Complex university course timetabling

Article *in* Journal of Scheduling · April 2011

DOI: 10.1007/s10951-010-0171-3 · Source: DBLP

3 authors, including:

Hana Rudová
Masaryk University
86 PUBLICATIONS **909** CITATIONS
SEE PROFILE

Tomáš Müller
Purdue University
26 PUBLICATIONS **429** CITATIONS
SEE PROFILE

Complex University Course Timetabling

Hana Rudová, Tomáš Müller and Keith Murray

May 2010

Abstract This paper summarizes the work done to solve a complex course timetabling problem at a large university and provides new insights into the overall timetabling process. The first step in the successful solution of this problem was to define a course structure model that allowed application of classical course timetabling methods. Several methods were necessary to solve the complete problem. First, support procedures were needed to detect and correct an infeasible problem where hard constraints were being violated. The resulting timetabling problem was then solved via a search for a complete assignment of times and rooms to classes, taking all hard and soft constraints into account. Methods were also developed for modifying a computed solution in response to changes introduced at a later time while having a minimal impact on existing assignments.

These problems are described formally using a weighted constraint satisfaction model of the timetabling problem and solutions are proposed through two types of algorithms: (1) generic iterative forward search with conflict-based statistics, and (2) branch and bound. Experimental results from the course timetabling system developed for Purdue University are provided. These include solutions computed by university schedulers using the system for two separate terms along with extensive experiments solving two central and six departmental problems, individually, and as a combined problem with almost 2,500 classes. The system described is currently used on all of the many varied course timetabling problems encountered each term at Purdue University.

Keywords University course timetabling · Search · Constraint satisfaction · Soft constraints · Dynamic problems · Heuristic

H. Rudová
Faculty of Informatics, Masaryk University
Botanická 68a, Brno 602 00, Czech Republic
Tel.: +420-549 496 345
Fax: +420-549 491 820
E-mail: hanka@fi.muni.cz

T. Müller and K. Murray
Space Management and Academic Scheduling, Purdue University
400 Centennial Mall Drive, West Lafayette, IN 47907-2016, USA
Tel.: +1-765-494-3911
Fax: +1-765-494-6480
E-mail: muller@unitime.org, kmurray@purdue.edu

1 Introduction

University course timetabling is the problem of assigning courses to a limited number of time periods and classrooms subject to a variety of hard constraints, with the quality of feasible solutions evaluated based on violations of soft constraints (Schaerf, 1999). Two representative instances of the problem are curriculum based course timetabling and post enrollment course timetabling, both of which were included in the International Timetabling Competition 2007 (McCollum et al., 2009; Lewis et al., 2007; Di Gaspero et al., 2007). In curriculum based timetabling, conflicts between courses are determined by the curricula published by the University. Conflicts in post enrollment timetabling are established directly by students who individually enroll into particular courses. Both types of problems have been frequently solved in the past, as evidenced by many surveys (Lewis, 2008b; Burke and Petrovic, 2002; Schaerf, 1999; Carter and Laporte, 1998).

The timetabling problem considered in this paper generally belongs to the class of post enrollment problems since it incorporates individual student course requests. Student requests have been addressed in earlier works (Hertz, 1991;

The final publication is available at Springer via http://dx.doi.org/10.1007/s10951-010-0171-3

Sampson et al., 1995; Robert and Hertz, 1996; Beyrouthy et al., 2007) where student sectioning (assignment of students into several sections of a course to allow for a desired number of students in each section) is considered as part of the solution of the overall timetabling problem. Similar problems incorporating student sectioning have also been solved on the high school level (Aubin and Ferland, 1989; Banks et al., 1998). It is notable that many later works have not considered student sectioning at all. This allows the complexity of the problem to decrease, but at the cost of being less relevant to complex real-world problems.

A basic form of the post enrollment problem, not including student sectioning, was solved by participants in the International Timetabling Competition 2002 and discussed in many follow-up works (Kostuch, 2005; Chiarandini et al., 2006; Di Gaspero and Schaerf, 2006). An extended version of this problem was considered in the track on post enrollment course timetabling at the 2007 competition (Lewis et al., 2007). The winner of the track (Cambazard et al., 2008) applied a local search algorithm, as did Mayer et al. (2008). Approaches using a combination of constructive and local search methods were applied by several other entries (Chiarandini et al., 2008; Lewis, 2008a; Müller, 2008). Deserving particular note are the solution methods described by Müller (2008), which extend the methodology and principles of the work described here. Müller was able to solve problems from all three of the competition tracks with the same solution methodology and succeeded as a finalist in all tracks and winner of two tracks.

The generality of the solution methodology is crucial because solving a complex university-wide timetabling problem requires considering aspects found in many different timetabling problems. The aim of the work presented here is to extend beyond the solution methods used in school/faculty or departmental problems (Causmaecker et al., 2009; Schimmelpfeng and Helberg, 2007; Di Gaspero and Schaerf, 2006; Avella and Vasil'ev, 2005; Hertz, 1991) and consider solution approaches for university-wide problems similar to that of Carter (2001). As pointed out by McCollum (2007), many works in the area of university timetabling concentrate on solving simplified problems or artificial data sets, but these have rarely been extended to the solution of actual university problems of any large scale.

The major differences between many of the problems studied and their real-life counterparts are the additional complexity imposed by course structures, the variety of constraints imposed, and the distributed responsibility for information needed to solve such problems at a university-wide level. University timetabling problems may also involve the solution of multiple subproblems with very different characteristics. In practice, therefore, the solution process should not be specifically tailored to a single problem type (Burke and Petrovic, 2002).

Another very important aspect of the solution methodology is the ability to consider changes in the problem solution (Müller et al., 2005; Piechowiak et al., 2005). In practical applications it is nearly inevitable that some circumstance will occur that necessitates a change to one or more assignments made to classes in the original solution. The question then becomes how to make the desired adjustments to this carefully constructed timetable while causing the least disruption to other assignments and the people who have made plans based on them.

Dynamic changes in the context of timetabling problems expressed as Resource Constraint Project Scheduling Problems (Brucker and Knust, 2001) were studied by Elkhyari et al. (2003). Problem changes along with uncertainty were surveyed by Brown and Miguel (2006), where course timetabling is used as a base example. Cases where interactive timetabling is needed to handle dynamic aspects of the problem were discussed by Cambazard et al. (2005), Piechowiak et al. (2005) and by Müller and Barták (2002). Dynamic problems appear frequently in real-life planning and scheduling applications (Ouelhadj et al., 2009; Verfaillie and Jussien, 2005). Sakkout and Wallace (2000) studied problems of minimal reconfiguration of schedules in response to a changing environment which is closely related to the timetabling problems considered here.

These ideas form the primary motivation of this work, which is to create course timetables for a large university such as Purdue, including all of its courses and students. Here, there is a need to create and modify timetables that better meet student course demand and allow students to be assigned to the constituent course sections in a way that minimizes conflicts. Purdue University is a large (39,000 students) public university with a broad spectrum of programs at the undergraduate and graduate levels. In a typical term there are 9,000 classes offered using 570 teaching spaces. Approximately 259,000 individual student class requests must be satisfied. The complete university timetabling problem is decomposed into a series of subproblems to be solved at the academic department level, where the resources required to provide instruction are controlled. Several other special problems, where shared resources or student interactions are of critical importance, are solved institution-wide. Since the initial timetable is produced several months before the actual start of the semester, it is also necessary to accommodate appropriate modifications to the published solution.

The first steps toward solving this problem focused on the critical large lecture problem which must be solved at the university-wide level. A constraint model was formulated by Rudová and Murray (2003) and solved using constraint logic programming with soft constraints. Later work (Müller

http://www.idsia.ch/Files/ttcomp2002/

et al., 2005) applied the iterative forward search algorithm (Müller, 2005) to solve the large lecture problem. This approach also allowed the inclusion of dynamic aspects introduced by changes of the underlying course timetabling problem over time. The university-wide problem was initially described and solved by Murray et al. (2007) and work on course sectioning was presented by Müller and Murray (2008). Murray et al. (2007) also proposed transformation of the complex course structure into a set of classes related by specific constraints. As a consequence, solution methodologies based on the classical timetabling model with classes could be applied.

The contribution made by this paper is to integrate many of the concepts introduced in this earlier work and expand upon it in several important areas. The first goal is to extend the set of departmental problems introduced by Murray et al. (2007) and rigorously describe the characteristics of each problem. The combined problem, including data from all of the original problems, is then introduced to present scalability of the proposed approach. Another goal is to provide more formal descriptions of the basic feasibility problem, classical initial problem, minimal perturbation problem, and the interactive problem. The constraint model, including domain variables and essential problem constraints is then formulated in greater detail. This paper also presents a more precise description of the iterative forward search algorithm based on a generic specification of solution evaluation and a comparator which allows its application for solving many of these problems. An algorithm for interactively finding solutions for problem changes based on branch and bound is then presented in detail. Finally, results of extensive experiments for two different types of central problems, six types of departmental problems, and the large combined problem with almost 2,500 classes are presented and results of human schedule managers using the system are reported.

2 Problem Description

The Purdue University timetabling problem is naturally decomposed into the following problems:

- a centrally timetabled large lecture problem (almost 900 classes timetabled into 55 rooms with up to 474 seats),
- individually timetabled departmental problems (about 70 problems with 10 to 500 classes using departmental laboratory spaces and centrally managed classrooms allocated to departments based on expected class hours),
- and a centrally timetabled computer laboratory problem (about 200 classes timetabled into 31 rooms with 20 to 45 seats).

Each of these sub-problems may have characteristics that are different from other problems in the set. Some of the different attributes for selected problems are listed in Table 1, all data sets for these problems are available from http://www.unitime.org.

Table 1 Main characteristics of the considered problems.

Problem	Classes	Meetings per class	Hours per class	Classes per subpart	Students	Classes per students	Timetabled classes per student	Rooms	Room capacity (min-max)	Frequency (in %) (used slots)	Utilization (in %)	Distribution constraints per class
pu-spr07-llr	803	2.09	2.40	1.25	27881	3.15	0.00	55	40-474	67.77	62.54	0.69
pu-fal07-llr	891	2.07	2.32	1.26	30855	3.23	0.00	55	40-474	74.63	70.40	0.71
pu-spr07-ms	440	2.32	2.43	3.52	11992	1.11	3.13	25	24-51	84.43	76.18	2.74
pu-fal07-ms	525	2.35	2.40	4.45	14331	1.10	3.18	33	24-61	78.30	67.88	2.18
pu-spr07-cs	93	1.63	2.14	1.82	725	2.03	3.19	13	17-61	36.18	30.42	2.83
pu-fal07-cs	174	1.31	1.92	2.72	2002	1.57	4.00	13	22-61	52.19	44.93	2.49
pu-spr07-ecet	107	1.17	2.57	2.55	648	2.30	3.15	9	16-49	60.33	39.95	2.78
pu-fal07-ecet	113	1.21	2.54	2.46	662	2.60	3.14	9	16-49	63.33	50.20	2.54
pu-spr07-sa	69	1.67	2.30	1.50	1312	1.40	2.71	4	40-48	72.00	54.46	1.25
pu-fal07-sa	63	2.00	2.43	1.47	925	1.13	2.62	6	14-48	50.67	52.24	0.92
pu-spr07-cfs	214	1.44	2.91	2.21	1610	1.94	2.94	29	10-71	41.52	26.70	1.79
pu-fal07-cfs	201	1.38	2.90	2.14	1936	1.75	3.17	28	10-71	39.73	23.14	3.28
pu-spr07-vpa	249	1.71	3.24	1.64	1836	2.17	2.47	47	10-45	34.34	28.80	2.06
pu-fal07-vpa	290	1.59	2.92	1.72	1747	2.22	2.42	41	10-45	40.39	32.81	1.26
pu-spr07-lab	443	1.25	1.97	4.82	8421	1.14	4.20	36	20-45	52.57	43.27	2.05
pu-fal07-lab	200	1.20	1.81	3.70	4835	1.08	4.49	31	20-45	27.19	23.21	3.29
pu-spr07-c8	2418	1.81	2.45	1.95	29514	4.16	0.00	213	10-474	55.27	56.35	1.76
pu-fal07-c8	2457	1.85	2.40	1.90	32399	4.10	0.00	208	10-474	56.83	61.29	1.74

Problems with the suffix llr and lab stand for the large lecture problem and the computer laboratory problem, respectively. The suffixes ms, cs, ecet, sa, cfs, and vpa denote data from departmental problems. The abbreviations fal07 and spr07 indicate the Fall 2007 and Spring 2007 semester. In addition to these problems, a combined data instance has been created containing classes from all of these problems (suffix c8). This problem contains nearly 2,500 classes with more than 32,000 students and 200 rooms. Even though this problem is the largest, the large lecture problem still remains of special importance since it covers the hardest problem components. As can be seen in Table 2, the number of slots used (frequency) in the pu-fal07-llr problem increases with the room capacity (frequency and utilization are defined on page 4 when describing problem features in Table 1). 85.63% of slots are used for the 23 largest rooms and there are no free slots in the 4 largest rooms.

