search over a simple, but general model of the scheduling problem. Preprocessing identifies feasible classroom and time pattern choice sets and therefore the set of candidate slots, $\mathcal{L}_s$, for each section. A local search algorithm is used in the second phase to find good schedules. The simplified model and preprocessing and search algorithms are presented in this section. CHRONOS, a scheduling support system that implements this two-phase approach and provides data management and schedule evaluation capabilities is described in the next section along with test results.

4.1 Search Model

CSP can be reformulated as a *multiple choice quadratic vertex packing* (MCQVP) problem. MCQVP effectively models resource conflict constraints with overlapping time patterns and allows a great deal of flexibility for problem instantiation. An instance of MCQVP, in the classroom scheduling context, is partially specified by an indexed set of paired room and time pattern (slot) candidates, $\mathcal{L}_s$, for each section, s. Preference information, and a list of conflicting choice pairs (i.e., pairs of choices that cannot be simultaneously selected) is also required.

The union of the slot candidate sets forms a set of indexed *choices* with choice j modeled with a binary choice variable, x_j, where x_j is identified with section $s(j)$. The jth choice therefore is defined by a section, room, and time pattern triplet, $[s(j), r(j), t(j)]$. If $x_j = 1$, section $s(j)$ is assigned to slot $l(j)$ (room $r(j)$ and time pattern $t(j)$). Choice set $\mathcal{J}_s = \{j : s(j) = s\}$ denotes the set of choice indexes associated with section s.

Coefficients c_j and d_{jk} quantify first and second order room and time preferences. Specifically, $c_j = \omega_{s(j)l(j)} + \rho_{s(j)l(j)}$ where $\omega_{s(j)l(j)}$ and $\rho_{s(j)l(j)}$ are instructor preference and room fit

values as defined for CSP. Also, $d_{jk} = \delta_{s(j)l(j),s(k)l(k)}$ is the cost of simultaneously selecting choices j and k (for $s(j) \neq s(k)$).

$$\text{maximize} \qquad Z = \sum_j c_j x_j - \sum_j \sum_k d_{jk} x_j x_k \tag{8}$$

(MCQVP) subject to:

$$\sum_{j \in \mathcal{J}_s} x_j = 1 \qquad \text{for all } s \tag{9}$$

$$x_j + x_k \leq 1 \qquad \text{for all conflicting choices } j \text{ and } k,\ j \neq k \tag{10}$$

$$x_j \in \{0, 1\} \tag{11}$$

Equation (8) represents the desire to maximize both first and second order assignment preferences for room and time assignments and can be obtained directly from (2). In a sense, the d_{jk} coefficients price "soft" conflicts. Soft conflicts occur when second order preferences such as scheduling two sections back-to-back are not honored and contrast with "hard" conflicts. Hard conflicts are precluded from feasible schedules by constraints of the form of equation (10). Hard conflicts arise when two sections are scheduled for the same room or instructor, or have too many common students. While no hard conflicts are allowed in a feasible schedule, soft conflicts are allowed but minimized by pricing.

The remainder of CSP is mapped to MCQVP as follows. Equation (6) is handled by restricting room and time pattern choices during preprocessing, based on section requirements. Equations (3) and (4) are reformulated as vertex packing constraints in (10) (a constraint for each section-slot choice pair) to allow overlapping time patterns. Objective (1) is also decomposed into vertex packing constraints in (10) using a threshold for $\epsilon_{s_1 s_2}$ to filter important student-based conflicts or specified conflict sets. The MCQVP model therefore maps directly to equations (2), (5), and (7), with (1), (3) and (4) reformulated as vertex packing constraints.

There are two major advantages to the MCQVP formulation. First, MCQVP is a more general model than CSP. Virtually any relation between sets or, equivalently, pairs of sections can be accomodated. Such relations are extremely important in classroom and other resource scheduling problems. Incorporating new relations requires relatively minor changes to preprocessing and other support software while leaving the search code unchanged. In

addition, our research has shown that efficient local search methods work remarkably well with MCQVP [12].

