University Course Timetabling with Soft Constraints

Hana Rudová[1] and Keith Murray[2]

[1] Faculty of Informatics, Masaryk University
Botanická 68a, Brno 602 00, Czech Republic
hanka@fi.muni.cz

[2] Space Management and Academic Scheduling, Purdue University
1128 ENAD, West Lafayette, IN 47907, USA
kmurray@purdue.edu

Abstract. An extension of constraint logic programming that allows for weighted partial satisfaction of soft constraints is described and applied to the development of an automated timetabling system for Purdue University. The soft constraint solver implemented in the proposed solution approach allows constraint propagation for hard constraints together with preference propagation for soft constraints. Repair search methods are proposed to improve upon initially generated (partial) assignments of the problem variables. The model and search methods applied to the solution of the large lecture room component are presented and discussed along with the computational results.

1 Introduction

This paper describes the design approach and solution techniques being developed for an automated timetabling system at Purdue University. The initial problem considered here is the design of an intelligent system to assist with construction of the large lecture component of the university's master class schedule. The design anticipates expanding the scope of the problem to accommodate a demand-driven approach to timetabling all classes at the University. In *demand-driven timetabling*, student course selections are utilized to construct a timetable that attempts to maximize the number of satisfied course requests. In the initial problem we consider the course demands of almost 29,000 students enrolled in approximately 750 classes, each taught several times a week.

A solution to the timetabling problem is being developed using *constraint logic programming* [21, 13]. CLP is a respected technology for solving hard problems which include many complicated (non-linear) constraints [11]. Its main advantages over other frameworks are the declarative nature of problem descriptions via logical constraints, and a constraint propagation technique for reducing the search space.

Timetabling problems [7, 20] are often over-constrained, which is the case with our problem since it is not possible to satisfy all requests of students for enrollment to specific courses. Preferential requirements for time and room assignment may also lead to the problem being over-constrained. *Soft constraints* [8, 4] can be applied to define all requirements declaratively rather than encapsulating many of them into the control part of the problem solution. In our problem solution, we have applied a weighted CSP [8] approach which considers weights/costs for each constraint and minimizes the weighted sum of unsatisfied constraints.

Soft constraints are often applied to solve timetabling problems with the help of constraint satisfaction. They are mostly applied via the standard constraint satisfaction method, which offers no special support for more effective resolution of the soft constraints. Golz et al. [10] applies the typical solution — the given unary soft constraints, with priorities, are integrated into the solution search through value and variable ordering heuristics. An optimization constraint was applied for solving medium-sized problems [12]. Abdennadher et al. [1] describe the solution of a department-sized problem, including soft constraint solver, for a cost-based approach implemented with Constraint Handling Rules [9].

Our work includes development of a new solver for soft constraints. The solver was implemented as an extension of the CLP(FD) library [5] of SICStus Prolog. This approach is of particular importance for the construction of demand-driven schedules where complete satisfaction of all constraints is not feasible. Soft constraints have also been applied to accommodate the preferences of instructors with respect to time and room assignments for their classes. We will also describe a new repair search technique which allows us to find a solution even when the problem is over-constrained.

The following sections of this paper will give a more complete description of our timetabling problem. Section 3 explains the soft constraint solver that was implemented, together with the search procedures used on this problem. This is followed by a description of how the problem has been modeled, including the representation of soft and hard constraints. In addition, the methods applied to the search for a feasible solution are discussed. Computational results are discussed in Sec. 5. Comparison with other approaches to solving demand-driven timetabling problems is presented in Sec. 6. The final section reviews the results of our work and looks to future extensions of the problem solution and soft constraint solver improvements.

2 Problem Description

At Purdue University, the timetabling process currently consists of constructing a master class schedule prior to student registration. The timetable for large lecture classes is constructed by a central scheduling office in order to balance the requirements of many departments offering large classes that serve students from across the university. Smaller classes, usually focused on students in a single discipline, are timetabled by "schedule deputies" in the individual departments. This process has been tailored to the political realities of a decentralized university, where faculty can be quite put off by the idea of having a central office tell them when to teach, or even by providing such an office with much information about when they are available to teach.

A natural decomposition of the university timetabling problem has therefore resulted, consisting of a central large lecture timetabling problem and 74 disciplinary problems. The large lecture problem consists of approximately 750 classes having a high density of interaction that must fit into 41 lecture rooms with capacities up to 474 students. In this problem, the course demands of almost 29,000 students out of a total enrollment of 38,000 must be considered. The departmental problems range from only a few classes up to almost 700, with an average size of slightly more than 100 organized classes. The largest departmental problems are simplified by having many sections of the same course that are offered at multiple times.

The timetable maps classes (students, instructors) to meeting locations and times. A major objective in developing an automated system is to minimize the number of potential student course conflicts which occur during this process. This requirement substantially influences the automated timetable generation process since there are many specific course requirements in most programs of study offered by the University.

