MISTA 2013

Real-life Examination Timetabling

Tomáš Müller

Abstract An examination timetabling problem at a large American university is presented. Though there are some important differences, the solution approach is based on the ITC 2007 winning solver which is integrated in the open source university timetabling system UniTime. In this work, nine real world benchmark data sets are made publicly available and the results on four of them are presented in this paper. A new approach to further decreasing the number of student conflicts by allowing some exams to be split into multiple examination periods is also studied.

1 Introduction

Examination timetabling is a well researched and important educational timetabling problem (Qu et al, 2009). It consists of individual meetings (*examinations*) that are to be assigned in time and space in a way that allows all the participants (students and instructors) to attend all the meetings that they require. The time is usually split into a set of non overlapping time slots (*examination periods*) of equal or varying length, and each examination needs to be assigned into one period that is of the same length or longer than the exam. Similarly, an examination needs to take place in a room that is big enough and there is usually only one examination in a room during one examination period. Besides direct conflicts (when a student or instructor is required to attend two examinations that are timetabled in same examination period) that need to be avoided or minimized, there are various other optimization criteria that need to be considered. For instance, it is often desired to avoid cases when a student needs to take an exam during two consecutive examination periods or having too many examinations during one day.

In this paper, we discuss a particular examination problem that is quite common at large American universities and that has been implemented in UniTime[1]. UniTime is a comprehensive university timetabling application that covers both post-enrollment (Rudová et al, 2011) and curriculum-based (Müller and Rudová, 2012)

Tomáš Müller
Space Management and Academic Scheduling, Purdue University
501 Northwestern Avenue, West Lafayette, USA
E-mail: muller@unitime.org

[1] `http://www.unitime.org`

course timetabling, examination timetabling, student scheduling (Müller and Murray, 2010), and event management. Also, nine real world instances from Purdue University are introduced in this work and made publicly available for benchmarking. This set covers almost 5 years of experience using UniTime for examination timetabling at the university (initially used in Fall 2008).

The examination problem at Purdue University is quite large. It consists of about 2,000 examinations, attended by up to 34,000 students with 120,000 student enrollments, placed in 29 distinct examination periods and using about 350 rooms. It is also quite hard to solve, as it is often impossible to find a complete timetable that has no direct conflicts. There are also a few, but substantial, differences to what has been studied so far (McCollum et al, 2012). For instance, multiple examinations in one room during the same period are not allowed, however, some of the large examinations require being split between multiple (usually up to 4) rooms. Also, two types of room seating are considered. An examination may either require a normal (class-like) seating or a special examination seating arrangement. This means that each room has two capacities: normal seating (usually equal to the number of chairs in a room) and examination seating (usually somewhere around half of the normal seating) which requires no two students sitting next to each other. Besides minimizing the number of direct conflicts, we also care about cases when a student has three or more examinations on a day, and about cases when two exams are back-to-back (a student must attend two exams that are on the same day in two consecutive periods). The more than two examinations on a day conflict is considered almost as bad as a direct conflict since in both cases a student may request an individual examination. In addition to these three types of student conflicts, there are several additional constraints and optimization criteria, like room availability and individual room and period preferences for an examination, that need to be considered.

It is worth mentioning that the largest problem (named PUR93) in Carter's famous and widely used benchmark data set (Carter et al, 1996) is based on older data from Purdue University. A solution to this problem using Carter's problem representation has never been used in practice at Purdue however.

The examination timetabling problem in UniTime is solved by an algorithm that is an extension of the winning algorithm in the second International Timetabling Competition[2] (ITC2007) using our Constraint Solver Library (Müller, 2009). It uses a constraint-based framework incorporating a series of algorithms based on local search techniques that operate over feasible, though not necessarily complete, solutions. In these solutions, some variables may be left unassigned, however, all hard constraints on assigned variables must be satisfied. The library is written in Java and is publicly available[3] under GNU's LGPL license.

The examination timetabling problem that has been studied in this work is described in the following section. The algorithm used to solve this problem is discussed next. The paper is concluded by a series of experiments on the four data sets collected at Purdue during the last four semesters.

[2] `http://www.cs.qub.ac.uk/itc2007`

[3] `http://cpsolver.googlecode.com`

2 Examination Timetabling Problem

The examination timetabling problem consists of a set of non-overlapping examination periods, a set of rooms, a set of examinations, and a set of distribution constraints.

Each **examination period** has defined a date, a start time and a length. It may also have a penalty associated with it. These penalties can be overridden on individual exams and are used to penalize certain unpopular examination periods (e.g., late evening or Saturday periods).

Each **room** has a normal and an examination seating capacity defined. A room can have GPS coordinates and there is a room availability matrix, stating during which periods a room can be used. Similarly as with examination periods, there can be a penalty associated with using the room at a certain period. Room availability and penalties can be expressed in a single two dimensional array containing infinite penalty for cases when a room is not available for examination timetabling at a certain period and an integer penalty otherwise. Note that we allow negative penalties which can be used to prefer a certain room or period over the others.

$$penalty_{rp}(r,p) = \begin{cases} \infty & \text{room } r \text{ not available at period } p \\ x \in \mathbb{Z} & \text{room } r \text{ has a penalty of } x \text{ at period } p \end{cases}$$

Each **examination** has defined a length, type of seating, maximal number of rooms into which it can be split (typically between 1 to 4), and a list of students enrolled in the exam. Additionally, each exam has a list of available periods and rooms associated with it. There can also be a penalty associated with each period or room of the exam for using the given period or room respectively.

All penalties in UniTime are based on the following scale. The room penalty on an examination is then computed by combining all such penalties set on individual rooms, buildings, room features and groups of rooms. For instance, there can be a penalty of -3 for a room with strongly preferred equipment that is located in a discouraged building.

-4 strongly preferred
-1 preferred
0 neutral / no penalty
1 discouraged
4 strongly discouraged
∞ prohibited

A **distribution constraint** is set between two or more exams and it can be either hard or soft. Hard constraints must be satisfied, soft constraints have a penalty that is incurred when the constraint is violated. Distribution constraints are of the following types:

- **Same Room:** Exams are required/expected to take place in the same room or rooms.
- **Different Room:** Exams are required/expected to take place in different rooms (no two exams of the constraint are to be placed in the same room).
- **Same Period:** Exams are required/expected to take place during the same periods.
- **Different Period:** Exams are required/expected to take place during different periods.

– **Precedence:** Exams are required/expected to take place in the order they are listed in the constraint. For instance if exam A is in the precedence constraint before exam B, it is to be placed in a period that precedes the period in which exam B is placed.

For more details, please see the XML format[4] that is used to store the input data as well as the resultant solution.

