A Conversion Scheme for Turning a Curriculum-based Timetabling Problem into a School Timetabling Problem

Kimmo Nurmi[1], Jari Kyngäs[1]
[1] Satakunta University of Applied Sciences
Tiedepuisto 3, 28600 Pori, Finland

Abstract. This article describes a conversion scheme for turning a curriculum-based timetabling problem into a school timetabling problem. The motivation for this paper is to give directions on how to solve problems lying between school timetabling and curriculum-based timetabling. The converted problem instances are solved using a previously published school timetabling algorithm. The algorithm finds a feasible solution within the given time limit for 12 of the 14 problems used in the 2nd International Timetabling Competition. This is probably the first paper that describes such a conversion scheme.

Keywords: *Timetabling, Conversion Scheme, Heuristic Search, Evolutionary Algorithms.*

1 Introduction

Several solution approaches to timetabling problems have been introduced in recent years. Most of the work has concentrated on examination, university and course timetabling (Schaerf 1999; Burke and Petrovic 2002; Burke et al. 2003; Burke et al. 2005; McCollum 2006; Burke and Rudová 2006). Timetabling researchers have obtained very promising and practicable results in these problem areas. *The 2nd International Timetabling Competition* organized and run by the eventMAP research Group at Queen's University (De Werra 1985) also concentrates on these problems. An important aim of the competition is to generate new approaches by attracting users from all areas of research. The second important aim is to close the gap that currently exists between research and practice within this important area of operational research.

This article describes *a conversion scheme* for turning a curriculum-based timetabling problem into a school timetabling problem. The 2nd International Timetabling Competition tackles the curriculum-based timetabling problem (McCollum 2007), which consists of the weekly scheduling of the lectures for several university courses within a given number of rooms and time periods. The conflicts between courses are set according to the curricula published by the University, not on the basis of enrolment data. The formulation of the problem applies to University of Udine (Italy) and to many Italian and International Universities, although it is slightly simplified with regard to the real problem to maintain a certain level of generality.

The school timetabling problem (Di Gaspero et al. 2007) has not been as extensively studied as the competition problems. We presented a *school timetabling algorithm* in (Nurmi 1998) and later extended it in (Nurmi and Kyngäs 2007). The original and extended algorithms were constructed to solve a school timetabling problem that arises at various Finnish school levels: comprehensive school, upper secondary school and higher education. The problem description is representative of many timetabling scenarios within the area of school timetabling.

The school timetabling problem differs quite substantially from the curriculum-based timetabling problem. However, the most important components are the same, so we can convert the curriculum-based timetabling problem to the school timetabling problem. We implement five conversion methods, and finally solve the converted problem using our original school timetabling algorithm. We believe that this is the first article that describes a conversion scheme to turn the university/course timetabling problem into a school timetabling problem.

2 Problem Description

In the curriculum-based timetabling problem *(the Competition problem)* we are given a number of teaching days in the week. Each day is split into a fixed number of timeslots, which is equal for all days. Each course consists of a fixed number of lectures to be scheduled in distinct periods (days and timeslots), attended by a given number of students and taught by a teacher. Each room has a capacity, expressed as the number of available seats. All rooms are equally suitable for all courses (if large enough). For each course there are a minimum number of days the course lectures should be spread over, moreover there are some periods for which the course cannot be scheduled. A curriculum is a group of courses whereby any pair of courses in the group has students in common. Based on the curricula, we have conflicts between courses and other soft constraints. The solution to the problem is the assignment of a period and a room to all lectures on each course.

In the school timetabling problem the timetable frame consists of *n* weeks, where $n = 1$ in most problems. Each week is split into a fixed number of days and each day into a fixed number of timeslots. A lecture is a predefined combination of a teacher, a student group, a room and the length of the lecture in timeslots. The sets of teachers, student groups and rooms are fixed. The student groups can have common students. In the Finnish problem each group should have at least a given number of periods in a day, but should not have more than another given number of periods. The rooms can be classified into certain room types. Some lectures (courses) can be scheduled to occur in the correct order during the week.

Table 1. The curriculum-based timetabling problem and the school timetabling problem.

Properties	Curriculum-based timetabling problem (Competition problem)	School timetabling problem (Finnish problem)
Timeslots	Weekly schedule, typically 5-6 days and 5-6 periods per day	Weekly schedule, typically 5 days and 8 periods per day
Teachers	Pre-assigned to courses	Pre-assigned to courses
	Unavailability constraints	Unavailability constraints
	No timeslot preferences	Timeslot preferences
Students	Students enroll on courses	Students either do not enroll on courses or only enroll on optional courses
Student groups	No student group concept	Students are preassigned to base groups, and also assigned to optional groups, which are built according to students' enrolments on optional courses. Some base/optional groups must be scheduled for different periods because they have common students.
	No unavailability constraints	Unavailability constraints
	No timeslot preferences	Timeslot preferences
Rooms	Each room has a capacity expressed in number of seats. All rooms are equally suitable for all courses.	No capacity constraints, but rooms with the same resources can be grouped together (room groups)
	No unavailability constraints	Unavailability constraints
Courses	A course is a pre-assigned combination of a teacher, a number of student enrolments and a total number of periods	A course is a pre-assigned combination of a teacher, a student group, a room and a total number of periods
	For each course there are a minimum number of days the lectures of the course should be spread over	No such constraint
	Conflicts between courses are set according to the curricula published by the university, not based on student enrolment data	Conflicts between courses are set according to the timetable published by the school
	No precedence constraint	Some courses are scheduled to occur in the correct order in the week
Lectures	A lecture is an instance of a course taking one or more periods	A lecture is an instance of a course taking a pre-assigned number of periods
Curricula	A curriculum is a group of courses where any pair of courses in the group has students in common	No curriculum concept
Solution	**Assignment of a period and a room to all lectures of each course**	**Assignment of a starting period to all lectures**