Table 2 Characteristics for regular times in pu-fal07-llr problem.

Frequency (used slots) in %	74.63	85.63	100.00
Utilization in %	70.40	74.46	77.97
Rooms	55	23	4
Room capacity (min-max)	40-474	100-474	445-474

The problems presented in Table 1 are related to each other. The large lecture problem is solved first. It includes classes from all departments requiring students to be scheduled into the pool of shared large lecture rooms. Consequently, departmental problems can be solved taking into account their large classes assigned earlier. Finally, the shared computer laboratory problem is solved. Since this is an easy problem (many computer labs have similar equipment and capacity, and there are less restrictions on time) it can be solved after the departmental problems.

Table 1 gives the main characteristics of each problem, including the number of *classes*, the average number of *meetings per week*, and the average number of *hours per class*. With few exceptions, 1 hour $\times$ 3 day per week classes meet on Monday, Wednesday, and Friday at the half hour (7:30 am, 8:30 am, ... 4:30 pm). 1.5 hour $\times$ 2 day per week classes meet on Tuesday and Thursday during set time blocks. 2 or 3 hours $\times$ 1 day per week classes must also fit within specific blocks, etc. Generally, all meetings of a class should be taught at the same time of day in the same location with the same instructor. The given structure of start times (denoted as *time patterns*) generally leads to 20 time slots per day of 30 minute duration (referred to as *regular times*). However, due to many irregular starting times and class durations, the basic time slot for the solver is 5 minutes. Also, the standard time horizon of 5 days from 7:30 am to 5:30 pm is extended to a full 7 day week split into 5 minutes slots ($7 \times 24 \times 12$ in total). As will be described in Section 3, classes are grouped into subparts (see the average *classes per subpart*) which requires special constraints to be considered.

Other important characteristics given are the number of *students* and the classes to which these students are enrolled. Students can be enrolled in classes associated with the given problem (see the average *classes per students*) or in classes belonging to previously solved problems (see average *timetabled classes per student*). This means that the timetabling process must take into account classes that are already timetabled and consider them as classes with a fixed time for the student. Typically, students in departmental problems have some classes that have already been timetabled in the large lecture problem. Also, students in the computer laboratory problem have classes previously timetabled in both the large lecture and departmental problems (the number of timetabled classes per student is highest for pu-spr07-lab and pu-fal07-lab problems). The large lecture (pu-spr07-llr and pu-fal07-llr) and combined problems (pu-spr07-c8 and pu-fal07-c8) do not list any timetabled classes since they do not use output from other problems.

Each problem also has a specified set of *rooms* where classes should be scheduled (most of the classes cannot share their room and use it as a unary resource) and each class has a specified number of seats. Table 1 includes information about minimum and maximum *room capacity*. As previously mentioned, two important characteristics of timetabling problems are *frequency* and *utilization* (Beyrouthy et al., 2009). While frequency denotes the ratio of the number of time periods used to those available (over all classrooms), utilization considers the ratio of used to available seats (in all classrooms over all time periods considered). Both frequency and utilization are presented based on regular times, since the number of classes taught at other times (e.g., evening or Saturday classes) is negligible.

The main types of constraints considered in the problems are defined and summarized in Table 3. A consistent scale (required, strongly preferred, preferred, neutral, discouraged, strongly discouraged, prohibited) is established for setting all of these constraints. Required and prohibited indicate hard constraints while other classifications indicate soft constraints which do not necessarily need to be fulfilled. Each class has specified time patterns that are allowed and preferences on the suitability of particular times and rooms assigned. Many classes have specified instructors, in which case classes with the same instructor cannot be taught at the same time. Typically, classes are not permitted to share the same room with other classes at the same time. When two classes have students in common, it is preferable that they are taught with no overlap. Table 1 specifies the average number of *distribution constraints per class*. The base list of available distribution constraints is shown in Table 3. These constraints define relationships between classes. For example, it may be desirable to schedule several classes such that a fixed amount of time is present between each pair of classes (typically, this may be used to indicate that classes must follow each other, i.e., back-to-back classes) or they must respect specific precedences (e.g., laboratories after the lecture). Sometimes it is desirable to place a set of classes, such as laboratories of the same course, during a defined set of time periods (e.g., during mornings or around lunch). Generally, several associated classes are placed in similar predefined times. Last but not least, there is a generic constraint which specifies days, times, or rooms to be (preferably) shared or to be avoided by the set of classes.

Table 3 Basic set of constraints.

		Hard constraint	Soft constraint
Times for class	Time pattern	x	
	Individual times	x	x
Rooms for class	Individual buildings/rooms	x	x
	Individual room equipment	x	x
Resource constraints	Room	x	
	Instructor	x	
Students	Conflicts between two classes		x
Distribution constraints between classes	Time between classes	x	x
	Time precedences between classes	x	x
	Classes placed in similar times	x	x
	Same or different meeting days/times/rooms for classes	x	x

The timetable maps classes into rooms and times under the above described hard and soft constraints. Satisfaction of soft constraints is projected to four basic criteria. The *student enrollments* criterion maximizes the number of student course enrollments that are satisfied. The *time preferences* criterion maximizes satisfaction of soft constraints on the time assignment of classes. The *room preferences* criterion maximizes the satisfaction of soft constraints on the room assignment of classes, and *distribution preferences* maximizes satisfaction of soft distribution constraints.

This section has described various features of the Purdue University problems. Many others have not been mentioned in detail to keep this paper to a reasonable length and these will be omitted from further discussion. A number are noted briefly here to demonstrate the ability of the solver to encompass constraints or features desired by the various departments using the system. Each is implemented as an extension of the main model and the solution procedures described in this paper. Distances between rooms are considered in order to penalize class placements that require students or instructors to travel large distances between consecutive classes (dissatisfaction of these soft and hard constraints is actually included in the student enrollment criterion since it means that certain student enrollments cannot be satisfied). It has already been mentioned that some classes should be scheduled to irregular start times and can have non-standard durations. Even more, the solver must handle specific date patterns for some classes, e.g., classes taught on odd or even weeks or classes taught during a given subset of weeks during the semester. In order to facilitate construction of timetables satisfactory to multiple departments in centrally managed problems, a soft constraint balancing the use of times among all departments is applied. This constraint allows each department to be allocated an approximately uniform distribution of times. There is also a set of soft constraints that seek to ensure efficient use of resources by discouraging use of much larger rooms than required or time placements that leave gaps in the schedule that are inconsistent with the standard time patterns used by most classes. A very important problem is the sectioning of students among classes associated with large courses, which may have more than 2,000 students (see page 1 for discussion on student sectioning). Generally, the current solution works with an initial sectioning which is further improved after the timetable is generated. Additional details about this extensive process are provided by Müller and Murray (2008). For more complete information on other aspects of these problems and their solution, please refer to `http://www.unitime.org`.

3 Course Model

A major obstacle to solving many actual university course timetabling problems is the complexity of the course structures found in these instances. The most common formulations of the problem are based on timetabling independent classes (Carter and Laporte, 1998; Schaerf, 1999; Petrovic and Burke, 2004). The addition of constraints necessary to address courses with multiple forms of instruction and different meeting requirements has typically resulted in highly specific problem models that are not widely applicable in other settings (McCollum, 2007). In large-scale problems, there may be many different sets of course structures to contend with. Without a more general framework for dealing with these, a workable approach to solving real problems is difficult to achieve.

Representing the many course structures found at a large university has necessitated the creation of a more general model that can be used to translate all parts of the course, and the relationships between them, into a set of classes and an extended set of constraints. These constraints generally fall into the set summarized in Table 3 and their realization is the same as for these basic constraints. Translation of the course structure into a set of classes related by specific constraints allows application of classical solution approaches for the timetabling of classes (though university course timetabling is the common term used in the literature (Burke and Petrovic, 2002; Schaerf, 1999; Carter and Laporte, 1998)).

The model developed for this work breaks each course down into as many as four levels to reflect the relationships among all of the classes that constitute it. While in a simple lecture course the class and the instructional offering are one and the same, for a large course there may be tens or hundreds of classes associated with a single instructional offering (e.g., a language or mathematics course offered to the whole university). A diagrammatic representation of the course structure model is shown in Figure 1.

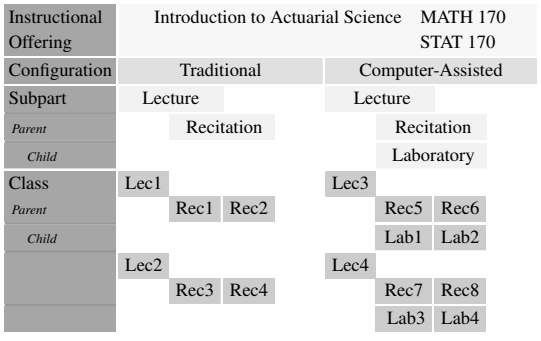

Fig. 1 Model of course structure.

For clarity, the term *instructional offering* is used in this model to distinguish the highest level of the structure from *course*, which is the more usual term for a series of lessons containing the subject matter to be taught. This accommodates the practice of cross-listing courses (i.e., the same subject matter is taught under more than one course title).

Many courses (i.e., instructional offerings in this model) have multiple subparts, such as tutorials or laboratories, that are associated with a parent lecture. In some cases, the course may even be offered with different configurations of these subparts. One instructor, for example, may teach the course material entirely in a lecture format whereas another may use a lecture and tutorial. The *configuration* level is included to account for these types of situations.

The *subpart* level accounts for multiple forms of instruction and/or different meeting requirements. If it is important for one group of students to be together in two or more different subparts of a course, a parent–child relationship may be included between classes at this level. If a parent–child relationship is established between two subparts, all students in a class belonging to the child subpart must also be sectioned to the appropriate parent class. Constraints are generated prohibiting an overlap in time between parent and child classes. Any attributes or preferences that apply to all classes within a subpart can be set at this level.

Timetabling takes place at the *class* level. There will typically be multiple classes associated with each subpart, especially when tutorial or laboratory sections are involved. Each class inherits attributes and preferences set on the subpart level, or these may be set for an individual class. Attributes or preferences set at the class level will override those set at higher levels. Each class must indicate the amount of time it meets, desired meeting pattern, weeks the class should meet during the term, and facility needs. Specific time, room, and room equipment preferences or requirements may also be set. If an instructor is entered, preferences may also be inherited from those set on the instructor.

An illustration of how the complete course structure is displayed by the user interface is shown in Figure 2. Here the configuration is displayed in the gray shaded area and the individual classes to be timetabled are listed below.

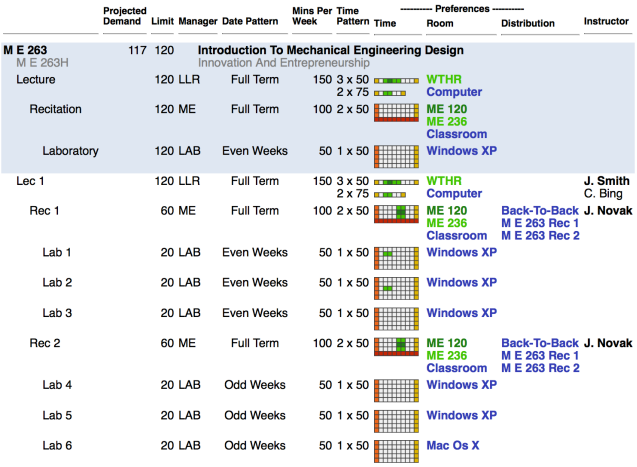

Fig. 2 Course structure with classes as displayed in user interface.