2.1 Hard Constraints

The following hard constraints need to be respected while assigning exams to examination periods and rooms:

– **One At A Time:** Only one exam can be placed in a room at any period.
– **Room Availability:** A room cannot be used during periods for which it is not available. This means that no exam can be placed in a room r during a period p when $penalty_{rp}(r, p) = \infty$.
– **Exam Availability:** An exam cannot be placed in a period or a room that is not available for the exam. This means that an exam e can not be placed in a period where $penalty_{ep}(e, p) = \infty$ or into a room r where $penalty_{er}(e, r) = \infty$.
– **Exam Size:** An exam must be placed in a room (or set of rooms) so that the overall seating capacity of the rooms equals or is greater than the number of students attending the exam, with respect to the requested seating type. This means that an examination e can be assigned in rooms $r_1, r_2, ..., r_n$ only when

$$\sum_{i=1,2,...,n} size_{seating(e)}(r_i) \geq |students(e)|$$

where $size_{seating(e)}(r_i)$ is the capacity of room r_i of the seating requested by the examination e (denoted $seating(e)$) and $students(e)$ is the list of students enrolled in the exam e.
– **Max Rooms:** Maximum number of rooms into which an exam can be split cannot be exceeded.
– **Hard Distributions:** Hard distribution constraints must be satisfied.

Exams that do require to take place in the same room must be placed in different examination periods as the **Same Room** constraint does not override the **One At A Time** hard constraint. Similarly, exams that are to be placed in the same period can not be placed in the same rooms.

2.2 Soft Constraints

During the search, besides looking for a complete solution (all exams are assigned to periods and rooms) that satisfies all hard constraints mentioned above, the following criteria are optimized. Each criterion has a weight associated with it (e.g., direct conflicts have typically much higher weight that back-to-back conflicts), the overall weighted sum of the following criteria is minimized.

[4] `http://www.unitime.org/exam_dataformat.php`

- **Direct Conflict:** A student is enrolled in two exams that are placed in the same period. If a student is attending three exams that are placed in the same period, it is counted as three direct conflicts (there is a direct conflict for every pair of two exams).
- **More Than Two Exams A Day Conflict:** A student is enrolled into three or more exams that take place on the same day.
- **Back-To-Back Conflict:** A student is enrolled into exams that are scheduled in consecutive periods (back-to-backs over an end of a day are not considered).
- **Period Penalty:** A penalization of an assignment of an exam into a period. It is a combination of the penalty $penalty_{ep}(e, p)$ set on the exam e for a particular period p and a default penalty that is set on the period itself $penalty_p(p)$.

$$PeriodPenalty(e, p) = 2 \times penalty_{ep}(e, p) + penalty_p(p)$$

  The constant 2 in the equation is used to give the individual examination penalty greater weight than the default penalty that is set on the period and applies to all examinations assigned in that period.
- **Room Penalty:** A penalization of an assignment of an exam into a room. It is a combination of the penalty $penalty_{er}(e, r_i)$ set on the examination e for a particular room r_i and the penalty $penalty_{rp}(r_i, p)$ set on the room r_i for the period p that is assigned. If an examination is assigned in two or more rooms $(r_1, r_2, ..., r_n)$, the room penalty for the exam is a sum of the room penalties for each of the assigned rooms.

$$RoomPenalty(e, p, r_1, r_2, ..., r_n) = \sum_{i=1,2,...,n} (2 \times penalty_{er}(e, r_i) + penalty_{rp}(r_i, p))$$

- **Distribution Penalty:** A penalty for a soft distribution constraint that is not satisfied. It is usually either 1 for preferred constraints or 4 for strongly preferred constraints.
- **Room Split:** A penalization for assigning an exam into multiple rooms. There is penalty of 1 of splitting an exam into two rooms, 4 into three rooms, 9 into four rooms. In general, it is $(n-1)^2$, where n is the number of rooms into which an exam is assigned.

$$RoomSplit(e, r_1, r_2, ..., r_n) = \begin{cases} 0 & n \leq 1 \\ (n-1)^2 & n > 1 \end{cases}$$

- **Room Split Distance:** A penalty for splitting an exam into rooms that are not close to each other. If an exam is split into two or more rooms, it is the average direct distance between these rooms. Distances are computed in meters between rooms that have GPS coordinates provided.

$$RoomSplitDistance(e, r_1, r_2, ..., r_n) = \frac{2}{n(n-1)} \sum_{i,j \in \{1,2,...,n\}; i<j} distance(r_i, r_j)$$

- **Room Size:** A penalty for using rooms that are too large for an exam. It is the difference between the size of the room (either exam or normal seating based on what is required by the exam) and the number of students enrolled in the exam. If an exam is assigned into two or more rooms, it is the difference between the size of the exam and the total sum of the room sizes in the seating type of the exam.

Between two exams, if it is desired to have a smaller exam in the smaller room (assuming that both rooms are big enough for both exams), the **Room Size** penalty can be raised to the power of some additional factor. For instance, with the room size factor $f = 1.1$, having an exam of 30 students in a room of 100 would create a penalization of $(100 - 30)^{1.1} = 107.06$, instead of just 70.

$$RoomSize(e, r_1, r_2, ..., r_n) = (\sum_{i=1,2,...,n} size_{seating(e)}(r_i) - |students(e)|)^f$$

- **Rotation Penalty:** An exam can have an average period placement (computed from previous examination problems) associated with it. If so, a penalty equal to the square root of the multiplication of the current period index (i.e., 1, 2, ..., 29) and the average period placement is associated with each exam. This effectively "rotates" the exam time placements by increasing the penalty on solutions in which exams with historically late period assignments are placed in late periods again.

$$RotationPenalty(e, p) = \sqrt{index(p) \times average(e)}$$

The average period placement $average(e)$ of an exam e is computed as the average of
  - the period index of the exam during the previous semester $index_p(e)$ and
  - the average period placement value the exam had at that time $average_p(e)$.

The period index of the exam during the previous semester is taken if the exam did not have the average period placement set at that time.

$$average(e) = \begin{cases} 0 & e \text{ is a new exam} \\ index_p(e) & e \text{ was new in previous term} \\ \frac{1}{2}index_p(e) + \frac{1}{2}average_p(e) & \text{otherwise} \end{cases}$$

For example, if an exam is going to be timetabled in Fall 2012 for the fifth time, the average period placement for the Fall 2012 examination timetabling problem is computed from the assignment of the previous four Fall[5] timetabling problems (2008, 2009, 2010, and 2011) as follows:

$$\begin{aligned} average_{12}(e) &= {}^{index_{11}(e)}/_2 + {}^{average_{11}(e)}/_2 \\ &= {}^{index_{11}(e)}/_2 + {}^{index_{10}(e)}/_4 + {}^{average_{10}(e)}/_4 \\ &= {}^{index_{11}(e)}/_2 + {}^{index_{10}(e)}/_4 + {}^{index_{09}(e)}/_8 + {}^{index_{08}(e)}/_8 \end{aligned}$$

This gives the more recent semesters more weight while keeping the older assignments in mind as well.