The problem is to assign predefined lectures to periods in such a way that no teacher, student group (if common students) or room is involved in more than one lecture in the same timeslot, and the other constraints are satisfied. The solution to the school timetabling problem is the assignment of a starting period to all lectures. The most common soft constraints are:

1. The timetable of a student group should have as few idle periods as possible.
2. The school day of some student groups should not start in the first period.
3. The school day of some student groups should end as early as possible.
4. The timetable of some teachers should have as few idle periods as possible.
5. For some teachers the preferred daily minimum and maximum number of periods are given.
6. Some teachers prefer not to be scheduled in certain periods.
7. Some teachers should only be scheduled on a limited number of days.
8. The lessons of a subject should be on different days (for the same student group).

Table 1 summarizes the curriculum-based timetabling problem (*CP*) and the school timetabling problem (*SP*). The five major differences are

1. SP has no concept for a curriculum.
2. CP has no concept for a student group.
3. In CP, students enroll on courses and room assignments should be made based on the number of student enrollments on each course.
4. In SP, rooms are pre-assigned to lectures.
5. In SP, both the number of lectures on a course and the number of periods in each lecture are pre-assigned, and cannot be dynamically changed as in CP.

Table 2. Hard and soft constraints of the competition problem and the corresponding construction of the school timetabling problem.

Hard constraints	Competition problem	Corresponding construction of the school timetabling problem
Course clashes	Two lectures on a course cannot take place in the same period	SP hard constraint
Curriculum clashes	Two courses belonging to a curriculum cannot take place in the same period. Some courses belong to two or more curricula.	A student group construction method
Room clashes	Two lectures cannot take place in the same room in the same period	SP hard constraint
Teacher clashes	Two lectures having the same teacher cannot take place in the same period	SP hard constraint
Teacher unavailabilities	A lecture cannot be scheduled to a period where the teacher is unavailable	SP hard constraint
Soft constraints		
Minimum working days	The lectures on each course must be spread over a minimum number of days	A lecture construction method
Curriculum compactness 1	Lectures belonging to a curriculum should be assigned to consecutive periods	SP soft constraint
Curriculum compactness 2	For a given curriculum there should not be a single one-period lecture assigned to a day	SP soft constraint
Room capacities	The number of students that attend a lecture (a corresponding course) must be less than or equal to the number of seats in the room that hosts the lecture	A room assignment heuristic
Room stability	All lectures on a course should be given in the same room	A room assignment heuristic + A room group construction method + A room group deconstruction heuristic

Table 2 summarizes the hard and soft constraints of the competition problem. CP is easier to solve to feasibility than the school timetabling problem. There are three main arguments for this.

The pre-assignment of rooms to lectures adds an extra hard constraint to SP. In CP, the room assignment only affects the number of soft constraint violations. A further hard constraint in SP concerns the student groups. Each student belongs to at least one base group and also to a varying number of optional groups. Students in the same base group take most of the courses together. The optional groups are built according to students' enrolments on optional courses. The base and optional groups are published by the school. Two lectures must be scheduled in different periods if their base/optional groups have common students. Finally, in SP we have to preassign the number of periods to each lecture, while in CP we can dynamically change both the number of lectures and the number of periods in each lecture.

It is quite rational to ask whether there is any purpose in converting a somewhat easier problem into a more difficult problem and then trying to solve that problem. The motivation is to give directions on how to solve problems lying between school timetabling and curriculum-based timetabling. To convert CP to SP, we need to implement five conversion methods. Figure 1 shows the conversion scheme. We have two main goals here: first, to build the student group structure from the curricula structure (*Student group construction method*) and second, to preassign a room to each lecture (*Room assignment heuristic*). The pre-assignment of rooms ensures that all lectures on a course are given in the same room. Thus the Room stability soft constraint will never be violated.

After the pre-assignment of rooms we can make the task of the school timetabling algorithm easier by grouping some rooms together (*Room group construction method*). Rooms with a roughly equal number of seats will be grouped together. Examples of such room groups in real-world timetabling problems are computer rooms and rooms for language studies. If the number of rooms in the group is *n*, we do not have a room clash in a timeslot until we have assigned more than *n* lectures to the timeslot that hosts the room group.