To minimize the potential for time conflicts, Purdue has historically subscribed to a set of standard meeting patterns. With few exceptions, 1 hour x 3 day per week classes meet on Monday, Wednesday, and Friday at the half hour. 1.5 hour x 2 day per week classes meet on Tuesday and Thursday during set time blocks. 2 or 3 hours x 1 day per week classes must also fit within specific blocks, etc. Generally, all meetings of a class should be taught in the same location. Conforming to this set of standard meeting patterns will be seen to have a strong influence on both the problem definition and the solution process, since the meeting patterns defined for each class introduce hard restrictions on the acceptability of any generated solution.

The other major constraints on the problem solution are instructor availability and a limited number of rooms available with sufficient capacity, specific equipment, and suitable location. Some of these constraints must be satisfied; others can be introduced within an optimization process in order to avoid an over-constrained problem.

This paper describes the construction of the timetable for the large lecture problem. The nature of this problem is the same as the timetabling problem at the university level. There is a demand for specific class combinations to meet individual student's needs. Meeting patterns direct possible time and location placement. Classroom allocation must respect instructional requirements and the preferences of faculty. All instructors may have specific time requirements and preferences for each class.

Another aspect of the timetabling problem that must be considered here is the need to perform an initial student sectioning. Most of the classes in the large lecture problem (about 75 %) correspond to single-section courses. Here we have exact information about all students who wish to attend a specific class. The remaining courses are divided into multiple sections. In this case, it is necessary to divide the students enrolled to each course into sections that will constitute the classes. Without this initial sectioning it is not easy to measure the desirability or undesirability of having classes overlap in the timetable. Our current approach sections students in lexicographic order before joint enrollments between classes are computed. This gives us the worst case possibility. The university currently processes a precise student schedule after the master class schedule is created, however, which should introduce some improvements. Possible directions for improving this solution will be discussed in the final section.

3 Solver for Soft Constraints

Constraint propagation algorithms for the soft constraints are implemented as a part of the preference constraint solver [17]. This constraint solver is built on top of the CLP(FD) solver of SICStus Prolog [5] and implemented with the help of attributed variables. An advantage of this implementation is the ability to include both hard constraints from the CLP(FD) library and soft constraints from a new *preference solver*.

3.1 Preference Variables and Preference Propagation

The preference solver handles preferences for each value in the domain of the variable which will be called the *preference variable*. Each preference corresponds to a natural number indicating the degree to which any soft constraint dependent on the domain value is violated. An increase of the preference during the computation with soft constraints is called a *preference propagation*. Let us note the difference with constraint propagation, which removes values from the domain of the domain variable during the computation of hard constraints. Removal of domain values may also occur with preference variables. This corresponds to violation of a hard constraint. We may also set the degree of acceptable violation for any preference variable. If the preference associated with a value in the domain of the preference variable should exceed this limit, it is removed from the domain. This possibility is of particular interest for time (or classroom) variables since all classes should be relatively equal in importance.

Zero preference means complete satisfaction of the constraint for the corresponding value in the domain of the variable. Any higher preference expresses a degree of violation that would result from the assignment of this value to variable. All values which are not present in the domain of the preference variable have the infinite preference `sup`. Preferences for each value in the domain of the variable may be initialized with a natural number. This allows us to handle initial preferences of values in the domain of the variable.

For each preference variable, the preference solver maintains an additional domain variable (*cost variable*) having the current best preference of the preference variable as its lower bound. The initial upper bound is set to infinity. Any preference propagation results in an increase of the current best preference, with the lower bound of the cost variable being increased accordingly. The sum of all cost variables gives us the total cost of the solution (*cost function*).

Example 1. The unary soft constraint `pref(PA, [7-5, 8-0, 10-0], CostPA)` creates the preference variable `PA` with initial domain containing values `7`, `8`, and `10` and preferences `5`, `0`, and `0`, resp. It means that the value `7` is discouraged wrt. other values. Preferences for remaining values are assumed as infinite, indicating complete unsatisfaction. The domain variable `CostPA` is created with the domain `0..sup`.

3.2 Soft Disjunctive Constraints

Let us describe two basic binary soft constraints we have implemented.

```
soft_different( PA, PB, Cost )
soft_disjunctive( PStart1, Duration1, PStart2, Duration2, Cost )
```

The `soft_different` constraint expresses that the two preference variables `PA`, `PB` should have different values. The constant `Cost` gives us the cost for violation of this constraint. The `soft_disjunctive` constraint asks for the non-overlapping of the two tasks specified by the preference variables `PStart1`, `PStart2` and the constant durations `Duration1`, `Duration2`. Again the `Cost` is the weight of this constraint.