- **Large Exams Penalty:** One for each large exam (an exam with more than a certain number of students enrolled) that is placed on or after a certain period
- **Room Distance:** If an examination was created for a particular class (as opposed to the whole course), it is often desired for the exam to take place in the same room where the class took place (called *original room*). This is modeled by having a negative penalty of -4 on such a room and a penalty of -1 on all the other available rooms of the same building as the original room. If other than the original room is assigned to the exam, the **Room Distance** criterion measures the distance between the assigned room and the original room. If the exam is in multiple rooms, the average distance between the assigned rooms and the original room of the class is used as a penalty for not having the original room assigned.

[5] Since there are typically different courses offered in Spring and Fall, we usually consider only semesters of the same season for the average periods.

2.3 ITC 2007

Following are the main differences between the examination timetabling problem described in this paper and the examination problem of the first track of the second International Timetabling Competition (ITC 2007). See (Müller, 2009) for more details about the application of the described algorithm on the competition instances.

- Direct student conflicts are allowed, but minimized. In the competition problem it is not allowed for a student to have two exams at the same period.
- We do not allow two exams to share a room. On the other hand, an exam may be split among multiple rooms.
- Rooms have two capacities, based on the seating (examination or normal seating).
- The presented problem allows for room and period restrictions and penalties that can be set for individual exams. The room and period penalties of the competition problem applies to all exams using the given room and/or period.
- Besides of direct and back-to-back student conflicts which are present in both problems, instead of the more than two exams on a day, the competition problem has two exams on a day and a more general period spread. Period spread gives a penalty when a student has two exams over a specified number of consecutive periods.

While there is no direct mapping between the two problems, there are a lot of similarities as well. For the presented algorithm, the competition problems appear to be harder in finding a complete solution. This is mostly because direct student conflicts are not allowed. But there also seem to be less space available, which also corresponds with the inability to split an exam into multiple rooms if needed and the general need for some exams to share a room. On the other hand, the problems presented in this work are considerably larger, in many cases with fewer examination periods, and the exams seem to be more interlinked with students. As is shown in the experiments, while disallowing direct student conflicts is certainly possible, it results in a dramatic increase of back-to-back and more than two exams a day conflicts for students.

3 Algorithm

The search algorithm consists of several phases: In the first (construction) phase, a complete solution is found using an Iterative Forward Search (IFS) algorithm (Müller, 2005). This algorithm makes use of Conflict-based Statistics (CBS) (Müller et al, 2004) to prevent itself from cycling. In the next phase, a local optimum is found using a Hill Climbing (HC) algorithm. Once a solution can no longer be improved using this method, the Great Deluge (GD) technique (Dueck, 1993) is used. The GD algorithm is altered so that it allows some oscillations of the bound that is imposed on the overall solution value. The search ends after a predetermined time limit has been reached. The best solution found within that limit is returned.

3.1 Construction Phase

Initially, a complete solution is found using the Iterative Forward Search algorithm (see Figure 1). It starts with all variables (examinations) being unassigned (Line 2). During each iteration, an unassigned variable is selected (Line 4) and a value from its

domain is assigned to it (assignment of an examination period and a room or rooms). If this causes any violations of hard constraints with existing assignments, the conflicting variables are unassigned (Lines 5 - 7). For example, if there is another examination in the selected room at the selected period, that exam will be unassigned. More precisely, the function HARDCONFLICTS (Line 6) returns a subset γ of the solution σ, so that there are no hard constraints violated between the remaining assignments $\sigma \backslash \gamma$ and the new assignment $\{v/d\}$ of the selected variable v (Line 4) with the selected value d (Line 5). The search ends when all exams are assigned or a time limit is reached (Line 3).

```
1: function CONSTRUCTION()
2: σ = ∅
3: while not COMPLETE(σ) and not TIMEOUT do
4:         v = SELECTVARIABLE(σ)
5:         d = SELECTVALUE(σ, v)
6:         γ = HARDCONFLICTS(σ, v, d)
7:         σ = σ\γ ∪ {v/d}
8: return σ
```

Fig. 1 Pseudo-code of the construction algorithm.

The search is also parametrized by variable and value selection criteria (Lines 4 and 5). The examinations are ordered based on the following criteria

- *domain size:* examinations with the smallest domain are assigned first
- *highest degree:* among exams with the same domain size, examinations with the highest number of other exams that have at least one student in common with the exam are assigned first
- *examination size:* among examinations with the same domain size and the number of correlated exams, the largest examinations (in the number of students taking the exam) are assigned first

If there are two or more unassigned exams with the same domain size, number of correlated exams and students, one is chosen randomly. For the selected variable, a value whose assignment increases the overall cost of the solution the least is selected among values that violate the smallest number of hard constraints (i.e., the number of conflicting variables that need to be unassigned in order to make the problem feasible after assignment of the selected value to the selected variable is minimized). If there is a tie, one of these is selected randomly. It does not go through the whole domain, but the best possible room placement is selected for each of the available periods. If there is no room placement that does not violate any hard constraint (e.g., all the big enough rooms are already occupied by some other exams), a valid room assignment is selected randomly.

Conflict-based Statistics is used during this process to prevent repetitive assignments of the same values by memorizing conflicting assignments. Conflict-based Statistics is a data structure that memorizes hard conflicts which have occurred during the search together with their frequency and the assignments that caused them. More precisely, it is an array

$$CBS[V_a = v_a \ \rightarrow \ \neg V_b = v_b] = c_{ab}.$$

This means that the assignment $V_a = v_a$ has caused a hard conflict c_{ab} times in the past with the assignment $V_b = v_b$. Note that this does not imply that the assignments

$V_a = v_a$ and $V_b = v_b$ cannot be used together in the case of non-binary constraints. In the value selection criterion, each hard conflict is then weighted by its frequency, i.e., by the number of past unassignments of the current value of the conflicting variable caused by the selected assignment (see Figure 2).

```
1: function selectValue(σ, v)
2:  α = ∅, p_min = ∞, h_min = ∞
3:  for a in valuesToCheck(v) do
4:      γ = hardConflicts(σ, v, a)
5:      h = Σ_{w/b∈γ} (1 + CBS[v, a, w, b])
5:      p = penalty(σ\γ ∪ {v/a})
6:      if h < h_min or ( h = h_min and p ≤ p_min ) then
7:          if h < h_min or ( h = h_min and p < p_min ) then α = ∅, h_min = h, p_min = p
8:          α = α ∪ a
9:  a = random(α)
10: for w/b in hardConflicts(σ, v, a) do
11:     CBS[v, a, w, b] = 1 + CBS[v, a, w, b]
12: return a
```

Fig. 2 Pseudo-code of the value selection criterion using the Conflict-based Statistics.