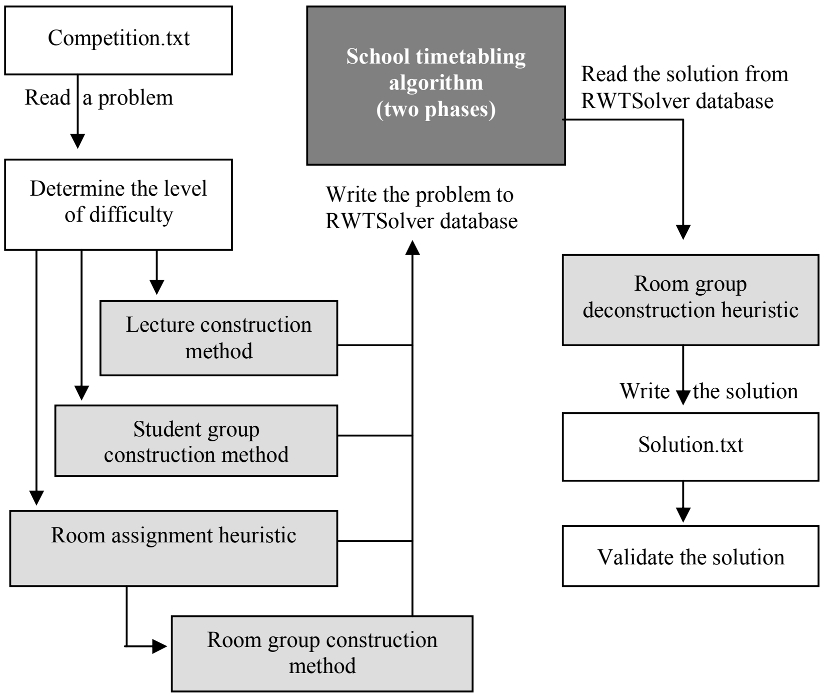
Competition.txt
Read a problem
Determine the level of difficulty
Lecture construction method
Student group construction method
Room assignment heuristic
Room group construction method
Write the problem to RWTSolver database
School timetabling algorithm (two phases)
Read the solution from RWTSolver database
Room group deconstruction heuristic
Write the solution
Solution.txt
Validate the solution

Fig. 1. The conversion scheme and the five conversion methods.

When the school timetabling algorithm has produced a solution we have to ungroup the rooms. That is, we have to assign a room from the group to each lecture that hosts the group in the solution (*Room group deconstruction heuristic*). Unfortunately, we can no longer ensure that all lectures on a course are given in the same room and the Room stability soft constraint will then be violated. As a result, the more rooms we put in the room groups, the easier it will be to construct a feasible timetable, but at the same time we introduce more soft constraint violations.

In SP, both the number of lectures on a course and the number of periods in each lecture are pre-assigned. Therefore, we need to construct a fixed number of lectures from the course list of CP (*Lecture construction method*). When constructing the lectures we have to consider the soft constraints. For example, the Minimum working days soft constraint states that the lectures on each course must be spread over a minimum number of days.

3 Conversion Methods

We need to implement five conversion methods to be able to convert a curriculum-based timetabling problem into a school timetabling problem. Figure 1 and Table 2 summarize the conversion scheme. The outlines of the conversion are as follows: First, we read a competition problem from a text file and then determine the difficulty of the problem. Next, we construct the lectures of the courses and create the student groups from the curricula. Then we pre-assign a room to each lecture. If the level of difficulty is greater than one (see Chapter 5), we construct the room groups in order to make the task of the school timetabling algorithm easier. We have now created an instance of the school timetabling problem.

We solve the problem instance using our original school timetabling algorithm (Nurmi 1998). The algorithm uses the configuration that was found to work best in (Nurmi and Kyngäs 2007). We do no further fine-tuning of the algorithm. We read the solution of the algorithm from the program (RWTSolver) database and deconstruct the room groups created earlier. Finally, we write this newly created solution to a text file, and validate it using the competition validator (McCollum 2007). A description of each of the conversion methods follows.

Lecture construction method

In the curriculum-based timetabling problem each course is given a total number of periods to be lectured. While constructing a timetable we can dynamically change both the number of lectures and the number of periods for each lecture. This gives a great deal of flexibility when minimizing soft constraint violations, especially Minimum working days, Curriculum compactness 1 and Room stability.

In the school timetabling problem both the number of lectures on a course and the number of periods for each lecture are pre-assigned. We need to construct a fixed number of lectures from the course list of CP. Basically, we have two options:

a) spread the lectures over the number of days or
b) spread the lectures to match the Minimum working days (soft constraint).

For example, if we have a course with eight periods of lectures and the Minimum working days is three, we get the following preassignments in the 5-day timetable:

a) 2, 2, 2, 1, 1, or
b) 3, 3, 2.

It should be intuitively clear that spreading lectures to match Minimum working days performs better. It has a built-in mechanism to reduce the number of soft constraint violations. Our test runs confirmed this intuition.

Student group construction method

In the curriculum-based timetabling problem, a curriculum is a group of courses where any pair of courses in the group has students in common. In the school timetabling problem, an individual student has no meaning; instead, students are pre-assigned to one or more base groups. Each student belongs to one base group and most of the lessons are scheduled based on this group. In addition, a student belongs to a number of optional groups, which are constructed based on the students' course choices. For example, one student prefers to take optional courses in Science and Sports and another in Art and Music. Each course is assigned to one and only one student group, either base or optional. Base and optional groups define a matrix, which in turn determines which groups cannot have lectures at the same time.

To build the student group structure of SP from the curricula structure of CP we create a group for each curriculum. Next, if two or more curricula share a course, we create a single group for each such course and delete such courses from other groups. Finally, we define the above-mentioned overlap matrix. For example, if we have the following curricula:

Curricula 1	Course A, Course B, Course C
Curricula 2	Course A, Course C
Curricula 3	Course C, Course D, Course E
Curricula 4	Course F, Course G, Course H,

we create the following groups and matrix:

Group 1	Course B
Group 2	(deleted)
Group 3	Course D, Course E
Group 4	Course F, Course G, Course H
Group 5	Course A
Group 6	Course C.