Algorithms for both constraints are based on the partial forward checking algorithm and inconsistency counts [8]. Let us take a look at the `soft_different` constraint. Once the first preference variable is instantiated to some value `X`, the inconsistency count for the second variable and the value `X` should be increased by `Cost`, i.e., the preference propagation is processed for this variable and value. The `soft_disjunctive` would process preference propagation for all values in the interval `X..(X+Duration-1)`.

Inconsistency counts are maintained for all of the unassigned variables and the values remaining in their domains. These can be applied during labeling and optimization, e.g., values with the smallest inconsistency count can be selected first (an optimistic approach). The change in inconsistency counts for each preference variable is also reflected by the corresponding cost variable, e.g., the smallest inconsistency count for the preference variable corresponds to the lower bound of its cost variable.

3.3 Search

The aim of our problem solution is to be able to search for the complete assignment of the preference variables giving the best possible satisfaction of all soft constraints. Since the evaluation of the assignment is given by the sum of the corresponding cost variables, we may apply a standard branch & bound algorithm.

Unfortunately it may not be easy to find any complete assignment. Mistakes in the assignment of some variable(s) may lead to a time consuming exploration of the search space with no complete solution. A complete assignment may not even exist due to conflicts among the hard constraints. Our approach is to find some initial partial assignment of the variables and subsequently repair it such that all, or at least most, of the variables have been assigned a value.

Let us describe the *initial search* first. For each preference variable, we maintain a count of how many times a value has been assigned to the variable. A limit is set on the number of attempts that are made to assign a value. Currently this limit corresponds to the size of the domain of each variable. If the limit is exceeded, the preference variable is left unassigned and we continue in the search. Whereas any assignment of a value to this variable will lead to a time consuming trashing[1], allowing several values to remain in its domain will not initiate a constraint propagation that would detect inconsistency with the already assigned variables. As a result of this search, we obtain a partial assignment of variables and the sets of values which were tried unsuccessfully for the remaining variables.

Any *repair search* can use the result of the former (initial or repair) search. First, we apply heuristics with the successfully assigned variables to give us a value for each variable to be tried *first*. Second, we apply heuristics with the remaining variables to give us values to be tried *last*.

This change in the value ordering can be included in the problem with the help of preferences. We can encourage certain values by increasing the initial preferences or discourage them by decreasing their initial preferences. This type of redefinition allows us to run a repair search using the same procedure as the initial search.

After construction of a partial assignment, the user can decide on a desirable continuation of the search. The following possibilities are of particular interest:

1. processing the repair search directed by the proposed value ordering heuristics;
2. changing variable ordering such that unassigned variables are tried first, then processing the repair search directed by the proposed value ordering heuristics;
3. defining other values to be tried first or last based on user input, and process the repair search directed by the updated value ordering heuristics;
4. relaxing some hard constraints based on user input, and processing the initial search or the repair search directed by the updated value ordering heuristics.

The first two possibilities are aimed at automated generation of a better assignment. The second approach seeks to find an assignment of variables which may be more difficult to locate in the search space using the original variable ordering. This step may lead to significant changes in the generated solution. The third possibility allows the user to direct the search into parts of the search space where a complete assignment of variables might more easily be found. The last step can be useful if the user discovers a conflict among some hard constraints. The repair searches can reuse most of the results from the former search and permute only some parts of the last partial assignment. Let us note that there is often an advantage to a user directed search in timetabling problems, since

[1] A large number of the variable assignments having to be undone repeatedly.

the user may be able to detect inconsistencies or propose a suitable assignment based on the partially generated timetable.

The user can also change the problem definition (add/delete/change classes or add/delete constraints) and apply the repair search while reusing results of the former solution for classes with no changes.

4 Problem Solving

We would like to describe a model for the timetabling problem which consists of variables for the time and room assignments of each class and of both hard and soft constraints, applying an approach described in the previous section. We also explore the control portion of the solution, which consists of the application of the proposed initial and repair searches.

4.1 Time and Classroom Variables

The domain of the time variables is represented by the natural numbers `0..104`, corresponding to `5` days of `21` half-hours. The domain of the classroom variables is represented by the natural numbers `1..Number_Of_Classrooms`.

Each class consists of between one and five meetings per week (typically two or three). All meetings have the same duration and are typically taught at the same time of day. Valid combinations of the number of meetings and the duration are called *meeting patterns*. Each meeting pattern (e.g., 1 hour x 3 meetings) has a defined set of days on which the meetings may be scheduled (e.g., Monday, Wednesday, Friday for 1 hour x 3 meetings). Interestingly, the start time of the first meeting of a class differs from the start times of the following meetings by a constant factor for most combinations (see Table 1). Excepting the MF (Monday and Friday) combination for 2 meetings per week

Table 1. Maximal sets of the possible combinations of days for class with given number of meetings per week (e.g., TTh means that course can have its meetings on Tuesday and Thursday).