4 Formal Problem Specification

Constraint satisfaction (Dechter, 2003) forms the basis of the solution procedures used in this work. Constraint programming has often been applied to solve course timetabling problems. Guéret et al. (1996) and Lajos (1996) applied constraint logic programming as was done in the initial approach used for Purdue University timetabling (Rudová and Murray, 2003). The constraint language Oz was applied by Henz and Würtz (1996) and constraint handling rules were used to implement cost propagation by Abdennadher and Marte (2000).

There has also been significant interest in hybridization of constraint programming with other methods, like tabu search (White and Zhang, 1998) and large neighborhood search schemes (Cambazard et al., 2008). A hybrid approach has also been applied here. Specifically, a constraint satisfaction model of the problem is used along with base consistency techniques in the combined constructive and repair scheme of the iterative forward search algorithm.

Weighted constraints, similar to those considered in work by Shapiro and Haralick (1981), and further studied by many others such as Larrosa and Schiex (2004), Bistarelli et al. (1997), Schiex et al. (1995), and Freuder and Wallace (1992) are applied in this work. For this paper, the *weighted constraint satisfaction problem* $P = (V, \mathscr{D}, C, w_c, w_\theta)$ is defined as a set of *variables* V, a set of finite *domains* $\mathscr{D}$ (each $v \in V$ takes a value from D_v such that $D_v \in \mathscr{D}$), a set of *hard and soft constraints* $C = C_h \cup C_s$ restricting the values that variables can be assigned at the same time, *constraint weight* w_c as a function associating each soft constraint $c \in C_s$ with its weight $w_c(c)$, and *assignment weight* as a function w_θ associating the value d of each variable v with its weight $w_\theta(v/d)$.

An *assignment* ω for a weighted CSP $(V, \mathscr{D}, C, w_c, w_\theta)$ is a set of pairs v/d such that $v \in V, d \in D_v, D_v \in \mathscr{D}$ and each v appears at most once in ω. We call the assignment ω *complete* if each $v \in V$ appears in ω, otherwise ω is called a *partial assignment*. The assignment ω is an *extension* of assignment η iff $\eta \subseteq \omega$ holds.

Consider a constraint $c \in C$ defined on variables $X \subseteq V$ and denote $n_c = \|X\|$ and $scope(c) = X$. Assignment ω *satisfies* the constraint c iff x_i/d_i exists in ω for all variables $x_i \in scope(c)$ and $(d_1, \ldots d_{n_c}) \in c$ holds (written $\omega \vDash c$). An assignment ω is *consistent* if it satisfies all of the hard constraints $c \in C_h$ whose scopes have no unassigned

variables. A complete assignment ω which satisfies all hard constraints is called a *solution* (alternatively a solution is a complete consistent assignment).

The next task is the evaluation of soft constraints corresponding with the type of problem considered. Four different solution evaluation methods and solution comparators are proposed based on realistic problems appearing in timetabling and other areas. Note that all comparators are defined for consistent assignments in order to facilitate their easy application during search process. Optimal solutions must still be both complete and consistent assignments.

4.1 Initial Problem

The *initial problem* includes all of the hard and soft constraints given in the input data, i.e., it is defined as a weighted constraint satisfaction problem as discussed in the previous section. As noted before, this is a commonly solved problem and it is also the most common timetabling problem approached in this paper.

When looking at soft constraints and the resulting optimization problem, each assignment ω can be evaluated as $F_s\omega$

$$F_s\omega = \sum_{c\in C_s \wedge \omega \models \neg c} w_c(c) + \sum_{v/d\in\omega} w_\theta(v/d)$$

The *cost of initial problem* F_{wcsp} of the consistent assignment ω is represented by the tuple

$$F_{\text{wcsp}}\omega = (\|\omega\|, F_s\omega) \ .$$

The ordering $\leq_{\text{wcsp}}$ of costs between two consistent assignments of the initial problem is given by

$$F_{\text{wcsp}}\omega \leq_{\text{wcsp}} F_{\text{wcsp}}\eta \equiv \\ ((\|\omega\| > \|\eta\|) \vee ((\|\omega\| = \|\eta\|) \wedge (F_s\omega \leq F_s\eta))) \ .$$

An *optimal solution of the initial problem* is a solution σ with the minimal $F_{\text{wcsp}}\sigma$.

Consider a consistent assignment ω with $F_{\text{wcsp}}\omega = (\|\omega\|, F_s\omega)$ and a new assignment v/d such that v is not present in ω. Such an assignment may increase the violation of soft constraints by the value

$$\Delta F_s(\omega, v/d) = \\ w_\theta(v/d) + \textstyle\sum_{c\in C_s \wedge v\in scope(c) \wedge \omega \not\models \neg c \wedge (\omega\cup\{v/d\}) \models \neg c} w_c(c) \ .$$

$\Delta F_s(\omega, v/d)$ will be important during the value selection heuristics when comparing possible contributions to $F_s(\omega)$ for particular values d of the variable v to be assigned.

4.2 Feasibility Problem

The simplest problem is the *feasibility problem* — including hard constraints only — since all soft constraints have been removed. Alternatively, it could be denoted a satisfaction problem because it directly corresponds to the common constraint satisfaction problem (Dechter, 2003). Solving this problem is very important in the initial stage of the overall solution process since it allows the detection of possible inconsistencies in hard constraints. Such inconsistencies must be removed from the problem (by the human schedule manager) in order to find a solution of the problem satisfying all hard constraints.

The *cost of feasibility problem* F_{csp} of the consistent assignment ω corresponds to

$$F_{\text{csp}}\omega = \|\omega\| \ .$$

The ordering $\leq_{\text{csp}}$ between costs of feasibility problem of two consistent assignments is given by

$$F_{\text{csp}}\omega \leq_{\text{csp}} F_{\text{csp}}\eta \equiv (\|\omega\| \geq \|\eta\|) \ .$$

Note that any solution σ has the best possible value $F_{\text{csp}}\sigma$ corresponding to the number of the variables in the problem.

4.3 Minimal Perturbation Problem

Once the initial problem is solved and/or published to users, many requests for changes may occur and the goal is to generate a new timetable which reflects both these changes and the original timetable. Solving the *minimal perturbation problem (MPP)* allows the number of changes to the original solution (perturbations) to be kept as small as possible. A minimal perturbation problem was formally described by Sakkout et al. (1998), where the number of changes between two solutions was evaluated by a general distance function. This definition has been extended by Barták et al. (2004) to handle partial assignments. The problem specification described here is based on Barták et al. (2004) since it is necessary to work with partial assignments during the solving process when they are compared and evaluated.

The minimal perturbation problem can be defined by the constraint satisfaction problem $(V, \mathscr{D}, C)$ to be solved, an *initial assignment* δ being a consistent assignment of the original problem, and a *distance function* Φ which evaluates the number of changes between two assignments. An optimal solution is a solution σ of the problem $(V, \mathscr{D}, C)$ having minimal distance $\Phi(\sigma, \delta)$. The distance function Φ may consider the number of differently assigned variables (Barták et al., 2004), as in the function

$$\Phi_{\text{var}}(\omega, \delta) = \|\{x/d_1 | x/d_1 \in \omega \wedge x/d_2 \in \delta \wedge d_1 \neq d_2\}\| \ .$$

Often a problem specific distance function is also considered (Sakkout et al., 1998). In case of timetabling, it may be desirable to concentrate on the number of changed time placements (room placement is less critical and its changes may allow for a better solution with respect to time placement)

$$\Phi_{\text{time}}(\omega,\delta) = \\ \|\{x/d_1 | x/d_1 \in \omega \wedge x/d_2 \in \delta \wedge \text{time}(d_1) \neq \text{time}(d_2)\}\|$$

where $\text{time}(d)$ corresponds to a time placement associated with the value d. The changed assignments x/d_1 are called *perturbations*. Note that the definition of the MPP does not include a specification of the original problem. There are various reasons for this. The distance function compares only assignments, as Barták et al. (2004), and the solving process works with the new problem $(V,\mathscr{D},C)$ rather than with the original problem. Also, such a definition allows the original problem to be changed arbitrarily, in particular through the addition or removal of variables and constraints, and by changes in the variables' domain.

In this work, the weighted constraint satisfaction problem $P = (V,\mathscr{D},C,w_c,w_\theta)$, rather than the common constraint satisfaction problem $(V,\mathscr{D},C)$, is considered within the MPP. It is, therefore, necessary to concentrate on optimization of both soft constraints $F_s\omega$ and perturbations $\Phi(\omega,\delta)$. Both of these aspects, along with consideration of partial assignments, are handled by F_{mpp} as the *cost of MPP* of the consistent assignment ω wrt. initial assignment δ

$$F_{\text{mpp}}(\omega,\delta) = (\|\omega\|, F_s\omega + w_{\text{mpp}}\Phi(\omega,\delta)) \ .$$

$F_s\omega$ reflecting violation of soft constraints was defined in Section 4.1. The weight w_{mpp} expresses the relation between the number of problem changes and the desired satisfaction of soft constraints.

Next, the ordering of costs for the MPP is defined. The ordering $\leq_{\text{mpp}}$ between costs of MPP for two consistent assignments ω and η with respect to the same initial assignment δ is given by

$$F_{\text{mpp}}(\omega,\delta) \leq_{\text{mpp}} F_{\text{mpp}}(\eta,\delta) \equiv \\ ((\|\omega\| > \|\eta\|) \vee ((\|\omega\| = \|\eta\|) \wedge \\ (F_s\omega + w_{\text{mpp}}\Phi(\omega,\delta) \leq F_s\eta + w_{\text{mpp}}\Phi(\eta,\delta)))) \ .$$

An *optimal solution of MPP* with respect to δ is a solution σ with the minimal $F_{\text{mpp}}(\sigma,\delta)$.

Now consider a consistent assignment ω and an initial assignment δ with $F_{\text{mpp}}\omega = (\|\omega\|, F_s\omega + w_{\text{mpp}}\Phi(\omega,\delta))$ and a new assignment v/d such that v is not included in ω. The possible contribution $\Delta F_s(\omega,v/d)$ of this value into $F_s(\omega)$ is the same as in Section 4.1. When comparing contributions of different values d of the variable v, information about changes with respect to the initial assignment δ is also included. The contribution to the distance function Φ_{var} can be calculated as

$$\Delta\Phi_{\text{var}}(\delta,v/d) = \begin{cases} w_{mpp} & \exists v/d_i \in \delta \wedge d_i \neq d \\ 0 & \text{otherwise} \end{cases} .$$

This means that the general definition of $\Delta\Phi(\delta,v/d)$ depends on the definition of $\Phi(\omega,\delta)$, for example $\Delta\Phi_{\text{time}}(\delta, v/d)$ is a simple modification of $\Delta\Phi_{\text{var}}(\delta,v/d)$.

The sum $\Delta F_s(\omega,v/d) + \Delta\Phi(\delta,v/d)$ will be applied to evaluate different values during value selection heuristics.

4.4 Interactive Problem

Another means of incorporating changes into timetabling problems is through interactive problem solving. While all decisions are left to the user, an interactive technique may be used to explore the possibility of making changes and to propose various suggestions to the user. In one *interactive step*, the user may propose a change to a single class that requires the reassignment of multiple other classes. Several interactive steps may also be made in combination to achieve the desired assignments for a set of classes. A user may commit a computed suggestion at any time and make it a new current solution (consistent assignment). Additional interactive steps and exploration of other suggestions may continue.

Example 1 Figure 3 illustrates a set of suggestions that is presented to the user, who may then decide which changes are the most desirable to be made. In this example, a change in the placement of the class PSY 120 Lec 5 has been requested. The reassignment of not more than two additional classes will be considered. This limits both the amount of calculation required and disruption to the original timetable.