4.2 Preprocessing

Preprocessing algorithms identify time pattern and classroom candidates, combine them to create choices for each section, and generate MCQVP constraints for the search. Choices are identified for section s by filtering time pattern and room lists for feasible candidates and applying further restrictions to these candidates to yield $\mathcal{T}_s$ and $\mathcal{R}_s$. $\mathcal{T}_s$ and $\mathcal{R}_s$ are then combined to yield the section-slot choices, $\mathcal{L}_s$, for each section, s, and explicitly guarantee satisfaction of equation (6). Choice preference values, c_j, are obtained by adding room and time pattern preferences, normalized between 0 and 100. While CHRONOS allows differential weighting of room and time pattern preferences, our work to date indicates that, while some decision makers may not include one or both, if included, they tend to treat them the same.

Candidate time patterns are selected by matching section time pattern requirements with a list of patterns maintained in the CHRONOS database. Candidate patterns must also satisfy explicit constraints on the times that sections can be taught and instructors are available. Time pattern preference values (later combined with room preference values for a choice) are based on designated day and time block ranges obtained from each instructor. For a given time pattern and section, the preference value is obtained by assigning a weight of $\frac{1}{3}$ to day matches and $\frac{2}{3}$ to time block overlaps. Time block values are reduced exponentially as the amount of overlap between instructor preferences and the time pattern decreases.

Room candidate selection begins, like time pattern selection, by constructing a list of feasible room candidates for each section based on size and other requirements. A fitness (preference) metric is also computed for each candidate. Room preference values are currently computed by weighting location (relative to the instructor's building) 20% and the room fraction of the seats utilized 80%. The initial room candidate list may be pared to a user-specified maximum size by applying a load balancing heuristic.

The load balancing heuristic begins by initializing each room with a pseudo "capacity" of 3 to 4 times its available hours, depending on the maximum number of room candidates per section specified by the user. We have found that 3 to 5 room candidates per section provides enough flexibility for the search algorithms. Room candidates are then selected for

each section, starting with the section with the fewest feasible rooms. Candidates with high fitness values are chosen first with ties going to rooms with the most remaining "capacity". As candidates are chosen, their remaining "capacity" is reduced by the hours required by the section. This heuristic spreads room candidates evenly among the sections, reduces the size of the search space and, by favoring rooms with high fitness values, improves the quality of solutions explored during the search. It is particularly advantageous where there are relatively large sets of identical rooms to schedule.

4.3 Local Search

The dynamic biased sampling with strategic oscillation (DBSO) algorithm is an enhanced local search algorithm developed for solving instances of MCQVP. DBSO is based on principles exploited by simulated annealing [11] and tabu search [7, 8, 10] algorithms. These algorithms incorporate strategies that balance *diversification* into new regions of the solution space with *intensification* around good local optima. The goal is to retain the simplicity and efficiency of local search while avoiding being trapped at local optima. DBSO uses dynamic biased sampling and strategic oscillation mechanisms to achieve this goal.

The DBSO search begins with an initial solution, $\mathbf{x}^0$. *Moves* then transform an existing solution into a new one in the current *neighborhood* at each step. Formally, a move, m, is a function that maps a trial solution at iteration $i-1$, $\mathbf{x}^{i-1}$, to a new solution, $\mathbf{x}^i$, at iteration i,

$$m : \mathbf{x}^{i-1} \mapsto \mathbf{x}^i$$

where $\mathbf{x}^i$ is a solution at the end of iteration i in the *neighborhood* of $\mathbf{x}^{i-1}$.

The search neighborhood used to solve MCQVP includes solutions that violate constraints in (10). Equation (9) feasibility is maintained, however. Notice that an initial solution satisfying (9) can be easily found by arbitrarily choosing exactly one assignment per section. And, given such an initial solution, any pairwise interchange (exchange) move resulting in a new choice for a single section is admissible for this neighborhood.