Overlap	Group 1	Group 3	Group 4	Group 5	Group 6
Group 1	(x)			x	X
Group 3		(x)			X
Group 4			(x)		
Group 5	x			(x)	X
Group 6	x	x		x	(x)

Room assignment heuristic

In the curriculum-based timetabling problem, students enroll on courses and room assignments should be made based on the number of student enrollments on each course. The number of students that attend a course must be less than or equal to the number of seats in the room that hosts the lecture. In the school timetabling problem, a room is pre-assigned to each lecture. Therefore, we need to implement a heuristic that assigns a room to each lecture.

First, we create a course-room matrix. Then we assign the number of periods on the course to each cell where the room can host the course. Next, we assign the total number of periods in a week to the last row of the matrix indicating the maximum sum. The problem now is to select one non-empty cell from each row so that the sum of the selected cells in each column is less than or equal to the total number of periods. Table 3 gives a sample problem instance.

Table 3. A competition problem instance and the corresponding generalized assignment problem.

Total number of periods in a week: 8

Course A	1 period	10 students
Course B	2 periods	20 students
Course C	3 periods	30 students
Course D	4 periods	40 students
Course E	1 period	50 students
Course F	2 periods	60 students
Course G	3 periods	70 students
Course H	4 periods	80 students
Course I	3 periods	90 students

Room 1	41 seats
Room 2	73 seats
Room 3	95 seats

	Room 1	Room 2	Room 3
Course A	1	1	1
Course B	2	2	2
Course C	3	3	3
Course D	4	4	4
Course E		1	1
Course F		2	2
Course G		3	3
Course H			4
Course I			3
	≤ 8	≤ 8	≤ 8

This problem is actually a classical *generalized assignment problem* (Martello and Toth 1990). The mathematical formulation of the generalized assignment problem is given in formula (1):

$$\begin{aligned} cost(x) = \quad & \sum_{i=1}^{m}\sum_{j=1}^{n} p_{ij}x_{ij} \quad \text{(min)} & (1)\\ & \sum_{j=1}^{n} w_{ij}x_{ij} \le W_i \qquad i = 1,\dots,m & \\ & \sum_{i=1}^{m} x_{ij} = 1 \qquad j = 1,\dots,n & \\ & x_{ij} = 0 \ \ or \ \ 1 \qquad i = 1,\dots,m \quad j = 1,\dots,n & \end{aligned}$$

The generalized assignment problem (GAP) can be stated as finding a minimum cost assignment of *n* jobs to *m* agents, such that each job is assigned to one and only one agent and each agent's resource capacity is honored.

GAP is NP-hard (Fischer et al. 1986) and even APX-hard to approximation (Chekuri and Khanna 2000). Moreover, the problem of finding whether a feasible solution exists is NP-Complete. Even if many heuristics for GAP exist (Kellerer et al. 2004; Cohen et al. 2006; Fleischer et al. 2006), we decided to build a new heuristic that seeks a feasible solution. We need a fast enough heuristic in the Timetabling Competition since we have only a limited time in which to solve a problem instance.

In our room assignment problem courses correspond to jobs and rooms correspond to jobs. We define a cost function as

$$cost(x) = \sum_{i=1}^{m}\sum_{j=1}^{n}(w_{ij}x_{ij} - W)^{+} + \sum_{i=1}^{m}\sum_{j=1}^{n}(W - w_{ij}x_{ij})^{+}, \qquad (2)$$

and variables

$w_{ij} = c_i$	for all room *j* that can host the course,
$w_{ij} = \infty$	for all room *j* that cannot host the course and
$W_i = W$	for all *i*,

where c_i is the number of periods of the course *i*, *W* is the total number of periods in a week and *a* is a large constant value (for example *nW*). We seek a solution whereby the first part of the cost function is zero. A course *i* is assigned to room *j*, if $x_{ij} = 1$.

The heuristic is based on the *greedy hill-climbing mutation operator* introduced in our school timetabling algorithm (Nurmi 1998). The application of the operator for GAP is as follows. The operator is based on moving a selected cell ce_1 from its old column col_1 to a new column col_2, moving another cell ce_2 from column col_2 to a new column col_3 and so on, ending up with a sequence of moves. The initial column selection is random. The new column for the cell is selected considering all possible columns and selecting the one that causes the least increase in the cost function when considering only the relocation cost. Moreover, the new cell from that column is again selected considering all the cells in that period and picking the one for which the removal causes the most decrease in the cost function when considering only the removal cost. The operator stops if the last move causes an increase in the cost function value and if the value is larger than that of the previous non-improving move.

We could further improve the operator by introducing a tabu list, which prevents reverse order moves in the same sequence of moves. Another improvement would be to use the simulated annealing refinement introduced in (Nurmi 1998). However, we were able to find a feasible solution to all GAPs arising from the competition problems with no need to use either of these improvement methods. An example of one application of the greedy hill-climbing mutation operator is given in Figure 2. The problem was introduced in Table 3. The solution states that room 1 hosts courses C and D, room 2 hosts courses A, B, F and G, and room 3 hosts courses E, H and I.