Suggestions

Score	Class	Date	Time	Room	Students
+43	PSY 120 Lec 5	Full Term	MWF 7:30a	WTHR 200 → CL50 224	0
+48.4	PSY 120 Lec 5	Full Term	MWF 7:30a → TTh 7:30a	WTHR 200 → CL50 224	+10
+63.3	PSY 120 Lec 5	Full Term	MWF 7:30a → MWF 4:30p	WTHR 200 → LILY 1105	+14
	POL 130 Lec 2	Full Term	MWF 4:30p → MWF 9:30a	LILY 1105 → RHPH 172	
+63.9	PSY 120 Lec 5	Full Term	MWF 7:30a → MWF 4:30p	WTHR 200 → LILY 1105	+16
	POL 130 Lec 2	Full Term	MWF 4:30p	LILY 1105 → FRNY G140	
+63.9	PSY 120 Lec 5	Full Term	MWF 7:30a → MWF 4:30p	WTHR 200 → LILY 1105	+16
	POL 130 Lec 2	Full Term	MWF 4:30p	LILY 1105 → LYNN 1136	

(all 235 possibilities up to 2 changes were considered, top 5 of 22 suggestions displayed) Search Deeper

Fig. 3 List of suggestions provided to user.

Consider the components of the problem to be solved in one interactive step. There is an original weighted constraint satisfaction problem $P = (V,\mathscr{D},C,w_c,w_\theta)$ with the constraints $C = C_h \cup C_s$ and a consistent assignment δ. There is a variable v_{bb}, which should be assigned a new value (different than in δ if assigned there), and there may exist a set

of required changes μ proposed by the user representing assignments which must be present in a new solution. These proposed assignments allow the user to explore the impact of a wide variety of scenarios. Assignments in μ may also have been computed during earlier steps of the interactive process. Note that the specification of assignments computed by earlier interactive steps is necessary to avoid possible repetitive changes.

An *interactive problem* is given by the consistent assignment of the original problem δ, by the maximum number M of additional changes in a new solution (called *maximum depth*), and by the new problem $P' = (V, \mathscr{D}, C'_h \cup C_s, w_c, w_\theta)$ where C'_h equals to $C_h \cup \{v_i = d_i \,|\, v_i/d_i \in \mu\} \cup C_{bb}$ and C_{bb} corresponds to the set $\{v_{bb} \neq d\}$ if v_{bb} is assigned a value d in δ, otherwise C_{bb} is an empty set.

Ω is the ordered set of consistent assignments ω of the problem P' (called *suggestions*) such that $\Phi_{\text{var}}(\omega, \delta) \leq M + \|\mu\|$ and all variables present in δ and μ are assigned. Here the distance function Φ_{var} defined in Section 4.3 allows computation of the number of differently assigned variables. Suggestions in Ω are evaluated by the function F_{wcsp} and ordered using $<_{\text{wcsp}}$. An *optimal suggestion* is a suggestion σ with the minimal $F_{\text{wcsp}}\sigma$. The best n elements (if they exist) of the suggestions Ω are usually of interest to provide users some alternatives. When Ω contains partial assignments ω further interactive steps must be processed to compute suggestions with all variables assigned. As noted previously, for each additional interactive step, the interactive problem must include a new assignment of v_{bb} in μ along with changed assignments v_i/d_i if the user decides to explore the current suggestion, where the changed assignments are described as $v_i/d_i \in \omega \wedge v_i/d_o \in \delta \wedge d_o \neq d_i$.

5 Domain Variables and Constraints

This section describes domain variables along with the hard and soft constraints in the timetabling problem. Each class is specified by a domain variable representing the desired values for meeting times (weeks and patterns of days the class should meet during the term, start times and duration of all meetings) and rooms. Domain variables will be denoted $v = (v_w, v_p, v_s, v_d, v_r)$ and their particular values $d = (d_w, d_p, d_s, d_d, d_r)$.

Example 2 $d =$ (11110000, MW, 7:30 am, 50, WTHR 200) represents the value of the domain variable for a class to be taught in the first four weeks of the eight week semester, on Monday and Wednesday at 7:30 am. The class will last 50 minutes and will be taught in room WTHR 200.

The domain variable $v = (v_w, v_p, v_s, v_d, v_r)$ with $(v_w, v_p, v_s, v_d) \in \{$(11111111, MWF, 7:30 am, 50), (11111111,MWF, 8:30 am, 50), (11111111, MWF, 9:30 am, 50), (11111111, TTh, 7:30 am, 75), (11111111, TTh, 9:00 am, 75)$\}$ and $v_r \in$ $\{$WTHR 200, CL50 224, EE 129, LILY 1105$\}$ represents a class which should be taught in one of the four given rooms, either on Monday–Wednesday–Friday at 7:30 am, 8:30 am or 9:30 am lasting 50 minutes, or on Tuesday–Thursday at 7:30 am or 9:00 am lasting 75 minutes. Note that the above time specification actually corresponds to the request that the class should be taught a total of three hours per week, during a time frame from 7:30 am to 10:30 am using a standard time pattern of Monday–Wednesday–Friday for 50 minutes or Tuesday–Thursday for 75 minutes.

5.1 Hard Constraints

Tying meetings together into standard time patterns in this way considerably simplifies the problem constraints. Time pattern constraints can be directly set on each domain variable of the class by simple inclusion of consistent combinations into the domain. Other hard unary constraints on time and room from Table 3 can be implemented by direct value removal.

Consider the properties of consistent assignments that are maintained throughout the search. Since no constraint propagation (Bessiere, 2006) is processed (see details in Section 6), the domains of future (unassigned) variables are not changed. Also, consistent assignments maintain the consistency of only those constraints having all variables assigned. This means that inconsistency of constraints is detected after assignment of their last variable. This is unnecessarily weak for non-binary constraints since many of their variables could have been assigned improper values. However, it is possible in many cases to translate non-binary constraints into sets of simpler (typically binary) constraints. This allows consideration of only consistent assignments that satisfy the components of complex constraints on assigned variables.

Resource constraints allow assignment of only one class to a room (or an instructor) during each time period. Each resource constraint is considered as a set of binary constraints prohibiting the placement of each pair of classes into overlapping time periods. This means that the satisfaction of these binary constraints is required as soon as the two corresponding domain variables are assigned. This can be simply realized with the help of an array of assigned classes. Any time a new class is assigned it must be placed into free slots of the array which efficiently enforces consistency of a new assignment.

The hard distribution constraints used in this problem are also decomposed to perform consistency checks on their assigned variables. The consistency of the constraint “classes placed in similar times” can be realized by simply requiring all assigned variables to be placed in the same predefined block of time (e.g., morning, noon, afternoon).

The constraint "same (or different) meeting days/times/rooms for classes" has an analogous behavior. All assigned classes must be taught at he same (or different) days/times/rooms.

The constraint "time precedences between classes" ensures that all assigned classes respect required precedence relationships. There must also be enough time available to place unassigned classes such that these precedences are respected. To achieve this, possible durations (and time pattens) of unassigned classes are taken into account to fit properly between corresponding classes.

The constraint "time between classes" requires a specific amount of time between classes. Assuming that some classes have already been assigned, all possible durations (and time patterns) for the unassigned classes are considered and some assignment must exist such that all unassigned classes are placed between assigned classes respecting the desired time between consecutive classes.

5.2 Soft Constraints

All unary soft constraints (individual times, individual buildings/rooms, individual room equipment) are modeled by an assignment weight w_θ which allows association of a weight with each assignment of a domain variable for the class. More specifically, the assignment weight for the class $v = (v_w, v_p, v_s, v_d, v_r)$ and its assignment $d = (d_w, d_p, d_s, d_d, d_r)$ corresponds to

$$w_\theta(v/d) = w_{\text{time}} w_\theta((v_w, v_p, v_s, v_d)/(d_w, d_p, d_s, d_d)) + w_{\text{room}} w_\theta(v_r/d_r)$$

meaning that each time and room placement has a specific weight. These placements are related together using global parameters w_{time} and w_{room} that are defined according to the importance of time and room preferences in the problem.

Student conflicts between two classes are taken into account using soft constraint with the weight corresponding to the number of students s enrolled in both classes, i.e., there is a soft constraint c between each two classes v_1 and v_2 with the weight $w_c(c) = s$ which is satisfied when the classes v_1 and v_2 do not overlap. Violation of this constraint is detected as soon as both variables are assigned as it corresponds with the behavior of hard constraints.

Soft distribution constraints are considered with the help of a weight $w_c(c)$ associated with each constraint c which takes a part in the overall objective function iff the constraint c is not satisfied. Possible violation of soft distribution constraints is detected with the help of decomposition into simpler constraints similarly to their counterparts in hard constraints. Note that the number of soft distribution constraints is very low in most Purdue problems. Most of the distribution constraints are requirements set by the course structure as hard constraints (and are generated during transformation of courses to classes — see Section 3 for details). This means that violation of a few soft distribution constraints may significantly impact the distribution preferences criterion.

6 Iterative Forward Search

The iterative forward search (IFS) algorithm (Müller, 2005; Müller et al., 2005) that has been developed is a generic approach, relevant to solving a wide range of problems. It is applied here to the initial problem, feasibility problem, and minimum perturbation problem. Each semester, schedule managers solve approximately 70 different school and departmental problems at Purdue University. While all of these fall into the category of course timetabling problems, the IFS algorithm has also been applied to problems from all three tracks of the International Timetabling Competition 2007 (Müller, 2008), and is used in the examination timetabling system at Purdue University. In addition, its applicability to student sectioning has been demonstrated (Müller and Murray, 2008).

Iterative forward search is a constructive search algorithm with repair features that is based on a constraint model of the problem defined by domain variables and constraints. In contrast to classical domain filtering (Mackworth, 1977; Debruyne and Bessière, 2001; Bessiere, 2006), which is not processed here, consistency is maintained for assigned variables in a similar manner to backtracking (van Beek, 2006) by keeping the assignment consistent on the past variables and the current variable. In comparison with tree search methods, any part of an existing partial assignment can be removed to avoid existing conflicts. It is similar in this regard to the ruin-and-recreate method (Schrimpf et al., 2000), where parts of an existing assignment are removed and created again, or with dynamic backtracking (Ginsberg, 1993), which allows changes in the ordering of assigned variables together with their unassignment. As discussed in earlier works (Müller, 2005), stronger forms of consistency, like arc consistency, are too computationally expensive for the approach and do not introduce improvements in the solution quality. This also means that global constraints like `alldifferent` (Régin, 1994), or filtering constraints for disjunctive and cumulative scheduling (Baptiste and Le Pape, 1995; Caseau and Laburthe, 1996), are not used since no propagation is really done.

The algorithm is parameterized by cost function F and comparator $<$ which can be directly applied to optimize the quality of the solution with respect to the type of problem (initial, feasibility, MPP). The search is processed for the weighted CSP $P = (V, \mathscr{D}, C, w_c, w_\theta)$ with domains in $\mathscr{D}$ respecting all hard unary constraints. It starts with an empty solution and maintains a consistent assignment during each

iteration. At each step, the current consistent assignment ω is extended by assignment of one variable or, alternatively, a new value is found for some variable in the existing assignment ω. Consequently, any conflicting variable assignments are removed and their original domains are restored. This means for the whole solving process that each variable v takes either the value $d \in D_v$ defined by the current consistent assignment ω or its original domain D_v as defined by $\mathscr{D}$. The quality of each new current consistent assignment is evaluated by the cost function F. Iterations are repeated to find a solution ω of the desired quality with respect to F and termination is typically invoked after a timeout period.

The structure of the IFS algorithm can be seen in Figure 4. The algorithm is parameterized by the weighted CSP

```
1: function IFS(P,F,<,δ)
2:     i = 0
3:     ω = ∅
4:     σ = ∅
5:     while CANCONTINUE(ω, i) do
6:         i = i + 1
7:         v = SELECTVARIABLE(P, ω)
8:         d = SELECTVALUE(P, ω, δ, F, <, v)
9:         γ = HARDCONFLICTS(P, ω, v/d)
10:        ω = ω\γ ∪ {v/d}
11:        if F(ω,δ) < F(σ,δ) then σ = ω
12:    end while
13:    return σ
14: end function
```

Fig. 4 Pseudo-code of the iterative forward search algorithm.