Formally, each DBSO move complements a pair of choices that correspond to alternative assignments for a single section, s. For example, assume that x_k is the solution variable for section s, ($x_k = 1$) at the beginning of the move and $x_{k'}$ is the variable selected to replace x_k as the choice for section s. At the beginning of the swap, $x_{k'} = 0$ since one and only one of the choices for each section can be in the solution at a time. Of course, $x_k = 0$ and $x_{k'} = 1$

when the exchange is complete.

A two-step process is used to choose a swap at each iteration. Selection begins by sampling about 30 percent of the sections for exchange consideration. The best exchange available within the section sample is then chosen based on changes in the objective function (equation (8)) and conflict feasibility (equation (10)). The relative importance of improving the objective function versus removing conflicts is varied during the search by strategic oscillation of the conflict penalty.

4.3.1 Dynamic Biased Sampling

Sections are sampled for move evaluation using *dynamic biased sampling* (DBS). This approach is responsible for strategic diversification in a relatively small region about a local optimum. Sections that have not been selected recently for evaluation and have had few exchanges made are assigned a high priority for sampling. By biasing sampling in favor of sections with high priority the search is forced to look at a variety of solutions. Thus, dynamic biased sampling may be viewed as an alternative method for implementing concurrent frequency and recency based tabu search as well as a type of candidate list strategy [9].

However, since sections are sampled randomly, every section has a non-zero (if small) probability of being chosen for evaluation, giving DBS other desirable properties. To the extent that low priority sections may be occasionally selected, DBS provides a mechanism to override low priority sections' "fuzzy" tabu status as well. When the best move corresponds to a section with a low priority, it will be taken. Sampling therefore provides an intensification mechanism similar to the Glover's aspiration criteria [10]. DBS therefore implements many of the advanced tabu search approaches automatically, with sampling parameters providing a simple control mechanism.

4.3.2 Strategic Oscillation

Penalty weight oscillation drives the search toward feasible solutions when the penalty is high and causes it to ignore feasibility entirely when the penalty reaches zero. The net effect is to *diversify* the search as it approaches a local optimum by reducing the conflict constraint (10) penalty. Hence, as the penalty is reduced, the search moves away from the local optimum into a "good" (i.e. high value, but infeasible) region of the search space. The cycle is completed as the solution is again driven feasible by high penalty values and the

objective function decreases.

Figure 1 illustrates the oscillation with a search trajectory for a single problem instance. The graph shows a shaded region delineated by two solid lines, with a point plotted for the solution at each iteration. The upper line plots the objective function (Z) value and the lower line plots this value *minus* the hard conflicts. Therefore, the shaded region represents the degree of equation (10) infeasibility at each iteration and feasible solutions are easily identified when the two curves come together and the shaded region disappears.