(1)	1	1
(2)	2	2
(3)	3	3
4	(4)	4
	(1)	1
	(2)	2
	3	(3)
		(4)
		(3)
6	7	10
≤ 8	≤ 8	≤ 8

Cost = $2a + 3$

1	[1]	1
(2)	2	2
(3)	3	3
4	(4)	4
	(1)	1
	(2)	2
	3	(3)
		(4)
		(3)
5	8	10
≤ 8	≤ 8	≤ 8

Cost = $2a + 3$

1	(1)	1
(2)	2	2
(3)	3	3
4	(4)	4
	1	[1]
	(2)	2
	3	**(3)**
		(4)
		(3)
5	7	11
≤ 8	≤ 8	≤ 8

Cost = $3a + 4$

1	(1)	1
(2)	2	2
(3)	3	3
4	**(4)**	4
	1	(1)
	(2)	2
	[3]	3
		(4)
		(3)
5	10	8
≤ 8	≤ 8	≤ 8

Cost = $2a + 3$

1	(1)	1
(2)	2	2
(3)	3	3
[4]	4	4
	1	(1)
	(2)	2
	(3)	3
		(4)
		(3)
9	6	8
≤ 8	≤ 8	≤ 8

Cost = $a + 2$

1	(1)	1
2	[2]	2
(3)	3	3
(4)	4	4
	1	(1)
	(2)	2
	(3)	3
		(4)
		(3)
7	8	8
≤ 8	≤ 8	≤ 8

Cost = 1

Fig. 2. An example of one application of the greedy hill-climbing mutation operator for the problem instance given in Table 3. The parentheses denote the current solution, the bolded value denotes the (old) selected cell ce_i and the square brackets denote the new selected cell in the new column col_j.

Table 4. A GAP instance with no feasible solution and a new instance for which a feasible solution can be found.

	R1	R2	R3
Course A	1	1	1
Course B	2	2	2
Course C	3	3	3
Course D	4	4	4
Course E		1	1
Course F		2	2
Course G			3
Course H			4
Course I			3
	≤ 8	≤ 8	≤ 8

	R1	R2	R3
Course A	(1)	1	1
Course B	2	(2)	2
Course C	(3)	3	3
Course D	(4)	4	4
Course E		1	(1)
Course F		(2)	2
Course G			(3)
Course H		**(4)**	4
Course I			(3)
	≤ 8	≤ 8	≤ 8

If no feasible GAP solution exists (or if we are unable to find one), we have to make the problem easier by allowing a course to be assigned to a room that has too few seats. For example, no feasible room assignment exists for the competition problem "Early dataset1" (Di Gaspero 2007). We ease the problem by first selecting a course-room pair for which the difference between number of attending students and size of room is greater than zero and minimized. Then we set the number of students attending that course to the size of the room. Next we try to solve this new GAP. This process is repeated until a feasible solution is found. Table 4 gives an example. The number of students attending course H will be set to 73.

Room group construction method

We usually need to make an instance easier by grouping some rooms together. This gives flexibility because we do not have a room clash in a timeslot until we have assigned more than n lectures to the timeslot that hosts the room group, where n is the number of rooms in the group.

The idea is to group such rooms together that can host exactly the same courses. For example, in the sample case given in Table 5, rooms 2 and 3 can both host courses A, B and C.

A pseudo code for the room group construction is given in Figure 3. The construction is quite straightforward. We first sort rooms by size and courses by attending students. Then we go through the course list and find groups of courses that can host exactly the same rooms. Finally we extract the room groups from the course list (see Table 5).

Table 5. An example of the room group construction method.

		GroupSet
Course A	10 students	1
Course B	20 students	1
Course C	30 students	2, 3
Course D	40 students	4, 5
Course E	50 students	4, 5
Course F	60 students	4, 5
Course G	70 students	6
		Room Groups
Room 1	21 seats	G_1
Room 2	31 seats	G_2
Room 3	33 seats	G_2
Room 4	61 seats	G_3
Room 5	63 seats	G_3
Room 6	71 seats	G_4

```
1. Sort roomlist by room size in ascending order
2. Sort courselist by attending students in ascending order
3. Set Course = first course in a course list
   Set Room = first room in a room list
   Set GroupSet(c) = {} for all courses c
4. If Attendants(Course) > Size(Room)
      Add Room to GroupSet(c) for all c in C
   Else
      Set C = {}
   End
5. If Attendants(Course) ≤ Size(Room)
      Add Room to GroupSet(Course)
      Set C = C U {Course}
   Else
      Goto 7
   End
6. Set Course = next course in a course list and Goto 5
7. Set Room = next room in a room list and Goto 4
8. Extract room groups from GroupSet
```

Fig. 3. Pseudo code for the room group construction method.

Room group deconstruction heuristic

When the school timetabling algorithm has produced a solution we have to assign a room from the room group to each lecture that hosts the group in the solution. We need a fast heuristic since we have only a limited time in which to solve a problem instance. Even if good heuristics might exist for this problem, we decided to build a new greedy heuristic. The heuristic starts by sorting one-hour lectures first by timeslot and then by course for each room group. Then it selects each lecture in turn and finds a room that by then hosts the course the most. If there is a tie, it selects a room that hosts the least total number of lectures. A pseudo code for the heuristic is given in Figure 4. Table 6 gives a sample problem instance and Table 7 gives a solution to the instance using the room group deconstruction heuristic. The solution includes five Room stability soft constraint violations.