$P = (V, \mathscr{D}, C, w_c, w_\theta)$, initial assignment δ, cost function F and corresponding comparator $<$. IFS$(P, F_{\text{wcsp}}, <_{\text{wcsp}}, \emptyset)$ is used to solve the initial problem, IFS$(P, F_{\text{csp}}, <_{\text{csp}}, \emptyset)$ is applied for the feasibility problem and IFS$(P, F_{\text{mpp}}, <_{\text{mpp}}, \delta)$ allows solution of the MPP. As mentioned earlier, the domain $\mathscr{D}$ of the problem P must respect hard unary constraints in P.

The current consistent assignment ω is initialized as an empty set. The best achieved consistent assignment is maintained in σ. The iteration counter i allows the solver to be terminated with the CANCONTINUE(ω, i) function which stops execution of the algorithm once a solution of desired quality is found or the desired relation to the value of iteration counter i has been achieved. During each iteration, a variable is selected by the function SELECTVARIABLE(P, ω). Unassigned variables are preferentially selected when ω is not a complete assignment. Otherwise, assigned variables are selected to find a more proper value improving the cost function F during the next search. Any ties are broken using random variable ordering since the variable selection heuristic did not prove to be meaningful (Müller et al., 2005) due to the possibility of repetitive variable unassignments. Next, value selection is processed by means of the function SELECTVALUE$(P, \omega, \delta, F, <, v)$. As is common for other search algorithms, selection of proper values for the variable is very important. Each value is evaluated with respect to the soft constraints and perturbations using the function $\Delta F_s(\omega, v/d) + \Delta\Phi(\delta, v/d)$ (both components are zero in the feasibility problem, while only the perturbation component is equal to zero in the initial problem). Evaluation with respect to the hard constraints is processed using conflict-based statistics, which are described in the next section. All of these factors are taken into account and the best value is finally selected. In the case of the MPP, if there is an assignment for a selected variable in the initial assignment δ, this value is selected prior to evaluation with some high probability (typically 80%) in order to maintain similarity with the initial assignment (otherwise value selection and evaluation is processed). It may happen that a newly selected value will cause some conflicts with already assigned variables (see next paragraph for details on their detection). These conflicting assignments γ are removed and corresponding variables become unassigned. The new consistent assignment is compared with the best achieved assignment and possibly stored in σ. Finally, a new iteration of the algorithm may continue.

The structure of the HARDCONFLICTS$(P, \omega, v/d)$ function is given in Figure 5. This function computes the set of

```
1: function HARDCONFLICTS(P, ω, v/d)
2:     if ∃d_v : v/d_v ∈ ω then γ = {v/d_v}
3:     else γ = ∅
4:     for c ∈ C_h ∧ v ∈ scope(c) do
5:         β = ω\γ ∪ {v/d}
6:         if β ⊨ ¬c then
7:             find α ⊆ ω\γ such that β\α ⊨ c
8:             γ = γ ∪ α
9:         end if
10:    end for
11:    return γ
12: end function
```

Fig. 5 Pseudo-code of function for computing conflicting variables.

conflicting assignments γ with the pair v/d such that $\omega\backslash\gamma \cup \{v/d\}$ introduces a consistent assignment. First, it is necessary to include in the conflicts γ any possible earlier assignment d_v of v included in ω (line 2). The next step is to sequentially go through the list of hard constraints $c \in C_h$ where v is included ($v \in scope(c)$). For each conflicting constraint c some assignments of variables in the scope of c must be added to γ to remove the conflict. These are the set of assignments α whose removal allows satisfaction of the constraint c using assignment v/d. Of course, it is important to choose the set α as small as possible and, even more, to select it such that the final set γ with all removed assignments will be the smallest possible. However, the algorithm does not try to compute the minimal set for either

γ or α as it could be computationally expensive. The aim is to efficiently compute a reasonably small set γ and correct possible mistakes in future iterations of the algorithm.

6.1 Conflict-based Statistics

The success of the search method is greatly strengthened by the introduction of a technique called conflict-based statistics. As will be presented in Section 8, the iterative forward search algorithm was unable to find a complete solution without use of this feature. It will be shown here how conflict-based statistics can be included in the iterative forward search process. However, this is a general approach which can be easily incorporated into other search algorithms (Müller et al., 2004).

The main idea behind conflict-based statistics is to memorize conflicts and discourage their future repetition. For instance, when a value is assigned to a variable, conflicts with some other assigned variables may occur. This means that there are one or more constraints which prohibit the applied assignment together with the existing assignments. A counter tracking how many times such an event occurred in the past is stored in memory. When a variable is selected for assignment, the stored information about repetition of past conflicts is taken into account. There is no limit on the size of statistics and no aging is applied to prevent short-term or long-term cycles.

A variety of methods similar to conflict-based statistics have been applied in earlier works. Classically, the variable selection methods of backtracking search schemes select the variable with the minimal number of conflicts (Minton et al., 1992). Memorization of the weighted conflicts is proposed in (Jussien and Lhomme, 2002), though conflicting assignment are not stored. Alternatively, backjumping algorithms (Dechter and Frost, 2002) may store conflicting assignments (or variables) to decide on the proper jump within a backtracking search to avoid unnecessary backtracking. Local search schemes like Guided Local Search (Voudouris and Tsang, 2003) are also known to build up penalties during a search and use them to escape from local minima.

Conflict-based statistics are a data structure that memorizes hard conflicts that have occurred during the search together with their frequency and the assignments that caused them. More precisely, it is an array $CBS[x = d_x \rightarrow \neg y = d_y] = c_{xy}$. This means that the assignment $x = d_x$ caused a hard conflict with the assignment $y = d_y$ c_{xy} times in the past. Note that this does not imply that assignments $x = d_x$ and $y = d_y$ cannot be used together in case of non-binary constraints. The proposed conflict-based statistics does not actually work with any constraint. It only memorizes the conflict assignments together with the assignment that caused them. This helps capture similar cases as well, e.g., when the applied assignment violates a constraint different from the past ones, but some of the conflicts created are the same. It also reduces the total space allocated by the statistics.

If a value d is selected for a variable v within the iterative forward search algorithm, then $\textsc{HardConflicts}(P, \omega, v/d)$ chooses previous assignments $\gamma = \{v_1/d_1, v_2/d_2, \ldots v_n/d_n\}$ to be unassigned in order to enforce consistency of the new partial assignment. As a consequence, the counters are incremented

$$CBS[v = d \rightarrow \neg v_1 = d_1], CBS[v = d \rightarrow \neg v_2 = d_2], \ldots, CBS[v = d \rightarrow \neg v_n = d_n] \ .$$

Conflict-based statistics are used as part of the value selection criterion. A *weighted min-conflict* criterion is used in the search procedure together with the above mentioned function $\Delta F_s(\omega, v/d) + \Delta\Phi(\delta, v/d)$. Each value d is evaluated as the sum of the number of conflicts multiplied by their frequencies

$$\sum_{v_i/d_i \in \omega, v_i/d_i \in \textsc{HardConflicts}(P,\omega,v/d)} CBS[v = d \rightarrow \neg v_i = d_i]$$

and the value with the smallest contribution is preferred. Stated in another way, the weighted min-conflict approach helps to select a value that should cause fewer conflicts than another value. The point is that these conflicts are less frequent, and therefore have a lower weighted sum. The hope is that this can help considerably in moving the search out of a local minimum.

Example 3 An example of this structure

$$x = 1 \ \Rightarrow \ 3 \times \neg y = 2, 4 \times \neg y = 3, 2 \times \neg z = 1, 120 \times \neg v = 1$$

expresses that variable y conflicted three times with its assignment 2 and four times with its assignment 3, variable z conflicted two times with its assignment 1 and v conflicted 120 times with its assignment 1, all because of later selections of value 1 for variable x. This structure can be used in the value selection heuristics to evaluate conflicts with the assigned variables. For example, if there is a variable x selected and its value 1 is in conflict with an assignment $y = 2$, it is known that a similar problem has already occurred three times in the past, and the conflict $x = 1$ can be weighted with the number 3.

An important question is the space complexity of the data structure needed for conflict-based statistics. It can be implemented as a hash table which is initially empty and contains only non-zero counters during computation. Consider two different variables $x = d_x$ and $y = d_y$. In the worst case, a counter exists for each pair of possible assignments $x = d_x$ and $y = d_y$. However, each increment of a counter in the statistics means an unassignment of an assigned variable. Therefore each counter $CBS[x = d_x \rightarrow \neg y = d_y] = c_{xy}$ in the statistics means that there was an assignment $y = d_y$

which was unassigned c_{xy} times when d_x was assigned to x. Because there is only one assignment done in each iteration, the following invariant holds during the search: the total sum of all counters in the statistics plus the current number of assigned variables equals to the number of processed iterations. Therefore, if the hash table is used, the total number of all its counters will never exceed the number of iterations processed so far.

Conflict-based statistics is also very important when detecting conflicts in the feasibility problem. Memorized conflicts gathered during the search can be provided to users. High values for some CBS counters highlight difficult parts of the problem which can consequently lead to the detection of inconsistencies by users.

Example 4 An example of the output provided to the user is presented in Figure 6 where conflicts between the class CS 101 Lab 2 and other classes can be seen. For example, there are 260 conflicts with CS 101 Lab 3 class on Monday 11:30 am. Based on this output, the user can conclude that four classes CS 101 Lab 2, CS 101 Lab 3, CS 101 Lab 1, and CS 101 Lab 4 request only three valid times in the same room which necessarily lead to the conflict.

Current Assignment of C S 101 Lab 2

Not assigned.

Room Locations: 1 (EDUC 108)

Time Locations: 3 (M 9:30a, M 11:30a, M 1:30p)

Conflict-based Statistics

- 2123× Room EDUC 108
  - 718× M 11:30a - 1:20p Full Term EDUC 108
    - 260× C S 101 Lab 3 ← M 11:30a - 1:20p Full Term EDUC 108
    - 235× C S 101 Lab 1 ← M 11:30a - 1:20p Full Term EDUC 108
    - 222× C S 101 Lab 4 ← M 11:30a - 1:20p Full Term EDUC 108
    - 1× ENGR 101 Lab 2 ← M 11:30a - 1:20p Full Term EDUC 108
  - 718× M 1:30p - 3:20p Full Term EDUC 108
    - 256× C S 101 Lab 1 ← M 1:30p - 3:20p Full Term EDUC 108
    - 235× C S 101 Lab 4 ← M 1:30p - 3:20p Full Term EDUC 108
    - 226× C S 101 Lab 3 ← M 1:30p - 3:20p Full Term EDUC 108
    - 1× ENGR 101 Lab 2 ← M 1:30p - 3:20p Full Term EDUC 108
  - 687× M 9:30a - 11:20a Full Term EDUC 108
    - 252× C S 101 Lab 1 ← M 9:30a - 11:20a Full Term EDUC 108
    - 240× C S 101 Lab 4 ← M 9:30a - 11:20a Full Term EDUC 108
    - 192× C S 101 Lab 3 ← M 9:30a - 11:20a Full Term EDUC 108

Fig. 6 Conflict-based statistics for one class CS 101 Lab 2.

7 Branch and Bound

The interactive problem (see Section 4.4 for definition) is solved using a form of the branch and bound search algorithm (Dechter, 2003) which is applied on the depth-bounded backtracking (Cheadle et al., 2003). The application of depth-bounded search is permitted by the existence of a natural limit on the number of additional variables which is directly related with the explored depth of the corresponding search tree. Clearly it does not make sense to present solutions changing too many additional variables since such solutions would not be desirable for users. Often there are solutions with only one additional change. Of course, this limit can be increased if needed, e.g., when no satisfactory solution is found. Still the maximum number of additional variables is often rather small in practice, which permits an interactive use of this algorithm.