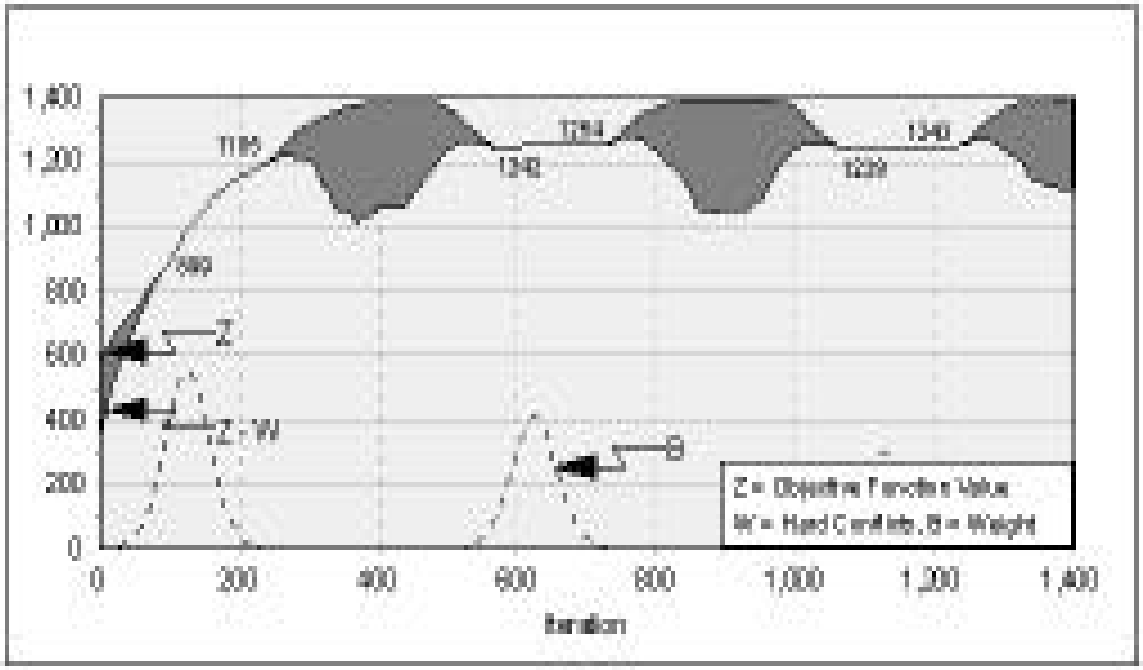

Figure 1: Search Algorithm Trajectory

The dotted curve on the graph in Figure 1 shows how the infeasibility penalty oscillates over time. The search first approaches a local optimum around iteration 200 and the search begins to diversify. Solutions are then found in a new (infeasible) region. The first cycle ends around iteration 600 with a new feasible solution found as the the penalty is increased. The cycle is repeated as the penalty oscillates between its minimum and maximum.

5 CHRONOS Implementation

The solution approach described in the previous section has been implemented in CHRONOS, a prototype system to support course scheduling at Purdue. CHRONOS provides computerized data management, "what-if" modeling, and search capabilities to support the decentralized scheduling process used at Purdue. The system is computationally efficient and easily modified to to incorporate new preferences and constraints and was designed to be linked with existing and planned information systems. The underlying search algorithm has been extensively tested on randomly generated and representative Purdue problems. CHRONOS is currently deployed for use in scheduling approximately 500 sections in 31 large lecture rooms at Purdue and is being enhanced for use by individual departments in a campus-wide client-server scheduling system.

5.1 Data Input

Schedules produced using CHRONOS are influenced by the user through the type and quality of data provided and the values of various parameters controlling preprocessing and search. Data required by CHRONOS describes sections to be taught, expected student demand for courses, instructors, rooms, and available times. These data are stored and maintained in a relational database and converted during preprocessing to the MCQVP model format for schedule evaluation and optimization. Parameters include cutoff values for generating student-based conflict constraints (section pairs may be explicitly specified as an option) and starting point, duration, and other search control values.

Classroom, time pattern, and instructor tables provide the basic data for scheduling with CHRONOS. Rooms are specified by location, size, type, and availability. Acceptable time patterns are defined by the days and the time block when meetings will be held. At Purdue, most time patterns are defined for either Tuesday, Thursday or Monday, Wednesday, Friday day sets. Instructor data include availability, location, and first and second order meeting time preferences. First order preferences specify which days and time are most desirable. Second order preferences indicate whether the instructor wishes to teach courses on the same days, back-to-back, with a one hour separation between them, and so on.

Section descriptors include the maximum size (number of students) and the instructor assigned, if known, as well as room and meeting time requirements. Room requirements such as room type, size and teaching facilities are specified for each section. The number

of times a section meets (days per week) and the total meeting time specify time pattern requirements. Times when sections are restricted from being taught due to course demand patterns or for other pedagogical reasons may also be designated. Room and/or time pattern assignments may optionally be specified as an initial solution for the search or fixed.