If a complete solution is not found during the construction phase, it is very likely due to an inconsistency in the input data. In such a case, the content of the Conflict-based Statistics provides very valuable information back to the user which can be used to identify the problematic hard constraint or constraints that need to be relaxed.

3.2 Hill Climbing Phase

Once a complete solution is found, a Hill Climbing algorithm is used in order to find a local optimum (see Figure 3). In each iteration a change in the assignment of the current solution is proposed by random selection from a problem-specific neighborhood (Lines 4 - 5). The generated move is only accepted when it does not worsen the overall solution value (i.e., the weighted sum of violated soft constraints, Lines 6 - 8). Only changes that do not violate any hard constraints are considered. This rule applies to the following phase as well. Neighbor assignments are also generated consistently throughout all phases. That is, a problem specific neighborhood is selected randomly and is used to generate a random change in the current solution.

```
1: function hc(σ)
2:  S_best = penaly(σ), idle = 0
3:  while idle ≤ HC_idle and not timeout do
4:      n = random({PeriodChange, RoomChange, ExamSwap, RandomMove})
5:      δ = generate(σ, n),   S = penalty(σ ⊗ δ)
6:      if S ≤ S_best then
7:          σ = σ ⊗ δ
8:          if S < S_best then idle = 0, S_best = S
9:      idle = idle + 1
10: return σ
```

Fig. 3 Pseudo-code of the Hill Climbing algorithm.

The hill climbing phase is finished after a specified number HC_{idle} of idle iterations during which a solution has not improved (counted in variable *idle* on Figure 3). In all the following experiments, the parameter HC_{idle} was set to $25,000$.

3.3 Great Deluge Phase

The Great Deluge algorithm uses a bound B that is imposed on the overall value of the current solution that the algorithm is working with (see Figure 4). This means that the generated change is only accepted when the value of the solution after an assignment does not exceed the bound (Lines 6 - 8). The bound starts at the value

$$B = GD_{ub} \cdot S_{best}$$

where S_{best} is the overall value of the best solution found so far (Line 1), and GD_{ub} is a problem specific parameter (upper bound coefficient, set to 1.05 in all the experiments). The bound is decreased after each iteration (Line 9). This is done by multiplying the bound by a cooling rate GD_{cr} (set to 0.99999995 in all the experiments).

$$B = B \cdot GD_{cr}$$

The search continues until the bound reaches a lower limit equal to ${GD_{lb}}^{at} \cdot S_{best}$, where GD_{lb} is a parameter defining lower bound coefficient (set to 0.95 in all the experiments). When this lower limit is reached, the bound is reset back to its upper limit of ${GD_{ub}}^{at} \cdot S_{best}$ (Lines 10-11).

$$B < {GD_{lb}}^{at} \cdot S_{best} \;\Rightarrow\; B = {GD_{ub}}^{at} \cdot S_{best}$$

The parameter at is a counter starting at 1. It is increased by one every time the lower limit is reached and the bound is increased. It is also reset back to 1 when a previous best solution is improved upon. This helps the solver to widen the search when it cannot find an improvement, allowing it to get out of a deep local minimum.

```
1: function GD(σ)
2:  S_best = penaly(σ),  B = S_best · GD_ub,  ω = σ,  at = 1
3:  while not TIMEOUT do
4:        n = RANDOM({PERIODCHANGE, ROOMCHANGE, EXAMSWAP, RANDOMMOVE})
5:        δ = GENERATE(σ, n),  S = penalty(σ ⊗ δ)
6:        if S ≤ B then
7:              σ = σ ⊗ δ
8:              if S < S_best then S_best = S,  ω = σ,  at = 1
9:        B = B · GD_cr
10:       if B < GD_lb^at · S_best then
11:             B = GD_ub^at · S_best,   at = at + 1
12: return ω
```

Fig. 4 Pseudo-code of the Great Deluge algorithm.

3.4 Neighborhoods

The following neighborhoods are selected with equal probability during the hill climbing and great deluge phases. All of the proposed neighborhoods attempt to change an assignment of an exam or to swap the periods and/or rooms of two exams. If a change cannot be made, it systematically searches for an alternate change by selecting the next feasible assignment that follows the initially proposed change (i.e., it tries to use one of the subsequent periods or rooms in the variable's domain in the order they are loaded from the input file) rather than randomly generating and checking some other change.

3.4.1 Period Change

An examination and a new period are randomly selected. If no conflict results from assigning the selected exam to the new period, the new assignment is returned. The following periods are tried otherwise. The first available period is returned, another neighborhood is tried if no such period can be found. For each period, it tries to keep the existing room assignment, but if the room or rooms are not available, it tries to find the best possible available room or rooms instead. If no such room assignment can be found, the search continues with the following period.

The best possible room assignment is found by looking at all the rooms that are available to the exam at a certain examination period, with the smallest number of splits that does not violate any hard constraints (e.g., same room distribution constraint) and that minimizes the weighted sum of room penalty, number of violated soft same / different room distribution constraints and the room split distance.

3.4.2 Room Change

An examination and a new room are randomly selected. A new placement is generated by using the current period, the randomly selected room. If the selected room is not big enough, some of the current rooms (but not all) are added into the room placement. If no conflict results from assigning the selected exam into the new room assignment while keeping its period assignment, the new assignment is returned. The other combinations of rooms (if there are multiple using the randomly selected room and some of the current ones) are tried otherwise. If there is no other combination, the search continues with the following room.

3.4.3 Examination Swap

Two examinations are randomly selected. A new placement is generated by swapping periods of the two exams. For each exam, the best possible room placement is found. If the two exams are in the same period, it just tries to change the room assignments by looking for the best available room placement ignoring the existing room assignments of the two exams. If no conflict results from the swap the assignment is returned. The following exams of the second exam in the pair are tried for an exam swap otherwise.

3.4.4 Random Move

An examination and a new period are randomly selected. The best possible room assignment is found for the new period. If there is such a room assignment found and no conflict results from assigning the selected exam to the new period and room(s), the new assignment is returned. The following periods are tried otherwise.