The structure of the initialization procedure of the branch and bound algorithm is presented in Figure 7. Parameters of

```
1: function BB(P, δ, μ, v_bb)
2:     if {v_bb/d} ⊆ δ then δ = δ\{v_bb/d}
3:     else d = nil
4:     γ = {v_bb/d}
5:     for v_i/d_i ∈ μ do
6:         if {v_i/d_o} ⊆ δ then o = {v_i/d_o}
7:         else o = ∅
8:         γ = γ ∪ HARDCONFLICTS(P, δ, v_i/d_i)\o
9:         δ = δ\o ∪ {v_i/d_i}
10:    end for
11:    return BACKTRACK(P, δ, μ, γ, ∅, 0)
12: end function
```

Fig. 7 Pseudo-code of initialization procedure for branch and bound algorithm.

the search are an original problem P, its consistent assignment δ, a set of required changes μ, and a variable v_{bb} to be changed (or assigned). It should be emphasized that δ may not necessarily be a complete assignment (and solution) of P since future interactive steps are expected to find a solution with all variables assigned.

When v_{bb} is assigned a value in δ, it is removed (line 2) and placed into the set of conflicting assignments γ (line 4). When v_{bb} is unassigned (line 3), it is placed in γ with some dummy value since further search will be based on the list of conflicts γ. Consequently all required assignments v_i/d_i in μ are processed. If v_i already has an assigned value d_o in δ, this assignment is removed from δ (lines 6 and 9). Possible conflicts in δ with the assignment v_i/d_i are removed with the help of the HARDCONFLICT$(P, \delta, v_i/d_i)$ function (see Figure 5) and added to the set γ (line 8). Possible earlier assignment v_i/d_o is replaced by v_i/d_i, i.e., it must be removed from γ (line 8).

The initialization stage is completed by calling the function BACKTRACK$(P, \delta, \mu, \gamma, \emptyset, 0)$ which performs the branch and bound algorithm itself. The parameter $\emptyset$ initializes the current set of suggestions and the last parameter counts the depth explored until the current point (i.e., it is initialized by 0). The pseudo-code of this function is displayed in Figure 8.

First, a check is performed that the maximum depth M together with the number of additional changes $\|\mu\|$ is not

```
1: function BACKTRACK(P, δ, μ, γ, Ω, m)
2:     if ||γ|| + m > M + ||μ|| then return ∅
3:     if γ = ∅ then return δ
4:     if TIMEOUT then return ∅
5:     if LB(F_wcsp(δ ∪ γ)) ≥_wcsp F_wcsp(Ω[n]) then return ∅
6:     v = SELECTVARIABLEBB(γ)
7:     let v/d_o ∈ γ
8:     for d ∈ D_v ordered by ΔF_s(δ, v/d) do
9:         if d = d_o then continue
10:        α = HARDCONFLICTS(P, δ, v/d)
11:        if α ∩ μ ≠ ∅ then continue
12:        Ω = Ω ∪ BACKTRACK(P, δ ∪ {v/d}, μ ∪ {v/d},
                             γ\{v/d_o} ∪ α, Ω, m + 1)
13:    end for
14:    return Ω
15: end function
```

Fig. 8 Pseudo-code of recursive calls in branch and bound algorithm.

exceeded by the current depth m together with the number of unresolved conflicts $\|\gamma\|$ (line 2). If this occurs, the search within this branch is interrupted and an empty set of suggestions is returned as a result. Successful computation of a new suggestion is detected when all conflicts are resolved (γ is empty) and the current consistent assignment δ is returned (line 3). Since the solution search is interactive, it is permissible to look for solutions within a set time limit, typically 5 seconds (line 4). For the initial call of the BACKTRACK function this means that only suggestions found within the time limit are computed.

An important bounding step is processed in line 5 of the code. A lower bound for the evaluation function F_{wcsp} is computed for assignment $\delta \cup \gamma$. While computation of the component $F_{\text{wcsp}}(\delta)$ is precise, the lower bound for the conflicts γ is computed for each variable v in γ such that the best possible evaluation among all its values $d \in D_v$ is included. When the quality of $LB(F_{\text{wcsp}}(\delta \cup \gamma))$ is worse than the quality of the n-th suggestion ($\Omega[n]$ denotes the n-th best element in the current set Ω) the search of the current branch is again interrupted, since only the n best elements of Ω are of interest.

The next part of the algorithm concentrates on assignment of a new variable from the conflict set γ. First a variable v is selected from γ (line 6). This selection is simply processed in the same order as assignments were placed into γ (resulting in assignment of the variable v_{bb} first). It is expected that the variable v has been originally assigned a value d_o which is present in γ (line 7). Consequently, all values d of variable v are processed in the order given by increasing values of $\Delta F_s(\delta, v/d)$. This means that values with a smaller number of conflicts in soft constraints are explored first. Evaluation of possible conflicts in hard constraints, which could be computed by the conflict-based statistics, is not processed since calculation of all conflicts for all pairs v/d is computationally expensive given the time limits in the interactive problem. Certainly the search is not performed for the original value d_o (line 9). The conflicting assignments α with hard constraints are computed using HARDCONFLICTS function as usual. Naturally, none of the required assignments from μ can be contained in α (line 11).

In line 12, new suggestions are added to Ω by performing a recursive call of the BACKTRACK function where a new assignment v/d is added to δ. Also, v/d is added to μ to explore solutions with this value (assignments in μ cannot be overridden). Of course, v/d_o is removed from γ and computed conflicts α are added there. The last parameter of the BACKTRACK function is an increased depth of the search since one more variable was assigned.

Finally, the computed Ω cumulated from all assignments of variable v is returned as a result.

8 System and Results

The system being implemented at Purdue University has been designed as a multi-user application with a completely web-based interface. The primary technologies used are the Enterprise Edition of Java 2 (J2EE), Hibernate, and an Oracle database. Experiments have been run on an Apple Mac Pro server machine with two dual core 3.0 GHz Intel Xeon processors and 4 GB of RAM, using Mac Os X and Java 1.5.

8.1 Initial Problem

Table 4 shows a summary of solutions for the individual problems that were discussed in Section 2, i.e., the large lecture problem (llr), the centrally timetabled computer laboratory problem (lab), six different departmental problems, and combined problem (c8).

The column titled *Final* contains the results that were produced by human schedule managers using the timetabling application for Spring 2007 and Fall 2007, respectively. These solutions were computed individually using the automated solver for the initial problem; however, changes were also made using the interactive solver.

The column *Run separately* contains results from 30 individual test runs each taking 30 minutes. The llr problem was solved first. Departmental problems were solved on top of the llr solution. The lab problem was solved last, on top of the llr and departmental solutions. Results for combined problems pu-spr07-c8 and pu-fal07-c8 were computed via summarization of the results from particular separate runs.

The column *Run combined* contains average results from 30 combined test runs for pu-spr07-c8 and pu-fal07-c8 problems each taking 4 hours (30 minutes per each of eight included sub-problems). This means that results for all other problems were computed via projection of results from combined runs.

Table 4 Particular runs (final, separate, combined, bound) with results for student enrollments (S), time preferences (T), room preferences (R), distribution preferences (D) given in percentage.

	Final				Run separately				Run combined				Bound			
Problem	S	T	R	D	S	T	R	D	S	T	R	D	S	T	R	D
pu-spr07-llr	98.63	89.71	92.86	66.67	98.86	93.20	86.90	72.22	97.67	90.11	84.86	63.47	99.42	98.82	99.47	100.00
pu-fal07-llr	98.76	81.77	91.69	96.55	99.00	89.49	78.05	93.56	98.25	88.18	74.98	63.69	99.54	96.29	98.94	100.00
pu-spr07-ms	99.37	68.34	76.22	57.14	99.62	75.45	75.96	48.39	98.28	77.42	73.98	55.98	99.87	86.11	94.51	78.57
pu-fal07-ms	97.80	71.48	80.00	72.73	99.62	71.88	86.33	63.68	98.99	70.60	85.42	52.86	99.71	84.39	97.50	76.46
pu-spr07-cs	94.85	83.86	100.00	33.33	97.15	73.91	100.00	33.33	93.56	70.92	100.00	44.17	98.55	86.27	100.00	100.00
pu-fal07-cs	94.14	83.20	90.48	100.00	98.53	76.40	83.49	89.67	97.15	76.09	83.17	70.19	99.28	96.91	100.00	100.00
pu-spr07-ecet	95.71	93.31	100.00	100.00	98.26	90.63	100.00	100.00	89.63	88.91	100.00	100.00	99.55	97.49	100.00	100.00
pu-fal07-ecet	95.93	95.68	100.00	100.00	97.95	92.45	100.00	100.00	90.35	89.68	100.00	100.00	99.34	99.28	100.00	100.00
pu-spr07-sa	95.81	81.54	100.00	50.00	99.30	96.34	100.00	48.33	98.38	93.32	100.00	45.00	99.57	100.00	100.00	50.00
pu-fal07-sa	96.07	79.38	100.00	100.00	99.29	95.29	85.71	100.00	96.51	87.35	62.14	100.00	99.74	100.00	100.00	100.00
pu-spr07-cfs	93.72	87.32	94.44	100.00	97.66	89.74	82.22	82.50	93.70	93.32	79.63	64.17	98.42	99.12	100.00	100.00
pu-fal07-cfs	94.09	96.30	50.00	86.67	98.12	97.14	85.00	92.67	94.70	94.97	66.67	84.30	98.52	100.00	100.00	93.33
pu-spr07-vpa	92.70	2.44	40.00	100.00	97.08	88.62	76.67	100.00	95.41	81.30	76.00	100.00	97.69	90.24	100.00	100.00
pu-fal07-vpa	93.19	0.00	100.00	100.00	96.79	0.00	100.00	100.00	95.06	0.00	100.00	100.00	96.79	0.00	100.00	100.00
pu-spr07-lab	97.45	87.60	75.76	68.02	99.39	94.08	69.19	50.30	97.71	94.58	68.15	57.00	99.82	97.67	83.31	86.29
pu-fal07-lab	85.42	89.74	71.46	77.03	97.71	84.73	44.25	38.15	97.29	85.95	39.32	22.00	98.12	93.69	87.64	78.38
pu-spr07-c8	97.99	84.87	82.81	61.39	98.69	90.16	77.37	50.70	98.16	89.91	75.79	56.58	98.95	97.55	91.91	87.59
pu-fal07-c8	98.35	83.01	87.55	78.00	98.63	86.70	73.49	61.04	98.55	86.62	70.43	54.18	99.35	95.76	96.36	81.15

The last column *Bound* provides information about the best possible solution quality for particular criteria using the solver. Reported bounds were obtained using the solver for 1 hour when only one criterion was considered at a time.

Results for particular automated runs, including root-mean-square variances, are available in Table 5. The values given here correspond to the column Run combined (Table 4) for problems pu-spr07-c8 and pu-fal07-c8 and to the column Run separately for other problems (remaining values were computed either by projection for Run combined or by summarization for Run separately).

Table 5 Results given in percentage with root-mean-square variances for automated runs.

Problem	Student enrollments	Time preferences	Room preferences	Distribution preferences
pu-spr07-llr	98.86±0.06	93.20±0.94	86.90±1.49	72.22±7.03
pu-fal07-llr	99.00±0.09	89.49±0.89	78.05±1.77	93.56±5.85
pu-spr07-ms	99.62±0.05	75.45±2.91	75.96±1.89	48.39±3.49
pu-fal07-ms	99.62±0.03	71.88±2.76	86.33±4.39	63.68±2.59
pu-spr07-cs	97.15±0.25	73.91±3.65	100.00±0.00	33.33±0.00
pu-fal07-cs	98.53±0.13	76.40±4.51	83.49±5.56	89.67±9.99
pu-spr07-ecet	98.26±0.23	90.63±2.18	100.00±0.00	100.00±0.00
pu-fal07-ecet	97.95±0.40	92.45±2.42	100.00±0.00	100.00±0.00
pu-spr07-sa	99.30±0.08	96.34±1.60	100.00±0.00	48.33±9.13
pu-fal07-sa	99.29±0.14	95.29±1.59	85.71±9.93	100.00±0.00
pu-spr07-cfs	97.66±0.16	89.74±2.68	82.22±5.37	82.50±18.74
pu-fal07-cfs	98.12±0.10	97.14±2.03	85.00±23.30	92.67±1.36
pu-spr07-vpa	97.08±0.13	88.62±4.12	76.67±10.61	100.00±0.00
pu-fal07-vpa	96.79±0.00	0.00±0.00	100.00±0.00	100.00±0.00
pu-spr07-lab	99.39±0.03	94.08±1.13	69.19±1.09	50.30±5.12
pu-fal07-lab	97.71±0.04	84.73±1.88	44.25±3.69	38.15±3.85
pu-spr07-c8	98.16±0.07	89.91±0.82	75.79±1.19	56.58±3.02
pu-fal07-c8	98.55±0.09	86.62±0.79	70.43±1.24	54.18±2.24

The results described here show different aspects of the solution approach. First, it is important to note that results are provided for cases when humans applied the system to solve particular problems ('Final' column). It is clear that these results are slightly worse than results reported in other experiments since they also include subsequent use of the solver on interactive problems. The second set of results ('Run separately' column) presents values obtained from segregated runs for the particular problems. This reflects the actual process used for timetabling. It is worthwhile to compare these results with another set of experiments ('Run combined' column) where the c8 data instances were solved. It is interesting to see that results from both runs are comparable, and similar values for particular criteria were obtained. Importantly, this experiment validates the scalability of the approach since the problems with almost 2,500 classes are truly large (for example, the International Timetabling Competition considered problems with at most 400 classes with simplified problem characteristics in the similar post enrollment track).

