Comprehensive approach to University Timetabling Problem

Wojciech Legierski, Łukasz Domagała

Silesian University of Technology, Institute of Automatic Control, 16 Akademicka str., 44-100 Gliwice, Poland
Wojciech.Legierski@polsl.pl, Lukasz.Domagala@student.polsl.pl

Abstract

The paper proposes a comprehensive approach to University Timetabling Problem, presents a constraint-based approach to automating solving and describes a system that allows concurrent access by multiple users. The timetabling needs to take into account a variety of complex constraints and uses special-purpose search strategies. Local search incorporated into the constraint programming is used to further optimize the timetable after the satisfactory solution has been found. Paper based on experience gained during implementation of the system at the Silesian University of Technology, assuming coordinating the work of above 100 people constructing the timetable and above 1000 teachers who can have influence on timetabling process. One of the original issues used in real system, presented in the paper, is multi-user access to the timetabling database giving possibility of offline work, solver extended by week definitions and dynamic resource assignment.

Introduction

Timetabling is regarded as a hard scheduling problem, where the most complicated issue is the changing of requirements along with the institution for which the timetable is produced. Automated timetabling can be traced back to the 1960s [Wer86]. Some trials of comprehensively approaching the timetabling problem are presented in the timetabling research center lead by prof. Burke [PB04] and in several PhD thesis [M05],[Rud01],[Mar02]. There are works connected with general data formulation , metaheuristic approaches, and user interfaces for timetabling. The paper presents a proposition of a comprehensive approach to the real-world problem at Silesian University of Technology. This paper presents the methods used for automated timetabling, data description and user interaction underlining connection of different idea to built whole timetabling system.

Problems description

A number of timetabling problems have been discussed in the literature [Sch95]. Based on the detailed classification proposed by Reise and Oliver [RL01], the presented problem consists of a mixture of following categories:

Class-Teacher Timetabling (CTT) – the problem amounts to allocating a timeslot to each course provided for a class of students that has a common programme,
Room Assignment (RA) - each course has to be placed in a suitable room (or rooms), with a sufficient number of seats and equipment needed by this course.
Course Timetabling (CT) - the problem assumes that students can choose courses and need not belong to some classes.
Staff Allocation (SA) - the problem consists of assigning teachers to different courses, taking into account their preferences. The problem assumes that one course can be conducted by several teachers.

Till now Examination Timetabling (ET) was not required, but is planned to be added in future.

Comprehensive approach to University Timetabling Problem (UTP), besides taking into account different timetabling problems, also assumes following tasks:

- formulating timetable data requires a lot of flexibility,
- automated methods should be available for sub-problems and should be able to take into account many soft and hard constraints,
- timetabling can be conducted by many users, simultaneously, which requires assistance in manual timetabling and quick availability to different resources' plans.

Constraints

Timetable of UTP has to fulfill the following constraints, which can be expressed as hard or soft:

- resources assigned to a course (classes, teachers, rooms, students) have time of unavailability and undesirability,
- courses with the same resource cannot overlap,
- some courses must be run simultaneously or in defined order,
- some resources can be constrained not to have courses in all days and more than some number during a day,
- no gaps constraint between courses for the same resource or gaps between some specific courses can be strictly defined,

- the number of courses per day should be roughly equal for defined resource - p,
- courses should start from early morning hours.

Data representation

Although UTP data is gathered in relational database for multi user access, data for the solver is saved as a XML file, which also expresses sub-problems for the solver. The main advantage of using XML file is the ability of defining relations between courses, resources and constraints in a very flexible way. The flexibility of UTP features:

- defining arbitrarily resources (classes, teachers, rooms, students)
- allowing assignment of some different resources to one course,
- assigning resources can be treated as disjunction of some resources, where also a number of chosen resources can be defined,
- constraints can be imposed on every resource and every course,

Additionally in UTP we are supposed to produce a plan which is coherent during a certain time-span (it would be for example one semester), with courses taking place cyclically with some period (most often one-week period). But frequently we face a situation where some courses do not fit into the period mentioned above, for example some of them should appear only in odd weeks or only in even weeks and thus have a two week period. Seeking solution to this problem we introduced the idea of "week definition". Different week definitions can be defined in the timetable, together with the information, which of them have common weeks and the courses assigned to them.

Multi user access

The UTP requires taking into account that there are many timetable designers, who are engaged in timetabling process. The teachers and students are asked to submit information about their choices as well as time preferences. The appropriate management of the user interaction is solved by introducing 3 levels of the rights assigned to each user and connected with set of resources like groups, teachers, students and rooms:

- user can be administrator of the resource,
- user can be planner of the resource,
- user can only use resource for courses.

Additionally each resource and courses have user which is call "owner". Owners and administrators can block resources to restrict changing them.

Manual timetabling assistance

Timetable designers often do not want to introduce all the constraints and trust the computer in putting courses in the best places. Manual timetabling assistance with constraint explanation seems to be a very important step in making timetable system useful. The assistance requires very quick access to a lot of data and relations between them to provide a satisfactory interface. Therefore after dragging the course, colors of unavailable timeslots change to color defining what sort of constraints will be violated. For example overlapping of rooms courses has gray color and undesirable hours of a teacher leaded the course has yellow color.