4 Experiments

The following experiments were computed on an Apple MacBook Pro with a 3.06 GHz Intel Core 2 Duo processor and 8 GB RAM, running Mac OS X 10.7.3 and Java 1.6.0. In all the experiments the solver was given a time limit of two hours, and the average values of ten independent runs are shown.

4.1 Data Instances

In this work, nine data sets were made available for benchmarking. They are all final examinations problems from Purdue University, for each semester starting Fall 2008 when UniTime was used for examination timetabling at the university for the first time. All the datasets can be downloaded at `http://www.unitime.org/exam_datasets.php`. In this paper, the last four data sets are discussed. The results for the remaining data set are available on the web site.

Problem	Fall 2012	Spring 2012	Fall 2011	Spring 2011
Exams	1,864	1,798	1,914	1,866
Students	33,279	31,593	33,856	31,688
Enrollments	117,271	111,355	122,386	113,224
Distribution constraints	20	13	6	1
Exams fixed in time	57	63	58	99
Exams fixed in room	24	6	70	170
Large exams (600+)	22	20	18	17
Exams needing room split	10	9	20	13
Exams with original room	1,533	1,485	1,524	1,485
Density	3.28%	3.59%	3.46%	3.52%
Available periods	28.2 ± 0.4	28.0 ± 0.5	28.2 ± 0.4	27.5 ± 0.7
Available rooms	262.9 ± 6.1	265.8 ± 3.2	256.3 ± 9.8	234.7 ± 11.8
that are big enough	143.3 ± 38.5	143.3 ± 37.1	135.1 ± 40.1	126.6 ± 37.5

Table 1 Characteristics of the last four data sets.

Table 1 show the basic properties of the four latest datasets. The number of examinations, students, student examination enrollments and the distribution constraints for each data set are presented. All of the datasets have 29 examination periods, all 2 hours long starting at 8 am, 10:30am, 1 pm, 3:30 pm, and 7 pm each day from Monday till Saturday, except of the last period on Saturday (at 7 pm). All four Saturday periods

(8 am, 10:30am, 1 pm, and 3:30 pm) are strongly discouraged (*penalty* = 4), Friday afternoon periods (1 pm, 3:30 pm, and 7 pm) are discouraged (*penalty* = 1).

In all problems there are only a few distribution constraints and all are marked as hard (they cannot be violated). Moreover, in each problem, there are a few exams pre-assigned in time and/or room. The number of large exams, those exams for 600 or more students, is not very high either, which is good as they all should fit within the first 24 periods (excluding Saturday and the last period on Friday; the density between these exams is over 80%). There are also a few exams in each problem that need to be split into at least two rooms as there is no room big enough for the exam in its domain, either because of the number of students in the exam or due to room requirements that are put on the exam. The probability of a pair of two exams having at least one student in common is shown in the *density* row. The last three rows show the domain sizes of the problems, i.e., average number of available periods and rooms to an exam. The last column only counts the rooms that are big enough for the exam to fit in without the need to be split into multiple rooms. About 43% of all exams require examination (alternative) seating.

Fall 2012	All	$\geq$ 100 seats	$\geq$ 200 seats	$\geq$ 400 seats	$\geq$ 600 seats
Rooms	347	30 (16)	12 (8)	7 (3)	2 (2)
Exams	1,864 (819)	248 (179)	87 (69)	37 (32)	22 (21)
Density	3.3%	29.6%	60.0%	81.2%	83.6%

Table 2 Number of rooms and exams for Fall 2012 of certain size.

There are about 350 rooms in the problem. Table 2 shows the tightness of the problems for large examination rooms on the data set from Fall 2012. The numbers in brackets denote rooms with examination seating of the given size or examinations requiring examination seating with the given number of students. The density of common students between exams increases dramatically for larger sized exams.

Assigning exams to examination periods while avoiding direct conflicts between students can be seen as a graph coloring problem. Each examination can be represented as a vertex, with edges between two vertices where there is at least one student in common between the two exams. Each color then correspond to one examination period. The chromatic number of such a graph, which is the smallest number of colors needed to color all the vertices in a way that there are no two vertices with the same color connected with an edge, corresponds to the smallest number of examination periods that are needed to create an examination timetable with no direct conflicts, while still ignoring all the other hard constraints of the problem. Table 3 shows the chromatic numbers for the graphs of the four data sets. It is interesting to see how the chromatic number is decreased when edges between exams that contain only up to 1, 2 or 5 students in common are removed. With only 29 examination periods in the problem, and with other hard constraints thrown in, this makes finding a (direct) conflict free schedule close to impossible. And, as we will show in Section 4.3, even having such a conflict free timetable would be impractical as the number of other student conflicts (primarily the more than two exams on a day conflicts) grows too high.

Problem	All Edges	> 1 student	> 2 students	> 5 students
Fall 2012	27	21	17	12
Spring 2012	27	19	17	12
Fall 2011	29	20	16	13
Spring 2011	29	21	17	12

Table 3 Chromatic number of a graph with edges between exams with at least one, two, three, or six students in common.

4.2 Configurations

The data sets were tested on four different configurations. Solutions were measured using a weighted sum of the violated soft constraints. The appropriate weights are shown in the Table 4.

Weight	**production**	**base**	**color**	**split**
Direct Conflict	1,000	10^6	-	10^6
More Than 2 A Day	100	10^4	10^4	10^4
Back-To-Back	10	100	100	100
Period Penalty	1	1	1	1
Room Penalty	1	1	1	1
Room Split	10	10	10	10
Room Split Distance	0.01	0.01	0.01	0.01
Room Size	$(0.001x)^{1.1}$	$(0.001x)^{1.1}$	$(0.001x)^{1.1}$	$(0.001x)^{1.1}$
Rotation Penalty	0.0001	0.0001	0.0001	0.0001
Large Exams Penalty	$2.5 \cdot 10^6$	$2.5 \cdot 10^6$	$2.5 \cdot 10^6$	$2.5 \cdot 10^6$
Room Distance	0.0001	0.0001	0.0001	0.0001
Hard Constraint Violation	-	-	1,000	-
Exam Period Split	-	-	-	5,000

Table 4 Four weights configurations used in the experiments.

The **production** configuration contains the weights as they are used in practice by the University. The **base** configuration gives more weight to direct student conflicts to give a better comparison with the next two configurations. In the **color** configuration, the graph coloring solution is used to construct an initial assignment of exams to periods and rooms and it is not allowed to create a direct conflict. To be able to do this, however, it was necessary to add the ability to break other hard constraints, namely the ability to use a prohibited period or room by an exam or to break a distribution constraint. It was still not allowed to have multiple examinations in the same room at the same period, to exceed the room capacity, or to assign an exam into more than the given maximal number of rooms.