Schedule conflicts arise if two sections are taught at times that overlap and have the same instructor or room assigned or a significant number of students in common. Students are assigned to sections outside the CHRONOS system following classroom-section (master) scheduling. When two courses that a student wants are scheduled at the same time, he or she will not be able to take one. Thus, master scheduling must consider expected demand for all pairs of courses to ensure that the chance of blocking students from courses is minimized. CHRONOS accepts estimates of joint demand in several forms, but the result is always a list of pairs of sections that must have all meetings scheduled at different times to satisfy student demand. The maximum number of expected student conflicts can be minimized by eliminating pairs from the list, beginning with those with relatively minor student conflicts until a schedule with no "soft" student conflicts is found.

5.2 Application Tests

CHRONOS was originally tested using six Purdue scheduling problems. Test problem characteristics and setup times (in minutes) are shown in Table 2. Problems R1 to R4 represent single departments, using their predefined room sets. R1, R2, and R3 are engineering departments with relatively low service loads whereas R4 is a large department with many nonmajor students. Problem R5 represents the core courses of a remote campus and R6 is an instance of the large lecture room scheduling problem currently handled by SMAS.

The number of rooms shown for each problem is the total and does not reflect availability. Many rooms were only available for scheduling part of the time, particularly for problems R1-R3. Conflict density is the percentage of choice pairs corresponding to the same room, instructor, or class and is therefore a measure of how heavily a problem is constrained.

Setup times are given for translating database information into numerical model coefficients and for initializing the search. These times can be significant as they are a function of the product of the number of sections (N_s) and the number of choices (N_x). An analysis of the data using a linear regression model yielded:

$$\text{Model setup time} = 6.3 + 8.2 \times 10^{-6} N_s N_x$$

and

$$\text{Search setup time} = -1.3 + 22.3 \times 10^{-6} N_s N_x$$

with R^2 values over 0.99. Search time per iteration is a linear function of the total number of choices, N_x.

Table 2: Test Problem Characteristics

Problem	Sections	Rooms	Choices	Conflict Density	Z Bound	Setup Time	
						Model	Search
R1	67	29	731	.0114	395.9	3.8	1.0
R2	196	39	5,707	.0054	682.3	16.5	23.8
R3	162	31	5,348	.0056	567.7	14.4	18.5
R4	362	41	25,166	.0048	2044.5	83.4	200.0
R5	137	47	5,784	.0047	464.5	15.5	17.0
R6	492	31	17,313	.0024	4006.9	73.2	190.0

In general, the objective function coefficients corresponding to first order preferences (the linear term in equation (8)) reflected room and time pattern "fit" with the class size and instructor preferences respectively and ranged between 0 and 10. The range for problems R2, R3, and R5 was 0 to 5 because time preferences were not specified. Second order preferences (soft conflicts) corresponding to the quadratic term in equation (8) were very small for all problems.

5.3 Test Results

Results obtained for each problem are shown in Table 3. Four runs, made with three random starting points and one "given" starting point obtained from the previous year's schedule are summarized for each problem. All runs were made on an IBM RISC 6000 model 520 with the C source code compiled using the "cc" compiler without optimization. Parameter values were calibrated prior to these runs using a small set of randomly generated synthetic problems. Likewise, the maximum search time was determined empirically by finding times beyond which no further improvement was likely for synthetic problems of various sizes. While it is possible that better solutions would be found with longer runs, the closeness of

the average solutions to bounds suggests that the point of diminishing returns was reached in most runs.

Bounds were obtained by summing the maximum linear (preference) objective function coefficients for each section. The choice set for each section was selected during preprocessing by eliminating infeasible and low preference value choices. Hence, while the bounds are for restricted problems, they often bound the original problems as well. The number of runs (out of 4) for which feasible (no conflict) solutions were found is also shown in Table 3, with statistics on the minimum number of unresolved conflicts for each problem. Finally, statistics on the best solution value (Z-Best) and the time to find it (T-Best) obtained for all starting points (runs), are presented for each problem.