Number of meetings	Possible combination of days
1	M or T or W or Th or F
2	MW or TTh or WF or MF
3	MWF
4	TWThF or MWThF or MTThF or MTWF or MTWTh
5	MTWThF

and the 4 meeting patterns (includes less than 1% of classes), one time variable is sufficient to contain the complete information about the start time of classes. This is a preference variable indicating the start time of the first meeting (`T1`).

It will be referred to as the *time preference variable*. The starting times of all remaining meetings (`T2,..., Tn`) are domain variables only, and may be related to the time preference variable by the simple constraint

$$\texttt{Ti}\#\!=\texttt{T1}+\texttt{Constant}*(\texttt{i}-\texttt{1}) \ . \tag{1}$$

Preferences associated with each value in the domain of the time preference variable allow us to express the degree to which any time assignment for a class is preferred or discouraged. The remaining domain variables may be referenced in the hard constraints (e.g., `serialized`), but they do not require the more expensive processing by the preference solver.

Since all class meetings should be taught in the same room, we suffice with only one common classroom variable for all meetings (called the *classroom preference variable*). As a preference variable, it associates a preference with each classroom expressing how desirable or undesirable it is for a given class.

4.2 Hard Constraints

Let us summarize the requirements which are implemented in the system using hard constraints:

1. meeting pattern specification;
2. prohibited or required times for classes;
3. class requires room with sufficient seating capacity;
4. class requires or prohibits some building(s) or room(s);
5. class requires or prohibits classroom of a specified generic type (computer, computer projection, audio recording, document camera, ...);
6. classes taught by the same instructor do not overlap;
7. sections of the same course do not overlap;
8. additional constraints over selected sets of classes: classes must be taught at the same times, on the same days, in the same classrooms, ...

Meeting pattern constraints relate the domain variables for all class meetings as was described in Eqn. 1. In addition, the domain of the time preference variable is reduced such that all invalid values are removed.

Example 2. A 1.5 hour x 2 meetings class is represented by the two variables `T1`, `T2`. The first of these is the time preference variable with the initial domain (`0..104`) reduced to the values `21,24,...,39` because the TTh combination is valid only. The second domain variable is related with `T1` by the constraint `T2 #= T1 + (21*2)*1`. The constant separating start times here is 21 periods x 2 days.

A 2 hours x 2 meetings class has MW, TTh, and WF as valid meeting day combinations. It is represented by the two variables `T1`, `T2` related by the same constraint as before. The reduced domain of `T1` corresponds to the values `0,4,8,12,16, 21,25,29,33,37, 42,46,50,54,58`.

Requirements 2–5 are implemented by domain reduction in the corresponding domains of the time and classroom preference variables. Requirements 6 and 7 are included with help of the constraint `serialized` which constrains tasks, each with a start time and duration, so that no tasks ever overlap. Built-in constraints of CLP(FD) library of SICStus Prolog are used to implement various requirements over selected sets of classes as mentioned in the item 8.

Additional hard constraints must be posted to assure that each class is assigned to just one suitable classroom. This requirement could be implemented via the `disjoint2` constraint, which ensures non-overlapping of a set of rectangles. In our case, the rectangle is defined by the start time variable (`Time`) and the duration (`Duration`) of each meeting, and by the classroom variable (`Classroom`) for the corresponding class:

```
disjoint2( [ rectangle(Time, Duration, Classroom, 1) | _ ] ) .
```

The number `1` represents the requirement of *one* classroom for each meeting.

A different type of propagation among the time variables is achieved via the `cumulative` constraint. It ensures that a resource can run several tasks in parallel, provided that the discrete resource capacity is not exceeded. If there are `N` tasks, each starting at a certain time (`StartI`), having a certain duration (`DurationI`) and consuming a certain amount of resource (`ResourceI`), then the sum of resource usage of all the tasks must not exceed resource limit (`ResourceLimit`) at any time:

```
cumulative([Start1,...,StartN], [Duration1,...,DurationN],
           [Resource1,...,ResourceN], ResourceLimit) .
```

The `cumulative` constraint helps to assign a classroom of sufficient size to each meeting while allowing smaller classes to be assigned to larger classrooms.

Example 3. Let us imagine a small example with 2 rooms for 40 students, 3 rooms for 20 students, and 1 room for 10 students. The set of `cumulative` constraints follows

```
cumulative(Time_meetings_with_size_40, Dur_40, ListOf1, 2),
cumulative(Time_meetings_with_size_20_40, Dur_20_40, ListOf1, 5),
cumulative(Time_all_meetings, Dur_all, ListOf1, 6).
```

The first constraint ensures that the largest classes are accommodated into the largest rooms, the second constraint allows medium-sized classes to be placed into rooms for 20 students, and also into rooms for 40 students if they are not already asked for by the first constraint. The third constraint allows movement of small classes between all rooms, subject to the condition that they are not occupied by any larger classes at the same time.