The last configuration (named **split**) attempts to minimize the number of direct conflicts by adding the ability to split an exam into two. In these runs, an additional neighborhood was used (along with all the other neighborhoods mentioned above) that allowed splitting a randomly selected examination into two, assigning each in a different period, and moving students freely between these two exams if the improvement in the weighted number of student conflicts (including direct, more than two on a day, and

back-to-backs) was higher than the penalization for splitting an exam **Exam Period Split**. If a randomly selected exam was already split, the neighborhood also tried to reshuffle the students between the two exams or merge the exams back together if the split was no longer needed (i.e., the weighted sum of student conflicts avoided by the split was smaller than the weight of the split).

4.3 Results

Results for all the runs are shown in the following tables. Table 5 shows the results for the four data sets using **production** configuration. This is the configuration that

Config: **production**	Fall 2012	Spring 2012	Fall 2011	Spring 2011
Direct Conflicts	79.7 ± 3.4	91.6 ± 5.8	107.2 ± 6.0	119.5 ± 6.5
More Than 2 A Day	345.2 ± 10.0	398.9 ± 11.9	448.2 ± 23.2	469.5 ± 7.9
Back-To-Back	4107.2 ± 74.5	4233.4 ± 51.5	4700.9 ± 94.5	4360.7 ± 65.9
Period Preference [%]	91.5 ± 0.3	91.5 ± 0.2	91.3 ± 0.3	91.0 ± 0.3
Room Preference [%]	74.3 ± 0.5	75.4 ± 0.5	70.1 ± 0.5	70.1 ± 0.4
Room Distance [m]	36.4 ± 2.1	35.2 ± 2.4	48.3 ± 2.9	48.9 ± 1.5
Room Split	43.0 ± 2.3	36.8 ± 1.9	94.6 ± 1.5	24.5 ± 1.4
Room Split Distance [m]	95.0 ± 11.3	69.4 ± 15.2	111.9 ± 12.9	48.1 ± 17.0
Room Size	72.3 ± 1.5	66.0 ± 2.8	52.7 ± 3.0	66.2 ± 1.9
Rotation Penalty	11.95 ± 0.05	12.09 ± 0.05	12.05 ± 0.07	11.95 ± 0.04
Large Exams Penalty	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0

Table 5 Results using **production** configuration. Average results for 10 independent runs are presented.

is used in practice at Purdue University. This gives a good proportion between direct and more than two exams on a day conflicts. The table shows the total number of direct student conflicts, more than two on a day conflicts and back-to-back conflicts. A percentage of the overall satisfaction of all the period and room penalties are displayed (100% would mean that all the examinations are assigned in a period or a room with the smallest penalty). The **period preferences** mostly correspond to discouraging Friday afternoon periods and strongly discouraging Saturday periods. The **room preferences** mostly correspond to assigning exams to their original room (a room where the class took place for all the class exams) or, if the room is not available, to the same building. The average distance to the original room if the original room was not assigned is shown in the **Room Distance** row. The number of exams being split into multiple rooms is very close to the absolute number of splits that are needed to fit all the big exams into available rooms. All the large exams were assigned before the last 5 periods.

The **Rotation Penalty** shows the average square root of the multiplication of the average period index and the assigned period index (both starting from 1). This number is minimized and it would be around 13.0 on average for an exam randomly assigned to a period with no regards to the rotation. It is a little less than a half of the number of examination periods (14.5 with 29 periods) as there are less exams timetabled during the discouraged Friday afternoon and strongly discouraged Saturday periods than on other (non discouraged) examination periods.

The effect of increasing direct conflict weight can be nicely seen in Table 6. The increase in the number of more than two on a day and back-to-back conflicts is most important. Impairment on other criteria is less profound.

Config: **base**	Fall 2012	Spring 2012	Fall 2011	Spring 2011
Direct Conflicts	32.7 ± 3.9	50.2 ± 2.9	53.9 ± 4.8	73.1 ± 4.4
More Than 2 A Day	344.8 ± 26.6	383.6 ± 18.7	545.2 ± 26.8	559.0 ± 20.3
Back-To-Back	4792.1 ± 151.2	4945.7 ± 147.5	5823.3 ± 86.6	5431.3 ± 157.9
Period Preference [%]	88.2 ± 0.4	87.5 ± 0.2	87.2 ± 0.4	87.7 ± 0.5
Room Preference [%]	72.4 ± 0.3	73.8 ± 0.5	69.0 ± 0.6	69.3 ± 0.7
Room Distance [m]	40.7 ± 1.6	35.5 ± 1.5	48.9 ± 3.4	51.1 ± 1.8
Room Split	48.5 ± 8.9	35.4 ± 4.2	58.5 ± 3.8	29.1 ± 2.8
Room Split Distance [m]	82.7 ± 16.0	73.9 ± 15.7	67.8 ± 11.0	54.0 ± 7.8
Room Size	70.0 ± 3.8	68.0 ± 1.6	51.6 ± 2.9	68.8 ± 2.2
Rotation Penalty	12.22 ± 0.04	12.42 ± 0.05	12.28 ± 0.04	12.30 ± 0.06
Large Exams Penalty	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0

Table 6 Results using **base** configuration. Average results for 10 independent runs are presented.

The configuration **color** with results in Table 7 was created in an attempt to bring the number of direct student conflicts down even more. While it was not allowed to

Config: **color**	Fall 2012	Spring 2012	Fall 2011	Spring 2011
Direct Conflicts	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
More Than 2 A Day	650.7 ± 38.0	663.3 ± 19.6	1152.6 ± 19.0	717.5 ± 20.6
Back-To-Back	6342.0 ± 133.5	6282.0 ± 94.5	6958.0 ± 145.1	6042.7 ± 82.0
Unavailable Period	12.7 ± 1.3	18.3 ± 0.9	17.4 ± 1.2	28.9 ± 1.3
Unavailable Room	10.8 ± 0.9	10.2 ± 1.0	22.6 ± 1.0	18.1 ± 1.8
Violated Distribution	2.8 ± 0.8	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
Period Preference [%]	85.8 ± 0.3	86.3 ± 0.4	86.5 ± 0.3	86.7 ± 0.4
Room Preference [%]	72.5 ± 0.4	73.5 ± 0.4	69.3 ± 0.9	69.5 ± 0.6
Room Distance [m]	38.6 ± 2.6	38.5 ± 1.5	54.3 ± 3.8	55.2 ± 1.9
Room Split	19.8 ± 9.7	14.4 ± 5.4	30.7 ± 6.0	13.8 ± 5.1
Room Split Distance [m]	150.5 ± 65.0	137.1 ± 50.8	159.4 ± 46.6	167.0 ± 39.9
Room Size	77.5 ± 2.3	68.8 ± 2.9	54.6 ± 2.2	68.9 ± 3.9
Rotation Penalty	12.33 ± 0.05	12.64 ± 0.05	12.56 ± 0.04	12.40 ± 0.04
Large Exams Penalty	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	2.0 ± 0.0

Table 7 Results using **color** configuration. Average results for 10 independent runs are presented.

create a direct conflict, it was allowed to assign an examination to a period or room that was not available for the exam or to break a hard distribution constraint. Unlike in all the other runs, the solver started from a graph coloring solution and each color was randomly assigned to a period. Best available room or rooms were assigned to each exam in the same order as the exams are taken during the construction phase of other runs. After the construction phase the search resumed as with all the other runs (i.e.,

with the hill climbing phase), but it was not allowed to create any direct conflict during the run. This means that only neighborhoods which did not create a direct student conflict were allowed.