Table 6. An instance of the room group deconstruction problem. The subscripts denote the order in which the lectures will be processed.

Courses Timeslots	C1	C2	C3	C4	C5	C6	C7
00	x_1			x_2			
01	x_3	x_4	x_5				
02					x_6		
03			x_7	x_8		x_9	
10						x_{10}	x_{11}
11		x_{12}		x_{13}		x_{14}	
12	x_{15}		x_{16}				
13		x_{17}	x_{18}				
20					x_{19}		
21					x_{20}	x_{21}	x_{22}
22	x_{23}					x_{24}	x_{25}
23	x_{26}		x_{27}				x_{28}

Table 7. A solution to the problem instance given in Table 6 using the room group deconstruction heuristic.

Courses Timeslots	C1	C2	C3	C4	C5	C6	C7
00	A			B			
01	A	C	B				
02					C		
03			B	A		C	
10						C	A
11		C		A		B	
12	A		B				
13		C	B				
20					C		
21					C	B	A
22	A					B	C
23	A		B				C

```
For each room group G
   Break lectures belonging to G into one-hour-lectures
   Sort one-hour-lectures first by timeslot and then by course
   For each lecture L in the sort list
      If timeslot(L) $\neq$ timeslot($L_{prev}$)
         For each R set RoomSet(R) = False
      End
      Max = 0
      For each room R in G
         If Not RoomSet(R)
            If number of assignments for R > Max
               bestR = R
               Max = number of assignments for R
               Tie = False
            Else If number of assignments for R = Max
               Tie = True
            End
         End If
      Next
      If Tie
         Select a room (bestR) which hosts the least total
         number of lectures
      End
      Set Room(L) = bestR
      Set RoomSet(bestR) = True
   Next
Next
```

Fig. 4. Pseudo code for the room group deconstruction heuristic.

We could use the heuristic each time we find a new solution candidate in the school timetabling algorithm. However, our goal is not to change the original algorithm at all. We could also apply the heuristic with some minor changes to be able to produce several solutions. Then we could pick the best among these. We ended up using the heuristic only once in order to lower the running time.

4 Complexity of the Competition

As we know very well, the timetabling problem is a very tough one. It is clear that limiting the running time of an algorithm affects its design process. The main point is to find a solution fast enough and with sufficient quality, not to find the solution of the best quality. In any competition the solution candidates must easily be comparable and the results must be reproducible. This could be realized without any running time restrictions.

The competition participants were asked to benchmark their computers with the program provided in order to know how much time they have available to run their program on their computers. The competition rules emphasized that all the finalists will be run on a standard computer, thus creating a level competition. However, it is not possible to provide a perfectly equitable benchmark program across many platforms and algorithms, and any such program is inevitably more favorable to some computers than others. We tested the benchmark program on three different computers. The results are given in Table 8. Our algorithm should run an equal number of generations in each computer in the time given by the benchmark program. However, the results are quite far from each other. At one time we started to suspect that there might be a problem in our algorithm (program). Therefore, we ran the algorithm on eight different computers for 14,000 generations (9,889,446 cost function evaluations) on each computer. Every computer produced exactly the same solution, so the program works properly.

Table 8. The results between the benchmark program and the number of generations run by our algorithm in the time limit given by the benchmark program.

Computer	Benchmark program	Number of generations run by the algorithm in the time limit given by the benchmark program
1	377 s	12,161
2	429 s	6,689
3	468 s	9,985

The organizers of the Timetabling Competition have done excellent work in providing researchers with a computing challenge that is very much a combination of theory and practice. However, it is an unpleasant task to design a benchmark program that tries to take every aspect of every algorithm into consideration. Our algorithm, for example, is very much dependent on memory usage and bus speeds, not so much on processor power.

We cannot guarantee that our results will be reproducible on different computers. The main reason can be seen in Figure 1. After the school timetabling algorithm has produced a solution, we still need to deconstruct the room groups to get the final solution. Because the school timetabling algorithm produces a different solution on different time stamps, we will get a different final solution depending on the total running time. Another reason can also be seen in Figure 1. At first we determine the level of difficulty of the problem to find out how many time-consuming operations we can use for the problem in hand. But since we do not know the total running time of our algorithm on a particular computer, we cannot predict the running time of the operations either.

In order to answer criticisms by some of the participants and to push forward the spirit of the competition, the organizers proposed a larger set of possible formulations (De Cesco et al. 2008). We strongly believe that they have succeeded in their aim of capturing many real-world formulations, as well as encouraging researchers to reduce their problems to one of them, gaining the opportunity to compare and assess their results. That is exactly what we are doing.

Table 9 presents the formulations of the competition problem and the problem we are actually solving. The fact that the competition formulation is a sub-formulation of our formulation can easily be seen. The Windows component counts as many violations as there are idle periods (i.e., periods without teaching). The IsolatedLectures component only considers the number of such incidents, not their length. The minimum number of lectures in our formulation is two (StudentMinMaxLoad component). The Room assignment heuristic guarantees that the number of students attending a course is less than or equal to the number of seats in the room that hosts the lecture (RoomCapacity). Furthermore it can be used to assign a suitable room to each course (RoomSuitability). The Lecture construction method creates double lectures (DoubleLectures).

Table 9. Problem formulation descriptions using the terminology presented in (De Cesco et al. 2008). A dash sign means that the component is not included and the numbers mean the weight associated to the component.