More precisely, we can post one `cumulative(Starts, Durations, ListOf1, Limit)` constraint for each possible size of classroom denoted by `Size`. The constant `ListOf1` denotes a list of 1 representing a unit resource requirement (one classroom) by each course. `Durations` represents the durations of classes with the start time Starts. Variables Starts and Limit should satisfy the following properties

$$\texttt{Starts} = \{\texttt{Start|meeting(Start,Duration,Capacity)} \wedge \texttt{Capacity} \geq \texttt{Size}\}$$
$$\texttt{Limit} = \mathrm{card}\{\texttt{Id|classroom(Id,Capacity)} \wedge \texttt{Capacity} \geq \texttt{Size}\}$$

Actually, it is sufficient to post this constraint only for some specific sizes of classrooms. Classrooms of similar size are grouped together to achieve better efficiency.

Another possibility for taking cumulative constraints into account consists of splitting classrooms into *groups* by size. Each class would be included in the group of corresponding size only. Such division can be useful if we do not want to put smaller classes into larger classrooms of other group at any time (e.g., smaller classes must be in the classrooms with a capacity smaller than 400 students).

4.3 Soft Constraints

Three types of soft constraints are currently handled by the system which will be discussed in this section:

1. unary constraints on time variables — faculty time preferences;
2. unary constraints on classroom variables — faculty preferences on the classroom selection for classes;
3. binary constraints for each joint enrollment between two classes.

Instructors may specify preferences for the days, hours, or parts of days they wish to encourage or discourage. This specification is transformed into a list of integer preferences corresponding to the possible start time of each class. We have seen that the initial selection of start times for each class is determined by its meeting pattern. The domain size of this set of start times can differ greatly among meeting patterns (it ranges from 5 to 50 possible values). This causes the relative effect of any given preference to vary greatly among the meeting patterns. To compensate for this effect, the number of preference points associated with instructor time preferences differ based on the meeting pattern.

Each class is associated with a time preference variable with preferences initialized either as specified by the instructor or to a set of default preferences. These default preferences are very important — their exclusion would lead to the construction of timetables which discriminate against classes for which no preferences have been provided. Many such classes would be placed in undesirable times, which no human timetabler would want to do.

Instructors may also specify positive or negative preferences towards the room selection for each class. It is possible to prefer or discourage particular classrooms, buildings, or properties of the room (e.g., "I prefer classrooms with a computer." or "I discourage classrooms without windows."). Each value in the domain of the classroom preference variable has either the specified preference or the neutral preference specification.

Any two classes potentially have a number of students who are enrolled to both at the same time. We seek to control their degree of overlap in the timetable by a generalization of the soft disjunctive constraint (see `soft_disjunctive` in Sec. 3.2). Such binary soft constraints include two time preference variables for corresponding classes, with the cost given by the number of students enrolled in both. Since each class may have several meetings, such a generalized disjunction needs to propagate preferences to the all values of the uninstantiated preference time variable which could be affected by the overlap of any of the meetings.

Preferences associated with the time preference variables are influenced by both the first and third soft constraints. Constraints of the first type initialize preferences. Constraints of the third type propagate (increase) them during the computation of the same cost function. The sum of the cost variables (see Sec. 3.1) for the time preference variables gives this cost function, i.e., the solution cost wrt. time assignment. As a consequence we need to balance the number of student joint enrollments from the third constraint with the number of preference points assigned by the first constraint, (e.g., the summarized preference points of the instructor for one class corresponds to overlapping for 20 students). The relationship between preference points and joint enrollments is specified as part of the input data.

4.4 Labeling

Separated Time & Classroom Variable Labeling. In our timetabling problem, we have time and classroom preference variables and two types of cost functions based on the sums of the corresponding cost variables. These cost functions are independent of each other and introduce two different criteria in our problem. Since minimizing student conflicts and accommodating the time preferences of classes were judged to be a more critical aspect of the problem than meeting preferences for classrooms, we have explored the following approach: labeling of time preference variables is processed first, followed by the labeling of room preference variables. Potential improvements of this search are mentioned in Sec. 7.

Initial & Repair Searches. The overall process of searching for a solution consists of two main parts — the initial search and the repair search. Results of a prior search are used only by the time preference variables. Information about an unassigned classroom variable reflects back upon the corresponding time preference variable as we describe in the next paragraph.

In the initial search, preferences associated with the time preference variables are set as they are defined by the problem. Values of the unassigned time preference variables that were unsuccessfully tried during the prior search are discouraged during the next search. There are other time preference variables for which a particular value is discouraged. These are the variables with a corresponding unassigned classroom preference variable in the former search. Since we have processed the assignment of the time variables first, such a non-assignment is the result of no classroom being available for the corresponding time. Successfully assigned values for all remaining time preference variables are encouraged in an automated repair search as it was proposed in item 1 in Sec. 1. In addition, variable ordering can be changed for unassigned time variables (item 2). The user can also introduce his own preferences (item 3) or relax the hard constraints (item 4).