Table 3: Search Results

Prob	Srch Time (Min)	Z Bound	No. Feas. Runs	Minimum Hard Conflicts		Z-Best		Z-Best/ Z-bound	T-Best (minutes)	
				Avg	Sdv	Avg.	Sdv		Avg.	Sdv
R1	1.0	395.9	4	0.0	0.0	388.9	0.0	.982	0.20	0.08
R2	7.0	682.3	3	0.2	0.5	676.5	1.3	.991	3.27	2.43
R3	5.0	567.7	4	0.0	0.0	560.0	0.8	.986	2.31	2.18
R4	30.0	2044.5	2	1.5	1.7	1967.0	3.9	.962	7.94	0.24
R5	5.0	464.5	4	0.0	0.0	464.0	0.1	.999	1.84	0.94
R6	35.0	4006.9	2	0.5	0.6	3944.4	7.0	.984	5.99	1.70

As shown in Table 3, cases R1, R3, and R5 proved easy for the DBSO algorithm. Feasible solutions were obtained from all starting points with objective function values approaching the upper bound. The other three problems, R2, R4, and R6, were more difficult. The algorithm found feasible solutions from better than 50 percent of the starting points and averaged relatively few conflicts when feasible solutions were not found. The best objective function values were always very close to the linear bound.

5.4 Large Lecture Rooms Experience

The results in Table 3 and equally promising results from the large computational experiment reported in [12], convinced Purdue's Department of Space Management and Academic Scheduling (SMAS) to use the system to schedule large lecture rooms while developing client-server applications for individual departments. For the past several semesters CHRONOS

has either provided the large lecture rooms schedule or been run in parallel with manual techniques to assess its effectiveness. Additional lessons have been learned and ehancements made as a result of operating in a production mode.

CHRONOS has performed well by quantitative measures such as the number of conflicts and the objective function value for resulting schedules. As often happens in application, however, a number of more subtle details created difficulites. Some difficulties were addressed with minor modifications to the system. Approaches are being developed for others.

Minor revisions included changes to:

- Incorporate distance into room selection
- Allow selection of rooms with projectors
- Select the same room for back-to-back sections

Distance and projector requirements were accommodated by changing the room candidate selection preprocessing logic. Initial runs allowed too many options, sometimes requiring professors and students to travel unacceptably long distances. The revised system considers the location of candidate classrooms relative to the professor's office much more carefully. Similarly, some classes require projectors for effective teaching. The initial version of CHRONOS had no way to take this into consideration. Revisions to the classroom database and preprocessor now identify those rooms having projectors and allow choices to be limited to them. Additional criteria can be easily added as the need is identified.

Professors teaching large lecture classes often request that classes be scheduled "back-to-back", or in successive time blocks on the same days. Initial CHRONOS data coding implemented back-to-back only in terms of times, leaving open the possibility that a professor might teach courses in adjacent time blocks but in rooms far apart. Only a minor change to the preprocessor was required to define choice pairs as back-to-back only if they have both the same room and adjacent times. Other relations can be easily added as well, should they be needed.

More serious challenges are a result of the system's optimization paradigm. Sometimes the quest for optimal solutions produces unintended side effects. Difficulties with CHRONOS fall in two main categories:

- Preferences and fairness

- Robustness.

While neither precludes use of CHRONOS, in different ways each has the potential to prevent realization of its full potential.

Preference and fairness issues arise primarily from variation in detail supplied by departments requesting classes in large lecture rooms. Some express very precise preferences for rooms and times; others give only vague specifications. CHRONOS maximizes expressed preferences. As a result SMAS staff have found schedules biased against those with imprecise requests. In one sense such departments are the "good citizens", willing to be flexible about rooms and times. But, optimized schedules tend to punish them with poor times and locations to accommodate more explicit requirements of others. Attempts to standardize departmental input show promise in this area.