	Competition formulation	*Our formulation*
Lectures	Hard	Hard
Conflicts	Hard	Hard
RoomOccupancy	Hard	Hard
Availability	Hard	Hard
RoomCapacity	1	Hard
MinWorkingDays	5	5
IsolatedLectures	2	–
Windows	–	2
RoomStability	1	1
StudentMinMaxLoad	–	1
TravelDistance	–	–
RoomSuitability	–	Hard
DoubleLectures	–	Hard

5 Competition Results

This chapter presents our results to the competition track 3, which tackles the curriculum-based timetabling problem (Di Gaspero 2007). Chapters 2 and 3 described how we convert a curriculum-based timetabling problem into a school timetabling problem. We solve the school timetabling problem using the algorithm presented in (Nurmi 1998). The algorithm is basically a genetic algorithm with one mutation operator and no recombination operators. The two most important features of the algorithm are the greedy hill-climbing mutation (GHCM) operator and the adaptive genetic penalty method (ADAGEN), which is a multiobjective optimization method. We do no fine-tuning of the algorithm, but use the same configuration found to work best in (Nurmi and Kyngäs 2007).

We present our results applying the rules of the competition. The main points here are that

- our algorithm runs on a single processor computer,
- we have 468 seconds to run the algorithm on our computer according to the benchmark program provided,
- the algorithm uses no other knowledge about a problem instance but those provided in the problem file, and the same version of the algorithm is used for all instances, and
- our results are repeatable within the given computer time (but only using the same computer that the results were originally obtained with).

As pointed out in Chapter 4, limiting the running time of an algorithm affects its design process. In our case this means that first we have to determine the difficulty of a problem instance to find out how many time-consuming operations we can use for the instance in hand. For the competition problem instance *c* we define the degree of difficulty as:

$$\texttt{diff}(c) = \frac{To}{Da \times Pe \times Po} \times \frac{Un}{Te} \times \frac{Gr - Cu}{Co}, \qquad (3)$$

where

```
To = Total number of periods in all the courses,
Da = Number of days,
Pe = Number of periods in a day,
Ro = Number of rooms,
Un = Number of unavailabilities,
Te = Number of teachers,
Gr = Number of student groups created,
Cu = Number of curricula and
Co = Number of courses.
```

The degree of difficulty does not necessarily define the overall difficulty of the problem instance, but it is designed to determine the difficulty for our algorithm under competition rules (mainly time limit). We classify the competition problem instances into five levels based on their *diff(c)* values. Table 10 summarizes the five levels.

Table 10. Five levels of difficulty based on the degree of difficulty of a problem instance.

Degree of difficulty, *diff(c)*	Level of difficulty, *level(c)*
[0.0, 1.4)	1 = very easy
[1.4, 2.2)	2 = easy
[2.2, 3.0)	3 = moderate
[3.0, 3.8)	4 = difficult
[3.8, ∞)	5 = hard

The competition rules stated that in order to compare two solutions, the number of hard constraint violations will be calculated and the solution with the lowest value for this will be the winner. If the two solutions are tied, the number of soft constraint violations will be calculated. The winner will be the solution that has the lowest value here. Our school timetabling algorithm uses a multiobjective optimization method (ADAGEN), which minimizes both hard and soft constraints at the same time. So there is a contradiction of some sort between the rules and our algorithm. We must first concentrate on minimizing the hard constraints and we have to do that quickly and then we should put all the effort into minimizing the soft constraints. That is, the algorithm solves a competition problem in two phases.

Based on the *level(c)* value of the problem instance, we first determine which soft constraints we cannot afford to minimize while minimizing all the hard constraints. This problem is then solved using ADAGEN multioptimization. When the algorithm has found a feasible solution, it is started again with all the soft constraints active this time. The second phase of the algorithm ends when the competition time limit is reached. Table 11 summarizes the use of room groups and soft constraints based on the *level(c)* value. Even if the lecture construction method creates exactly the number of lectures given by the Minimum working days (soft constraint), the algorithm can still place two lectures on the same day. Therefore, we have to minimize this soft constraint as well. If a competition problem is very easy to solve, we do not have to use the room group construction method.

A soft constraint in the objective function is weighted based on its penalty value given in the competition rules. When a feasible solution is found, all the weights of the soft constraints are set to their corresponding penalty values and the algorithm is started again.

Table 11. The use of room groups and soft constraints based on the difficulty of a problem instance. The numbers correspond to the weights of the soft constraints in the objective function of the ADAGEN method.

Level(c)	Room groups in use	Minimum working days	Curriculum compactness 1	Curriculum compactness 2
Very easy	No	5	1	2
Easy	Yes	5	1	2
Moderate	Yes	5	1	0
Difficult	Yes	1	0	0
Hard	Yes	0	0	0
When feasible				

solution found (phase II)	(no change)	5	1	2

We are finally ready to present our results to the competition problems. The early and late problems were available at the time of writing this article. We solved these 14 problems and found a feasible solution to 12 of them within the given time limit, which was 468 seconds for the computer used. The problems E5 and L5 cannot be solved to feasibility using our formulation (see Chapter 4). The time spent on the conversion scheme is about two percent (10 seconds) of the total running time. Table 12 summarizes the results.