Value & Variable Ordering. Different heuristics were used for time and classroom variable labeling. We have employed the first-fail heuristics to determine the ordering of classes for time assignment. It selects the most highly constrained time preference variables with respect to both hard and soft constraints. First, we select among the variables having the smallest domain. Ties are broken based on the greatest number of soft constraints related to this variable. If not selected early, the domain of such a variable may become too small to select a sufficiently preferred value, or it may even become empty and cause a backtracking. Early propagation of soft constraints is also encouraged, so as not to discover mistakes too late. A specific class time assignment was selected among the most preferred values, i.e., we have chosen an optimistic approach for the value selection. This approach lead us to a good first solution from the point of view of cost function over time variables.

The first-fail approach was also used to choose a class to be placed into a classroom. The best results of the cost function over classroom variables were achieved by the following value ordering heuristics. First, the most preferred classroom was selected. Ties were broken by the selection of the smallest available classroom so as not to waste available resources. A branch and bound search was applied to improve the cost function over room variables.

During the repair searches, we have updated our variable ordering heuristics as it is pointed in Section 3.3 about the proposed search procedure, i.e., each variable unassigned in the last search was assigned prior to successfully assigned variables in the consequent search (see item 2).

5 Computational Results

Our data set from fall semester 2001 includes 747 classes to be placed into 41 classrooms. The classes included represent 81,328 course requirements for 28,994 students. The results presented here were computed by SICStus Prolog 3.9.0 on a PC with AMD Athlon/850 MHz processor and with 256 MB of memory.

Table 2 shows the computational results for the initial search and the subsequent automated repair searches. *Unassigned classes* refers to the number of classes with either time or classroom variables that were not assigned during labeling. *Satisfied time* gives the percentage of how many encouraged times for classes were selected. *Unsatisfied time* refers to the percentage of the discouraged times for classes which were selected. *Student conflicts* estimates the percentage of unsatisfied requirements for courses by students. *Preferred classrooms* measures the percentage of classes for which encouraged classrooms were selected. The current data set does not include any preferential requirements which discouraged specific classrooms.

Table 2. Results of the initial search and the automated repair searches

Run	initial	repair I.	repair II.	repair III.	repair IV.
Unassigned classes	19	15	10	5	3
Satisfied time (%)	83.6	81.1	79.4	79.9	79.7
Unsatisfied time (%)	4.1	4.3	4.0	4.0	4.0
Student conflicts (%)	1.6	1.7	1.9	1.9	1.9
Preferred classrooms (%)	77.6	49.7	49.0	48.3	49.0

The initial search procedure took about 2-3 minutes for the time labeling, one step of the branch and bound search for classroom variables took 1-2 seconds. The time labeling takes a longer time due to the preference propagation and more complex constraints (e.g., `cumulative`) posted on time variables. The length of the overall run depends on the number of repair steps, computation of the best solution took about 15 minutes (one initial step followed by four repair searches).

Originally we started to solve the problem using a built-in backtracking search algorithm with a variety of variable and value ordering heuristics. This attempt did not lead to any solution after 10 hours of computations however. Too many failed computations were repeated exploring parts of search tree with no solution.

We can see that most of preferential requirements of instructors were satisfied during assignment of time variables. The unsatisfied time percentage mostly illustrates that a number of classes must be taught at unpopular times due to the limited number of rooms available. Let us also note that these results include default preferences for classes with no preferred times. The automated repair search (see item 1 in Sec. 1) during time assignment was able to improve on the initial solution substantially. The solution continued to improve through four repair searches. Additional steps did not improve the quality of the solution further.

One of the more important lessons learned here is that the `disjoint2` and `cumulative` constraints must be used together in a redundant manner to find an acceptable solution (for description of both constraints see Sec. 4.2). The `cumulative` constraints were able to introduce additional constraint propagation for the time variables by informing them of the available room resources. Neither `cumulative` nor `disjoint2` constraints alone were able to find an acceptable solution. Results using only the `cumulative` constraints left about 20 unassigned classes. Using only the `disjoint2` constraint resulted in 50 unassigned classes.

6 Related Work

Purdue University. Currently the timetable for Purdue University is constructed by a manual process. An earlier approach examined for automating construction of the Purdue University timetable modeled the room-time assignment problem as a multiple choice quadratic vertex packing and utilized a tabu search algorithm [14]. This approach was further developed into a prototype system used to create a schedule for large lecture classes in spring 1994, but was never adopted by university schedulers due to inadequacies in the way it handled instructor time preferences and student conflicts.

Constraint Programming. There have not been many attempts [18, 3] to apply constraint programming to the solution of demand-driven timetabling problems where it is not possible to satisfy all requests of students. Such over-constrained problems require enhancements to the classical constraint satisfaction approach. Once these are developed, we can apply all of the advantages of constraint programming, including a declarative description of the problem together with strong propagation techniques.