The last experiment ('Bound' column) indicates the difficulty of problems with respect to particular criteria. It is clear that time preferences for classes (instructors) together with the room preferences were often sacrificed in combined and separate runs to obtain reasonable quality in student conflict minimization.

Looking at specific values of root-mean-square variances in Table 5, it is clear that there is a higher value for some of the criteria than would be normally be expected. This is because of the small number of corresponding constraints for the given problems.

Another experiment that should be considered shows the importance of conflict-based statistics. In the run without conflict-based statistics, only 98.2±0.09% out of 891 domain variables in the pu-fal07-llr data instance were assigned in 10 runs each taking 3 hours. Although other optimization criteria (student enrollments 99.08±0.05%, time preferences 99.34±0.55%, room preferences 72.5±1.71% and distribution preferences 94.14±5.15%) were satisfactorily fulfilled for the classes that were placed, it was not possible to satisfy all hard constraints and assign all variables/classes.

8.2 Minimal Perturbation Problem

Experiments shown in Figure 9 give results for the minimal perturbation problem on large lecture problems from Spring 2007 and Fall 2007. These data sets represent complex instances of the considered problem types as noted in Section 2. For both data sets, pu-spr07-llr and pu-fal07-llr, the best solution was taken from a solution to the initial problem and 50 different problems containing randomly generated time change requirements on classes. The graphs give results showing the number of additional changes required as a result of the imposed changes. Since time changes are of interest, the distance function considered corresponds to $\Phi_{\text{time}}(\delta, v/d)$. More precisely, the distance function has been slightly modified and includes the number of students affected by a time change to prefer changes of classes with a smaller number of students.

Each run was performed five times (30 minutes per run) and average results are displayed. Graphs in the first row present how the increase in the number of classes requiring a time change influenced the number of additional classes whose time placement was changed due to requested modifications. The second row presents the relationship between the number of students affected by the imposed change and the additional students whose class placements were modified to accommodate the requested time changes. The last row summarizes the changes in four problem criteria based on the increasing number of classes with a time change. In all cases, a very small increase in the number of additional classes and students grows to a much larger increase corresponding with the number of requested modifications. Also, it is clear that satisfaction of a problem's solution criteria degrades slightly as more changes are involved since the new criterion of minimal changes must be considered. Results for the Fall 2007 data sets are slightly worse, but this respects the characteristics of the problem—given the same set of classrooms, there are 803 classes for Spring 2007 and 891 classes for Fall 2007 (this is typical for each year).

8.3 Interactive Problem

The last set of experiments, shown in Table 6, gives results for the interactive solver applied on large lecture problems pu-fal07-llr and pu-spr07-llr and their solutions produced by human schedule managers (see column 'Final' at Table 4). The maximal depth M was set to 3 (given class must change and 2 additional changes has been allowed). The results are computed as the average over all runs of the interactive solver on every class in the problem that is not fixed in both time and room (a change in the placement for such a class is strictly prohibited by the problem definition). Results for runs with common time limit of 5 seconds are presented in the third and fifth columns while the second and fourth columns report results achieved for runs of the interactive solver without any time limit.

Overall, the interactive solver was able to find an optimal suggestion within 5 seconds in more than half of the cases and came close in the remaining cases (quality of the objective function was on average worse by 0.3 % for Fall 2007 and by 0.2 % for Spring 2007). Note that the original solution was produced by humans, who do not necessarily select optimal solutions. This allowed for improvement in the quality of the objective function in further runs of the solver. There were only 4 cases (corresponding to 0.7 % of cases) in Fall 2007 problem and none in Spring 2007 where it was impossible to find a suggestion within the time limit even though there was at least one. The average time spent by the interactive solver roughly corresponds to the number of backtracks made (5 seconds versus no time limit), however it was possible to compute more than 75 % of all the suggestions.

9 Conclusions

Based on the results of this work, it is clear that complex university course timetabling problems can be solved at a level where these solutions are of practical use in the real world. A deeper insight into such a solution for Purdue University has been provided here starting from translation of a course structure into the set of classes related by specific constraints,

Table 6 Results for the interactive solver.

Problem	pu-fal07-llr		pu-spr07-llr	
Classes	891		803	
Classes fixed in time & room (%)	31.0		33.8	
Time limit (s)	–	5	–	5
No suggestion found (%)	1.6	2.3	0.8	0.8
Optimal suggestion found (%)	98.4	51.5	99.2	67.0
Complete space explored	98.4	21.5	99.2	33.3
Number of suggestions	232.8	174.9	228.6	184.5
Number of backtracks	66367.9	2886.9	13949.1	2592.0
Time spent (s)	128.6	4.7	39.9	4.2
Improvements in objective function (%)	+1.1	+0.8	+0.9	+0.7

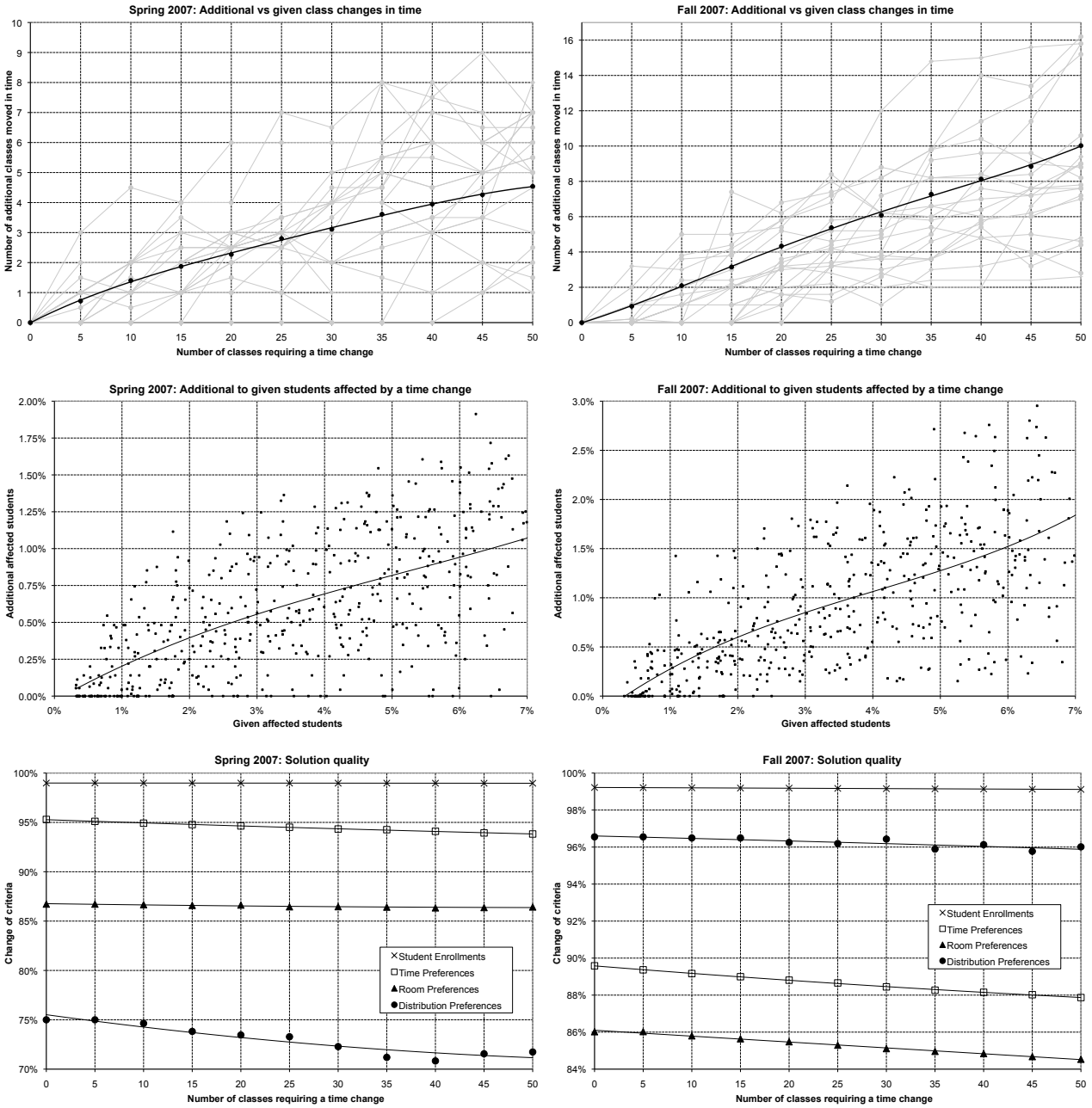

Fig. 9 Results for minimal perturbation problem for data sets pu-spr07-llr and pu-fal07-llr.

continuing with the formal description of different timetabling problems through their constraint models, and finally giving formal specification and justification of the algorithms developed.

Even though this paper describes the solution of a concrete timetabling problem, both the problem formalization and the solution methodologies are general and can be applied to solve other problems defined using weighted constraint satisfaction. The formal problem specification allows a precise understanding of the problems solved as well as giving a uniform description of distinct static and dynamic problems.

Iterative forward search has been described as a generic procedure extendable through the specification of the cost function and the comparator. Together with conflict-based statistics, it allows the solution of complex timetabling problems. A variant of the branch and bound algorithm provides appropriate solution methodology for time limiting computation within interactive solver.

The course timetabling problem represents the base problem whose solution was described here. To provide a com-

plete solution of the timetabling problem, student sectioning must also be taken into account. The work described here allows a natural extension of this process as has already been published by Müller and Murray (2008). It is notable that an extension of the algorithms described here creates the basis for the sectioning solver.

In addition to student sectioning, other significant work has been done related to the results published here. As noted in an earlier section, the timetabling solver developed by one of the co-authors (Müller, 2008) for the International Timetabling Competition 2007 was a finalist in all three and winner of two tracks. Importantly, the solver was built on a similar constraint model and applied iterative forward search for solution construction and additional local search methods for solution improvement. Last, but not least, another resulting work is an examination timetabling solver for Purdue University which has been applied to generate examination timetables for the two most recent terms.

In conclusion, it is clear that the methodology described here is applicable for solving a wide range of timetabling problems. This has been verified by continued use of the system to create all course timetables at Purdue University.

Acknowledgements Hana Rudová was supported by the Ministry of Education, Youth and Sports of the Czech Republic under the research intent No. 0021622419 and by the Grant Agency of the Czech Republic with grant No. 201/07/0205.

References

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

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

Pasquale Avella and Igor Vasil'ev. A computational study of a cutting plane algorithm for university course timetabling. *Journal of Scheduling*, 8:497–514, 2005.

Don Banks, Peter van Beek, and Amnon Meisels. A heuristic incremental modeling approach to course timetabling. In *Proceedings of the 12th Biennial Conference of the Canadian Society for Computational Studies of Intelligence on Advances in Artificial Intelligence*, pages 16–29. Springer-Verlag LNCS 1418, 1998.

Philippe Baptiste and Claude Le Pape. A theoretical and experimental comparison of constraint propagation techniques for disjunctive scheduling. In Chris S. Mellish, editor, *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 600–606. Morgan Kaufmann, 1995.