Table 12. Results for the early (E) and late (L) competition problems. The cost of the best solution is the weighted sum of the soft constraint violations given by the validator provided by the organizers. The number of violations is given for each soft constraint. The last column gives the best solutions found in the competition (McCollum 2007).

	diff(c)	*level(c)*	**Best found**	Minimum working days	Curriculum compactness 1 + 2	Room capacities	Room stability	**Best known**
E1	0.720	1	12	0	0	12	0	5
E2	3.179	4	173	1	66	0	36	50
E3	3.002	4	158	2	63	0	22	71
E4	2.185	2	93	1	32	0	24	35
E5	6.841	5	H = 7					309
E6	3.669	4	206	5	75	0	31	48
E7	3.437	4	179	0	64	0	51	20
L1	2.527	3	122	1	42	0	33	40
L2	2.721	3	179	5	61	0	32	105
L3	3.834	5	164	2	66	0	22	16
L4	0.564	1	0	0	0	0	0	0
L5	9.136	5	H = 5					333
L6	2.403	3	163	3	60	0	28	66
L7	3.264	4	142	1	53	0	31	57

6 Conclusions and Future Work

In this article we considered a conversion scheme for turning a curriculum-based timetabling problem to a school timetabling problem. Our algorithm found a feasible solution for 12 of the 14 problems used in the 2nd International Timetabling Competition within the given time limit. We believe that our approach gives directions on how to solve problems lying between school timetabling and curriculum-based timetabling problems.

Our direction for future research would be to strengthen the presented conversion methods and heuristics, and to solve the different combinations of problems presented in (De Cesco et al. 2008). Another direction for future work would be to tackle Round-Robin Scheduling (Rasmussen and Trick 2006).

References

Burke, E.K., De Causmaecker, P. (2003). The Practice and Theory of Automated Timetabling IV: Revised Selected Papers from the 4th International conference, Gent 2002. Lecture Notes in Computer Science, vol. 2740, Springer.

Burke, E.K., Petrovic S. (2002). Recent Research Directions in Automated Timetabling. European Journal of Operational Research, 140/2, 266-280.

Burke, E.K., Rudová H. (2006). Proceedings of the 6th International Conference on the Practice and Theory of Automated Timetabling. Brno, Czech.

Burke, E.K., Trick M. (2005). The Practice and Theory of Automated Timetabling V: Revised Selected Papers from the 5th International conference, Pittsburgh 2004. Lecture Notes in Computer Science, vol. 3616, Springer.

Chekuri, C., Khanna, S. (2000). A PTAS for the multiple Knapsack problem. In: Proc. of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 213-222.

Cohen, R., Katzir, L., Raz, D. (2006). An Efficient Approximation for the Generalized Assignment Problem. Information Processing Letters, vol 100, Issue 4, 162-166.

De Cesco, F., Di Gaspero, L, Schaerf, A. (2008): Benchmarking Curriculum-Based Course Timetabling: Formulations, Data Formats, Instances, Validation, and Results. Submitted to the 7th International Conference on the Practice and Theory of Automated Timetabling. Montreal, Canada.

De Werra, D. (1985). An Introduction to Timetabling. European Journal of Operations Research, vol. 19, 151-162.

Di Gaspero, L., McCollum B., Schaerf, A. (2007): The 2nd International Timetabling Competition, Curriculum-based Course Timetabling (Track 3). Technical Report, http://www.cs.qub.ac.uk/itc2007/curriculmcourse/report/curriculumtechreport.pdf. Accessed 1 June 2008.

Fisher, M., Jaikumar, R., Van Wassenhove, L. (1986). A Multiplier Adjustment Method for the Generalized Assignment Problem. Management Science, vol. 32, 1095-1103.

Fleischer, L., Goemans, M. X., Mirrokni, V. S., Sviridenko, M. (2006). Tight Approximation Algorithms for Maximum General Assignment Problems. In: Proc. of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 611-620.

Kellerer, H., Pferschy, U., Pisinger, D. (2004). Knapsack Problems. Springer.

Lewis, R., Paechter, B., McCollum, B. (2007). Post Enrolment based Course Timetabling: A Description of the Problem Model used for Track Two of the 2nd Int. Timetabling Competition. Working Paper A2007-3, Cardiff University, Wales.

Martello, S., Toth P. (1990). Knapsack Problems: Algorithms and Computer Implementations. Wiley, New York.

McCollum, B. (2006). University Timetabling: Bridging the Gap between Research and Practice. In: Proc. of the 6th Int. Conf. on the Practice and Theory of Automated Timetabling, 15-35, Brno, Czech.

McCollum, B. (2007): The 2nd International Timetabling Competition, http://www.cs.qub.ac.uk/itc2007/index.htm. Accessed 1 June 2008.

Nurmi, K. (1998). Genetic Algorithms for Timetabling and Traveling Salesman Problems. University of Turku, Finland.

Nurmi, K., Kyngäs, J. (2007). A Framework for School Timetabling Problem. In: Proc. of the 3rd Multidisciplinary Int. Scheduling Conf.: Theory and Applications, 386-393, Paris, France.

Rasmussen, R.V., Trick, M.A. (2006). Round Robin Scheduling – a survey. Working Paper no. 2006/2, Department of Operations Research, University of Aarhus, Denmark.

Schaerf, A. (1999). A Survey of Automated Timetabling. Artificial Intelligence Review, 13/2, 87-127.