While there were no direct conflicts in any run using the **color** configuration, the number of more than two exams on a day and back-to-back conflicts went up quite substantially. The number of more than two exams on a day almost doubled compared to the **base** runs and there were at least one thousand more back-to-backs in most of the runs. Note that the number of hard constraint violations somewhat corresponds to the number of direct conflicts we had in the runs using the **base** configuration. All things considered, it is very interesting to see the effect of requiring a (direct) conflict free examination schedule on the other criteria; however, the results are not usable in practice, especially because of the broken hard constraints and the number of more than two exams on a day. At Purdue, more than two exams on a day conflicts are almost as bad as direct conflicts as in both cases a student is allowed to request an individual examination, though some students do not request an individual examination because of the number of exams on a day. Unfortunately, since a student with a conflict goes directly to his / her professor, we have no statistics on how many students actually do request an individual exam, but the numbers we are getting with the **production** configuration seem to work quite well. Certainly, there are more complaints about having Saturday exams than on student examination conflicts.

A different approach to further decreasing the number of student conflicts was tried using the **split** configuration. Here the solver is allowed not only to split an exam between multiple rooms, but also to split it between multiple examination periods. It can be seen that the number of exams that were split roughly corresponds to the number of direct conflicts somewhere in between the **base** and **production** configurations. This is because most of the direct conflicts in the solutions are between exams that have one or two students in common. On the other hand, allowing the solver to move students with any conflict (including back-to-backs) in between the exams helps to decrease the number of more than two on a day and back-to-back conflicts quite substantially. Also, the solutions of the **split** configuration are very similar to the **base** configuration solutions in all the other criteria, which are also shown in the Table 8.

Config: **split**	Fall 2012	Spring 2012	Fall 2011	Spring 2011
Direct Conflicts	0.0 ± 0.0	7.0 ± 0.0	0.0 ± 0.0	1.0 ± 0.0
More Than 2 A Day	71.3 ± 11.6	90.9 ± 8.5	96.9 ± 39.0	87.9 ± 11.9
Back-To-Back	1802.7 ± 112.0	1934.9 ± 138.1	2001.2 ± 464.3	1721.5 ± 132.9
Period Splits	64.10 ± 3.54	75.60 ± 6.35	88.80 ± 9.13	83.30 ± 5.83
Period Preference [%]	88.6 ± 0.4	88.0 ± 0.3	87.8 ± 1.5	87.8 ± 0.3
Room Preference [%]	72.3 ± 0.7	73.8 ± 0.5	64.7 ± 11.3	67.6 ± 2.5
Room Distance [m]	40.4 ± 2.5	36.3 ± 2.0	68.4 ± 55.7	56.3 ± 11.1
Room Split	46.8 ± 3.6	37.1 ± 6.5	55.0 ± 12.8	31.2 ± 6.7
Room Split Distance [m]	41.0 ± 10.1	37.2 ± 12.6	63.2 ± 34.5	57.5 ± 19.2
Room Size	84.1 ± 6.3	74.3 ± 2.7	70.7 ± 14.9	79.6 ± 3.3
Rotation Penalty	12.19 ± 0.04	12.32 ± 0.05	12.34 ± 0.28	12.41 ± 0.08
Large Exams Penalty	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0

Table 8 Results using **split** configuration. Average results for 10 independent runs are presented.

The ability to split an examination in periods is interesting and gives very promising results. Besides the fact that the number of makeup exams (individual exams scheduled for students with a direct or more than two on a day conflict) seems to correspond with the automatically generated period splits, it can also be used to decrease the number of exams that are now offered on Saturday, while keeping the number of student conflicts on a manageable level. On the other hand, the need to offer certain exams during two distinct examination periods has some drawbacks as well. For instance, students on the second period can have some advantage over students taking the exam first.

Figure 5 shows how the quality of the computed solution improves during the search. The average results of 10 independent runs of the Fall 2012 problem using the **production** configuration are presented. The top most chart shows the **Overall Solution Value**, which is the sum of all the violated soft constraints, each weighted by the appropriate weight from the **production** configuration. This weighted sum is minimized during the search. Each of the following charts shows one of the objectives. The **Large Exams Penalty** is omitted from the list as this penalty was always zero across all the runs due to its high weight.

For the Fall 2012 problem and the **production** configuration, the construction phase did not take more than 10 seconds and the hill climbing phase was finished within a minute. We can see the hill climbing phase before the short plateau at the beginning of each chart. The hill climbing phase is particularly useful for setting the initial bound of the following great deluge phase and starting the phase from a good solution. Despite of that, it still takes about two minutes for the great deluge phase to find a better solution. But the improvements are quite dramatic, especially in the number of direct conflicts and more than two exams a day, which have the highest weights. The other criteria get quite disturbed by these improvements at first (as in many cases, decreasing the number of student conflicts goes against all the other criteria). However, as the bound goes down, the other criteria get improved as well and in about 10 minutes (when the bound reaches the lower limit for the first time) the solution looks quite good in all of its characteristics. In fact, the **Overall Solution Value** of the solution computed after the first 15 minutes is on average only about 6.5% higher than the overall value we get after the full two hours. On the other hand, we can see small but steady improvements in the solution quality all the way during the remaining search. The lower limit of the bound got reached 18 times on average during the search and the counter *at* that widens the limits for the bound (see Figure 4) did not get higher than 3.

The experiment also validates the current practice at Purdue University. While 15 to 30 minutes works well for all the trial runs that are done during the data entry (e.g, to validate that the entered requirements do not over-constraint the problem or create too many student conflicts), the two hour limit gives a good base for a production timetable. Once a production timetable is created and published, there is rarely a need for a new run. On the other hand, UniTime allows for manual modifications of the timetable in an interactive mode, where the solver does not make any decisions but it is used to provide alternatives and suggestions how to fix created problems. For instance, if an operator wants to move an exam into a different room, it will provide suggestions how to move the other affected exams around so that there are no hard constraints violated and the quality of the modified solution (especially the number of student conflicts) is maintained.