Roman Barták, Tomáš Müller, and Hana Rudová. A new approach to modeling and solving minimal perturbation problems. In *Recent Advances in Constraints*, pages 233–249. Springer Verlag LNAI 3010, 2004.

Christian Bessiere. Constraint propagation. In Rossi et al. (2006), pages 29–83.

Camille Beyrouthy, Edmund K. Burke, Barry McCollum Dario Landa-Silva, Paul McMullan, and Andrew J. Parkes. The teaching space allocation problem with splitting. In Burke and Rudová (2007), pages 228–247.

Camille Beyrouthy, Edmund K. Burke, Dario Landa-Silva, Barry McCollum, Paul McMullan, and Andrew J. Parkes. Towards improving the utilisation of university teaching space. *Journal of the Operational Research Society*, 60: 130–143, 2009.

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

Kenneth N. Brown and Ian Miguel. Uncertainty and change. In Rossi et al. (2006), pages 731–760.

Peter Brucker and Sigrid Knust. Resource-constrained project scheduling and timetabling. In Burke and Erben (2001), pages 277–293.

Edmund Burke and Michael Carter, editors. *Practice and Theory of Automated Timetabling II*. Springer-Verlag LNCS 1408, 1998.

Edmund Burke and Patrick De Causmaecker, editors. *Practice and Theory of Automated Timetabling IV*. Springer-Verlag LNCS 2740, 2003.

Edmund Burke and Wilhelm Erben, editors. *Practice and Theory of Automated Timetabling III*. Springer-Verlag LNCS 2079, 2001.

Edmund Burke and Michel Gendreau, editors. *7th International Conference on the Practice and Theory of Automated Timetabling*. Université de Montréal, Montréal, Canada, 2008.

Edmund Burke and Peter Ross, editors. *Practice and Theory of Automated Timetabling*. Springer-Verlag LNCS 1153, 1996.

Edmund Burke and Hana Rudová, editors. *Practice and Theory of Automated Timetabling VI*. Springer-Verlag LNCS 3867, 2007.

Edmund Burke and Michael Trick, editors. *Practice and Theory of Automated Timetabling V*. Springer-Verlag LNCS 3616, 2005.

Edmund K. Burke and Sanja Petrovic. Recent research directions in automated timetabling. *European Journal of Operational Research*, 140:266–280, 2002.

Hadrien Cambazard, Fabien Demazeau, Narendra Jussien, and Philippe David. Interactively solving school timetabling problems using extensions of constraint programming. In Burke and Trick (2005), pages 190–207.

Hadrien Cambazard, Emmanuel Hebrard, Barry O'Sullivan, and Alexandre Papadopoulos. Local search and constraint programming for the post-enrolment-based course timeta-

bling problem. In Burke and Gendreau (2008).

Michael W. Carter. A comprehensive course timetabling and student scheduling system at the University of Waterloo. In Burke and Erben (2001), pages 64–82.

Michael W. Carter and Gilbert Laporte. Recent developments in practical course timetabling. In Burke and Carter (1998), pages 3–19.

Yves Caseau and François Laburthe. Cumulative scheduling with task intervals. In Michael Maher, editor, *Proceedings of the 1996 Joint International Conference and Symposium on Logic Programming*, pages 363–377. MIT Press, 1996.

Patrick De Causmaecker, Peter Demeester, and Greet Vanden Berghe. A decomposed metaheuristic approach for a real-world university timetabling problem. *European Journal of Operational Research*, 195(1):307–318, 2009.

Andrew M. Cheadle, Warwick Harvey, Andrew J. Sadler, Joachim Schimpf, Kish Shen, and Mark G. Wallace. ECL^{i}PS^e: A tutorial introduction. Technical Report IC-Parc-03-1, IC-Parc, Imperial College London, 2003.

Marco Chiarandini, Mauro Birattari, Krzysztof Socha, and Olivia Rossi-Doria. An effective hybrid algorithm for university course timetabling. *Journal of Scheduling*, 9 (6):403–432, 2006.

Marco Chiarandini, Chris Fawcett, and Holger H. Hoos. A modular multiphase heuristic solver for post enrolment course timetabling. In Burke and Gendreau (2008).

Romuald Debruyne and Christian Bessière. Domain filtering consistencies. *Journal of Artificial Intelligence Research*, 14:205–230, 2001.

Rina Dechter. *Constraint Processing*. Morgan Kaufmann Publishers, 2003.

Rina Dechter and Daniel Frost. Backjump-based backtracking for constraint satisfaction problems. *Artificial Intelligence*, 136(2):147–188, 2002.

Luca Di Gaspero and Andrea Schaerf. Neighborhood portfolio approach for local search applied to timetabling problems. *Journal of Mathematical Modeling and Algorithms*, 5(1):65–89, 2006.

Luca Di Gaspero, Barry McCollum, and Andrea Schaerf. The second international timetabling competition (ITC-2007): Curriculum-based course timetabling (track 3). Technical Report QUB/IEEE/Tech/ITC2007/Curriculum-CTT/v1.0, University, Belfast, United Kingdom, 2007.

Abdallah Elkhyari, Christelle Guéret, and Narendra Jussien. Solving dynamic timetabling problems as dynamic resource constrained project scheduling problems using new constraint programming tools. In Burke and Causmaecker (2003), pages 39–59.

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

Matthew L. Ginsberg. Dynamic backtracking. *Journal of Artificial Intelligence Research*, 1:23–46, 1993.

Christelle Guéret, Narendra Jussien, Patrice Boizumault, and Christian Prins. Building university timetables using constraint logic programming. In Burke and Ross (1996), pages 130–145.

Martin Henz and Jörg Würtz. Using Oz for college timetabling. In Burke and Ross (1996), pages 162–177.

A. Hertz. Tabu search for large scale timetabling problems. *European Journal of Operational Research*, 54(1):39–47, 1991.

Narendra Jussien and Olivier Lhomme. Local search with constraint propagation and conflict-based heuristics. *Artificial Intelligence*, 139(1):21–45, 2002.

Philipp Kostuch. The university course timetabling problem with a 3-phase approach. In Burke and Trick (2005), pages 109–125.

Gyuri Lajos. Complete university modular timetabling using constraint logic programming. In Burke and Ross (1996), pages 146–161.

Javier Larrosa and Thomas Schiex. Solving weighted CSP by maintaining arc consistency. *Artificial Intelligence*, 159(1–2):1–26, 2004.

Rhyd Lewis. A time-dependent metaheuristic algorithm for Track-2 of the Second international timetabling competition. In Burke and Gendreau (2008).

Rhydian Lewis. A survey of metaheuristic-based techniques for university timetabling problems. *OR Spectrum*, 30(1): 167–190, 2008b.

Rhydian Lewis, Ben Paechter, and Barry McCollum. Post enrolment based course timetabling: A description of the problem model used for track two of the second international timetabling competition. Cardiff Working Papers in Accounting and Finance A2007-3, Cardiff Business School, Cardiff University, 2007.

Alan K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8(1):99–118, 1977.

Alfred Mayer, Clemens Nothegger, Andreas Chwatal, and Günther R. Raidl. Solving the post enrolment course timetabling problem by ant colony optimization. In Burke and Gendreau (2008).

Barry McCollum. A perspective on bridging the gap between theory and practice in university timetabling. In Burke and Rudová (2007), pages 3–23.

Barry McCollum, Andrea Schaerf, Ben Paechter, Paul McMullan, Rhyd Lewis, Andrew J. Parkes, Luca Di Gaspero, Rong Qu, and Edmund K. Burke. Setting the research agenda in automated timetabling: The second international timetabling competition. *INFORMS Journal on Computing*, 2009. Accepted.

Steven Minton, Mark D. Johnston, Andrew B. Philips, and Philip Laird. Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58:161–205, 1992.

Tomáš Müller. *Constraint-based Timetabling*. PhD thesis, Charles University in Prague, Faculty of Mathematics and Physics, 2005.

Tomáš Müller. ITC2007 solver description: A hybrid approach. In Burke and Gendreau (2008).

Tomáš Müller and Roman Barták. Interactive timetabling: Concepts, techniques, and practical results. In Edmund Burke and Patrick De Causmaecker, editors, *PATAT 2002 — Proceedings of the 4th international conference on the Practice And Theory of Automated Timetabling*, pages 58–72, 2002.

Tomáš Müller and Keith Murray. Comprehensive approach to student sectioning. In Burke and Gendreau (2008).

Tomáš Müller, Roman Barták, and Hana Rudová. Conflict-based statistics. In Jens Gottlieb, Dario Landa-Silva, Nysret Musliu, and Eric Soubeiga, editors, *EU/ME Workshop on Design and Evaluation of Advanced Hybrid Meta-Heuristics*. University of Nottingham, 2004.

Tomáš Müller, Roman Barták, and Hana Rudová. Minimal perturbation problem in course timetabling. In Burke and Trick (2005), pages 126–146.

Keith Murray, Tomáš Müller, and Hana Rudová. Modeling and solution of a complex university course timetabling problem. In Burke and Rudová (2007), pages 189–209.

Djamila Ouelhadj, Sanja Petrovic, and Rong Qu. A survey of dynamic scheduling in manufacturing systems. *Journal of Scheduling*, 12(4):417–431, 2009.

Sanja Petrovic and Edmund K. Burke. University timetabling. In Joseph Y-T. Leung, editor, *The Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, chapter 45. CRC Press, 2004.

Sylvain Piechowiak, Jingxua Ma, and René Mandiau. An open interactive timetabling tool. In Burke and Trick (2005), pages 34–50.

Jean-Charles Régin. A filtering algorithm for constraints of difference in CSPs. In *Proceedings of the Eleventh National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference*, pages 362–367. AAAI Press/MIT Press, 1994.

Vincent Robert and Alain Hertz. How to decompose constrained course scheduling problems into easier assignment type subproblems. In Burke and Ross (1996), pages 364–373.

Francesca Rossi, Peter van Beek, and Toby Walsh, editors. *Handbook of Constraint Programming*. Elsevier, 2006.

Hana Rudová and Keith Murray. University course timetabling with soft constraints. In Burke and Causmaecker (2003), pages 310–328.

Hani El Sakkout and Mark Wallace. Probe backtrack search for minimal perturbation in dynamic scheduling. *Constraints*, 4(5):359–388, 2000.

Hani El Sakkout, Thomas Richards, and Mark Wallace. Minimal perturbation in dynamic scheduling. In Henri Prade, editor, *Proceedings of the 13th European Conference on Artificial Intelligence (ECAI-98)*, pages 504–508. John Wiley & Sons, 1998.

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

Andrea Schaerf. A survey of automated timetabling. *Artificial Intelligence Review*, 13(2):87–127, 1999.

Thomas Schiex, Hélène Fargier, and Gérard Verfaillie. Valued constraint satisfaction problems: Hard and easy problems. In Chris S. Mellish, editor, *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 631–639. Morgan Kaufmann, 1995.

Katja Schimmelpfeng and Stefan Helberg. Application of a real-world university-course timetabling model solved by integer programming. *OR Spectrum*, 29(4):783–803, 2007.

Gerhard Schrimpf, Johannes Schneider, Hermann Stamm-Wilbrandt, and Gunter Dueck. Record breaking optimization results using the ruin and recreate principle. *Journal of Computational Physics*, 159(2):139–171, 2000.

Linda G. Shapiro and Robert M. Haralick. Structural descriptions and inexact matching. *IEEE Transactions Pattern Analysis Machine Intelligence*, 3:504–519, 1981.

Peter van Beek. Backtracking search algorithms. In Rossi et al. (2006), pages 85–134.

Gérard Verfaillie and Narendra Jussien. Constraint solving in uncertain and dynamic environments – a survey. *Constraints*, 10(3):253–281, 2005.

Christos Voudouris and Edward Tsang. Guided local search. In Fred Glover and Gary A. Kochenberger, editors, *Handbook of Metaheuristics*, chapter 7. Kluwer Academic Publishers, 2003.

George M. White and Junhan Zhang. Generating complete university timetables by combining tabu search with constraint logic. In Burke and Carter (1998), pages 187–198.