Anecdotal evidence suggests that brittleness, or a lack of robustness, often a characterizes optimization-based schedules, and CHRONOS schedules often exhibit this property. Robust schedules are important because SMAS produces its large lecture room schedule relatively early in the semester, but numerous changes occur as registration proceeds. Each change requires rearrangement of the schedule, and it is preferable that this be done with a minimum number of assignments changed. Previous manual schedules, which were not so tightly optimized, left slack in the schedule, making incremental changes relatively easy to arrange. In contrast, SMAS schedulers find that changing CHRONOS schedules often leads to a long cascade of revisions to other class assignments. Current work to address this problem includes building strategic "holes" into the schedule to facilitate last minute changes.

6 Conclusions

Many of those doing class scheduling at Purdue are nearing retirement, and schedulers who follow them will lack much of their historical perspective and rapport with the faculty. Experience with CHRONOS has shown that a scheduling support system to aid more junior and less experienced staff is well within the capability of modern optimization-based software. Placing CHRONOS in the hands of less sophisticated users in a client-server environment is the ultimate goal at Purdue and work is proceeding toward full implementation. The user interface requires far more polishing, and perceived difficulties with fairness and robustness have yet to be completely addressed, at least in the context of the large lecture room problem.

These concerns are not insurmountable, but they are as important to ultimate success as further development of CHRONOS's underlying mathematical structure.

References

[1] Aubin, J. and Ferland, J.A., "A Large Scale Timetabling Problem", *Computers and Operations Research*, 16:1, 1989, 67-77.

[2] Baker, K.R., *Introduction to Sequencing and Scheduling*, John Wiley and Sons, 1974, 305 pages.

[3] Bean, J.C, Birge, J.R., Mittenthal, J. and Noon, C., "Match-up Scheduling with Multiple Resources, Release Dates, and Disruptions", *Operations Research*, 39, 1991, 470-483.

[4] Carter, M.W., "A Lagrangian Relaxation Approach to the Classroom Assignment Problem", *INFOR*, 27:2, May 1989, 230-246.

[5] Dinkel, J.J., Mote, J. and Venkataramanan, M.A., "An Efficient Decision Support System for Academic Course Scheduling", *Operations Research*, 37:6, Nov.-Dec. 1989, 853-864.

[6] Ferland, J.A. and Roy, S., "Timetabling Problem for University as Assignment of Activities to Resources", *Computers and Operations Research*, 12:2, 1985, 207-218.

[7] Glover, F., "Tabu Search — Part I", *ORSA Journal on Computing*, 1:3, Summer 1989, 190-206.

[8] Glover, F., "Tabu Search — Part II", *ORSA Journal on Computing*, 2:1, Winter 1990, 4-31.

[9] Glover, F., "Candidate List Strategies and Tabu Search", *Working Paper, Center for Applied Artificial Intelligence*, University of Colorado, June 1989, 38 pages.

[10] Glover, F., "Tabu Search: A Tutorial", *Working Paper, Center for Applied Artificial Intelligence*, University of Colorado, February 1990, 60 pages.

[10] Hertz, A.,, "Tabu Search for Large Scale Timetabling Problems", *European Journal of Operational Research*, 54, 1991, 39-47.

[11] Kirkpatrick, S., Gelatt, C.D. and Vecchi, M.P, "Optimization by Simulated Annealing", *Science*, 220, 671-680, 1983.

[12] Mooney, E.L., Rardin, R.L., "Tabu Search for a Class of Scheduling Problems", *Annals of Operations Research*, 41, 1993, 253-278.

[13] Mulvey, J.M., "A Classroomtime Assignment Model", *European Journal of Operational Research*, 9, 1983, 64-70.

[14] Nordlund, G., "Timetabling Problem Research", *Unpublished Technical Report*, Space Management and Academic Scheduling Office, Purdue University, April 1986.

[15] Tripathy, A., "School Timetabling — A Case in Large Binary Integer Linear Programming", *Management Science*, 30:12, December 1984, 1473:1489.

[16] White, G.M. and Wong, K.S., "Interactive Timetabling in Universities", *Computers in Education*, 12:4, 1988, 521-529.