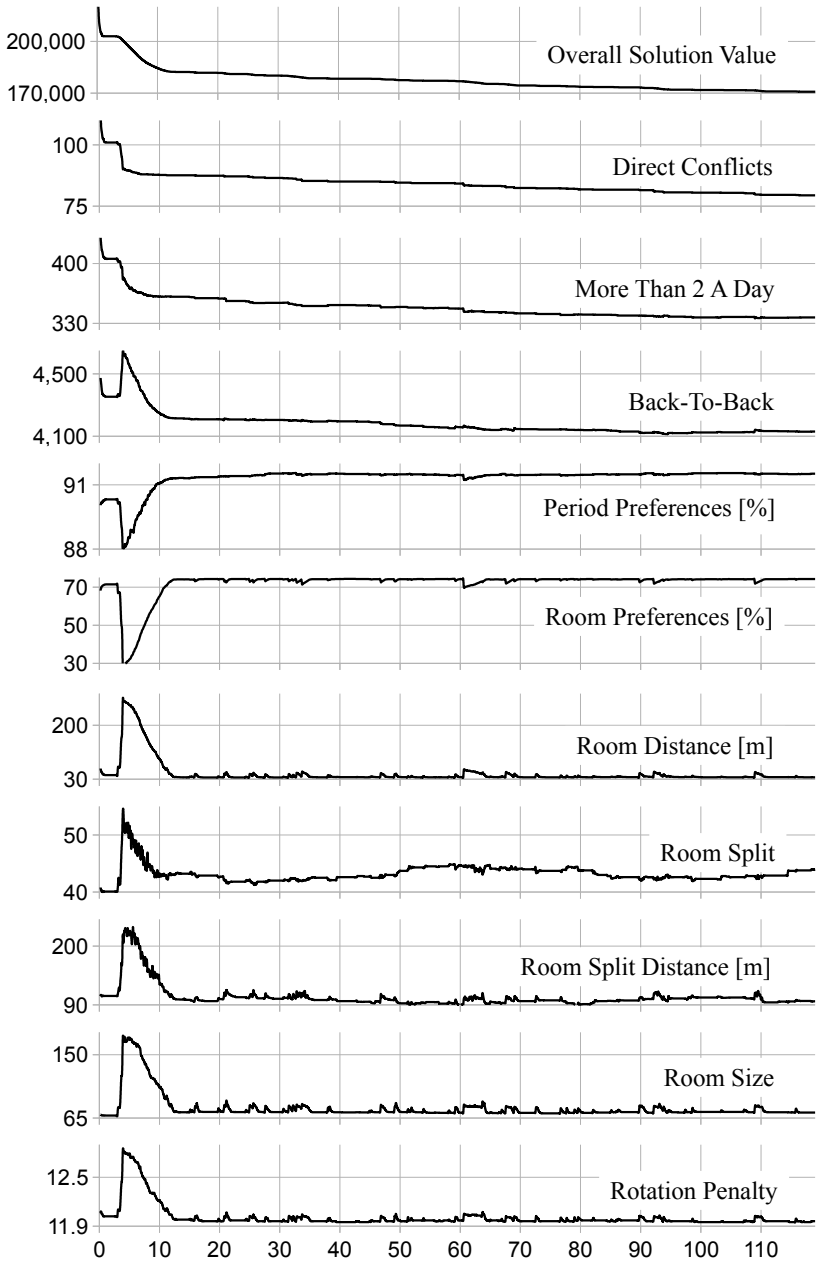

Fig. 5 Progression of the best computed solution during the search. Each chart represents one objective, the horizontal axe represents time in minutes. Average results of 10 independent runs on the Fall 2012 problem using **production** configuration are presented.

5 Conclusion

In this paper, we have presented a real world examination timetabling problem and an algorithm that was implemented in the university timetabling system UniTime.

Though we only have data from Purdue University at the moment, we believe it is applicable to many other American universities (especially the larger ones that need to deal with large exams offered to students across multiple curricula) and we plan to extend the benchmark suite that was created in this work with data from other institutions in the future. There are several other universities that are using UniTime for examination timetabling or that are in the process of adopting UniTime.

From the research perspective, the ability to split an exam into multiple examination periods seems to be very interesting. As for the future work, several other features have also been added recently to the examination solver; however, we do not have enough data to publish results at the moment. For instance, besides being able to split an exam into multiple rooms, it is now also possible to put multiple examinations into one room during the same examination period. This room sharing ability is controlled by a compatibility metrics that can be a part of the input data. For instance, only certain rooms can be shared between multiple examinations, only exams that are of the same length can share a room, and the room has to be big enough to hold all the exams that are placed in there at the same time.

References

Carter MW, Laporte G, Lee SY (1996) Examination timetabling: Algorithmic strategies and applications. Journal of the Operational Research Society 47(3):373–383, DOI 10.1057/jors.1996.37

Dueck G (1993) New optimization heuristics: The great deluge algorithm and the record-to record travel. Journal of Computational Physics 104:86–92

McCollum B, McMullan P, Parkes AJ, Burke EK, Qu R (2012) A new model for automated examination timetabling. Annals of Operations Research 194:291–315, DOI 10.1007/s10479-011-0997-x

Müller T (2005) Constraint-based timetabling. PhD thesis, Charles University in Prague, Faculty of Mathematics and Physics, URL `http://muller.unitime.org/phd-thesis.pdf`

Müller T (2009) Itc2007 solver description: a hybrid approach. Annals of Operations Research 172:429–446, DOI 10.1007/s10479-009-0644-y

Müller T, Murray K (2010) Comprehensive approach to student sectioning. Annals of Operations Research 181:249–269

Müller T, Rudová H (2012) Real-life curriculum-based timetabling. In: Proceedings of the 9th International Conference on the Practice and Theory of Automated Timetabling – PATAT 2012, pp 57–72

Müller T, Barták R, Rudová H (2004) Conflict-based statistics. In: Gottlieb J, Silva DL, Musliu N, Soubeiga E (eds) EU/ME Workshop on Design and Evaluation of Advanced Hybrid Meta-Heuristics, University of Nottingham

Qu R, Burke E, McCollum B, Merlot L, Lee S (2009) A survey of search methodologies and automated system development for examination timetabling. Journal of Scheduling 12:55–89, DOI 10.1007/s10951-008-0077-5

Rudová H, Müller T, Murray K (2011) Complex university course timetabling. Journal of Scheduling 14(2):187–207