We have previously constructed a demand-driven schedule for the Faculty of Informatics at Masaryk University [18] having 270 classes and about 1250 students. Conflicts of students between classes were controlled using a similar cost function as in our current approach. Constraint logic programming allowed implementation of a variety of constraints available in ILOG Scheduler [15]. Soft constraints were implemented with the help of special variable and value ordering heuristics defined by the preferences of particular variables in the constraints. The timetable constructed was able to satisfy 94 % of the demands of students and more than 90 % of the preferential requirements of teachers. Unfortunately, it was not easy to extend this implementation to larger problems due to the bound consistency algorithms in ILOG Scheduler. Since these algorithms only propagate changes over the bounds of the domain variables, both constraint and preference propagations were much weaker than there are now.

A solution of the section assignment sub-problem is included in Banks, van Veek, and Meisels [3] via iterative addition of constraints into a CSP representation. Inconsistent constraints are not included in the final CSP representation, which allows solution of an over-constrained problem. This implementation, including its own constraint satisfaction solver, was verified using random timetabling problems based on problems from high schools in Edmonton, Alberta, Canada. The largest data set included requirements of 2,000 students and 200 courses. They were able to satisfy 98 % of student demand on more than half of the experiments. The solution presented is influenced by a special set of times that must be assigned to each course. It conforms well to high schools, but is rather different from the situation in university course timetabling. University class meeting patterns are not as strict, which results in a problem with a variety of additional requirements and preferences.

Other Approaches. The comprehensive university timetabling system described by Carter [6] is characterized by problem decomposition with respect to both type and size of final sub-problems. They were able to solve the problem for 20,000 students and 3,000 course sections. The system was used for 15 years at the University of Waterloo.

Aubin&Ferland [2] propose an iterative heuristic method to solve the problem which alternately assigns times and students to course sections until no further improvements to the solution can be found. The system was tested on data from a High School in Montreal including demands of 3,300 students and 1,000 courses.

Robert&Hertz [16] decompose the problem into a series of easier sub-problems corresponding to time, section, and classroom assignments and solve them via tabu search methods. The method presented is able to generate an initial solution which can be incrementally improved after problem redefinition (negotiation on initial constraints with teachers and students). The initial solution for about 500 students and 340 course sections satisfied approximately 10 % of the student requirements and 80 % of the preferential requirements of teachers.

The local search heuristic procedure of Sampson, Freeland, and Elliot [19] solves a problem having a smaller solution space with 89 course sections and 230 students. They were able to meet 94 % of the student scheduling requirements at the Graduate School of Business Administration at the University of Virginia.

7 Conclusion

We have proposed and implemented a solution to a large scale university timetabling problem. We have constructed a demand-driven schedule which is able to reflect diverse requirements of students during course enrollment. Our solution is able to satisfy the course requests of 98 % of students. About 80 % of preferential requirements on time variables were also met with only a small number of classes taught at discouraged times (about 4 %). The automated search was able to find suitable times and classrooms for 744 classes. The remaining 3 classes should be possible to assign with user input.

Our proposal included a new solver for soft constraints which is of particular interest for timetabling problems where the costs in the problem are directly related to the present values for time and room assignments of classes. We have proposed a special repair search algorithm which allows us to find a solution to the problem (even when it is over-constrained). We have also discussed a special set of `cumulative` constraints which, together with the `disjoint2` constraint, processes stronger constraint propagation.

Our future research will include an extension of the problem solution together with improvements to the preference solver that has been described. Also, our approach must be validated using data sets from other semesters.

Currently we are working on the inclusion of new variable ordering heuristics which will process an interleaved labeling for both time and classroom variables. We intend to process an earlier assignment of classroom variables for which late labeling proved to be difficult or impossible. Other heuristics will be aimed at improving the solution with the help of a pattern matching mechanism based on the sets of meeting patterns in the problem.

We would like to do an extensive study of the proposed repair search algorithm and its possible application to a general problem solution. We also intend to explore how it may be extended to the solution of the minimal perturbation problem, which is very important for all timetablers. Once timetables are published they require many changes based on additional input. These changes should be incorporated into the problem solution with minimal impact on any previously generated solution.

Purdue University currently relies on a completely manual process for constructing its timetable. A detailed comparison of results between the approach described in this paper and the manual process for the full large lecture problem is one of the next steps in our work.

A new approach for making initial student section assignments for courses with multiple sections will also be examined in future work. This is required to construct a joint enrollment matrix that is more representative of the probable cost of having two classes overlap. An approach such as the homogeneous sectioning used by Carter [6] is attractive since it tends to result in fewer classes having joint enrollments with others. This simplifies the task of finding non-conflicting assignments and appears to be an accurate representation when a final sectioning process will take place after construction of the timetable.

We feel that the solution methods used for the large lecture problem should be directly applicable to construction of the 74 academic unit timetables. Some solution refinements may be necessary to simplify time assignments for introductory courses with large numbers of sections. Additional system architecture work will also be necessary to allow unit timetablers to use local preference files, and to work cooperatively if there is a high degree of interrelationship between classes offered by the units.

Acknowledgements

This work is partially supported by the Grant Agency of Czech Republic under the contract 201/01/0942 and by Purdue University.

We would like to thank our students who are assisting with the solution of this problem, and Purdue staff who have helped in many ways.

References

[1] Slim Abdennadher and Michael Marte. University course timetabling using constraint handling rules. *Journal of Applied Artificial Intelligence*, 14(4):311–326, 2000.

[2] Jean Aubin and Jacques A. Ferland. A large scale timetabling problem. *Computers & Operations Research*, 16(1):67–77, 1989.

[3] Don Banks, Peter van Beek, and Amnon Meisels. A heuristic incremental modeling approach to course timetabling. In *Proceedings of Artificial Intelligence'98, Canada*, 1998.

[4] Stefano Bistarelli, Ugo Montanari, and Francesca Rossi. Semiring-based constraint solving and optimization. *Journal of ACM*, 44(2):201–236, March 1997.

[5] Mats Carlsson, Greger Ottosson, and Björn Carlson. An open-ended finite domain constraint solver. In *Programming Languages: Implementations, Logics, and Programming*. Springer-Verlag LNCS 1292, 1997.

[6] Michael W. Carter. A comprihensive course timetabling and student scheduling system at the University of Waterloo. In Edmund Burke and Wilhelm Erben, editors, *Practice and Theory of Automated Timetabling III*, pages 64–82. Springer-Verlag LNCS 2079, 2001.

[7] Michael W. Carter and Gilbert Laporte. Resent developments in practical course timetabling. In Edmund Burke and Michael Carter, editors, *Practice and Theory of Automated Timetabling II*, pages 3–19. Springer-Verlag LNCS 1408, 1998.

[8] Eugene C. Freuder and Richard J. Wallace. Partial constraint satisfaction. *Artificial Intelligence*, 58:21–70, 1992.

[9] Thom Frühwirth. Constraint handling rules. In Andreas Podelski, editor, *Constraint Programming: Basics and Trends*, LNCS 910. Springer, 1995. (Châtillon-sur-Seine Spring School, France, May 1994).

[10] Hans-Joachim Goltz, Georg Küchler, and Dirk Matzke. Constraint-based timetabling for universities. In *Proceedings INAP'98, 11th International Conference on Applications of Prolog*, pages 75–80, 1998.

[11] Christelle Guéret, Narendra Jussien, Patrice Boizumault, and Christian Prins. Building university timetables using constraint logic programming. In Edmund Burke and Peter Ross, editors, *Practice and Theory of Automated Timetabling*, pages 130–145. Springer-Verlag LNCS 1153, 1996.

[12] Martin Henz and Jörg Würtz. Using Oz for college timetabling. In Edmund Burke and Peter Ross, editors, *Practice and Theory of Automated Timetabling*, pages 162–177. Springer-Verlag LNCS 1153, 1996.

[13] Joxan Jaffar and Michael J. Maher. Constraint logic programming: A survey. *Journal of Logic Programming*, 19:503–581, 1994.

[14] Edward Mooney. *Tabu Search Heuristics for Resource Scheduling*. PhD thesis, Purdue University, 1991.

[15] Claude Le Pape. Implementation of resource constraints in ILOG SCHEDULE: a library for the development of constraint-based scheduling systems. *Intelligent Systems Engineering*, 3(2):55–66, 1994.

[16] Vincent Robert and Alain Hertz. How to decompose constrained course scheduling problems into easier assignment type subproblems. In Edmund Burke and Peter Ross, editors, *Practice and Theory of Automated Timetabling*, pages 364–373. Springer-Verlag LNCS 1153, 1996.

[17] Hana Rudová. Soft scheduling. In *Proceedings of the 2001 ERCIM Workshop on Constraints*. Charles University, Faculty of Mathematics and Physics, 2001. See http://arXiv.org/html/cs.PL/0110012.

[18] Hana Rudová and Luděk Matyska. Constraint-based timetabling with student schedules. In Edmund Burke and Wilhelm Erben, editors, *PATAT 2000 — Proceedings of the 3rd international conference on the Practice And Theory of Automated Timetabling*, pages 109–123, 2000.

[19] Scott E. Sampson, James R. Freeland, and Elliot N. Weiss. Class scheduling to maximize participant satisfaction. *Interfaces*, 25(3):30–41, 1995.

[20] Andrea Schaerf. A survey of automated timetabling. Technical Report CS-R9567, CWI, Amsterdam, NL, 1995.

[21] Pascal Van Hentenryck. *Constraint Satisfaction in Logic Programming*. MIT Press, 1989.