Structure of the system

The proposed solution for comprehensive approach to UTP requires a usage of different languages and technologies for different features. Therefore the proposed system consists of 4 parts as presented in Figure.1. The system was firstly presented by the author in [LW03]. Presented system was extended mainly by multi-user access.

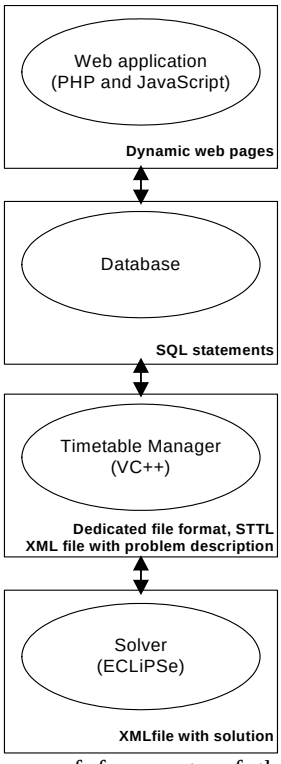

Figure 1, Diagram of four parts of the system, their dependencies and their output data format.

Web application

HTML seems to be an obvious solution for presenting results of the timetabling process in the Internet, but it provides only static pages which are not sufficient for

SAT. JavaScript improves the user interface and provides the capability to create dynamic pages.
As client-side extensions it allows an application to place elements on a HTML form and respond to user events such as mouse clicks, form input, and page navigation. Server-side web-scripting languages provide developers with the capability to quickly and efficiently build Web applications with database communication. They became in the last decade a standard for dynamic pages. PHP being one of the most popular scripting languages was chosen for developing the system. Its main advantages are facilities for database communication. People are used to Web services which provide structured information, search engines, email application etc. The proposed system had to be similar to those services in its functionality and generality. Most timetable applications give a possibility to save schedules for particular classes, teachers and rooms as HTML code, but they do not allow interaction with its users. Creating a timetable is a process which involves often a lot of people working on it. There are not only timetable designers, but also teachers, who should be able to send their requirements and preferences to the database. This is to a high extent facilitated by a web application, which allows interaction between teachers, students and timetable designers. An example screen of the web timetabling application is presented in Figure 2.

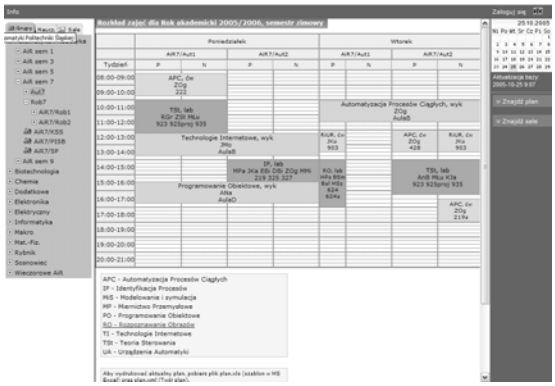

Figure 2. The example screen of the Web application

Timetable Manager

It is hard to develop a fully functional user interface using only Internet technologies. Therefore VC++ was used to build the Timetable Manager, a program for timetable designers. The idea of the program was to simplify manual timetabling and to provide as much information as possible during this process. Operating on the data locally significantly increases performance during data manipulation, data manipulation is based on SQL queries on database. Well-known features – drag and drop can be implemented, layout is based on tree navigation. One of the most important feature of the timetamble manager is assistance during manual timetabling. Small timetables, which automatically show schedules for resources of a selected course can be freely placed by the user. During manual scheduling available timeslots are shown and constraint violations are explained by proper colors. Data can be saved in two ways:

- data is saved locally in dedicated file format,
- data is synchronized with remote database.

The system takes into account privileges of users and does not allow unauthorized change of data. An example screen of the Timetable Manager is presented in Figure 3.

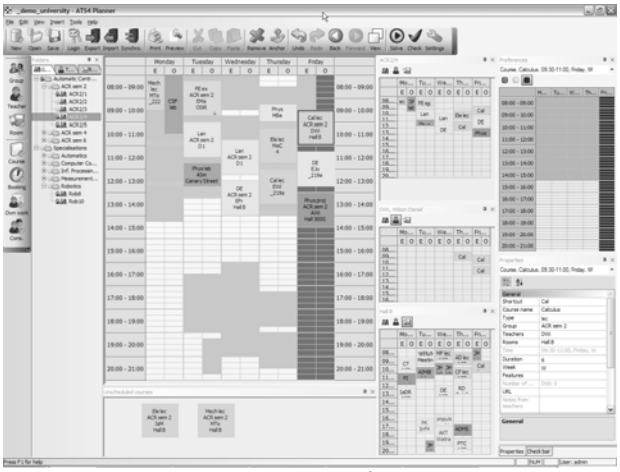

Figure 3. The example screen of manual assistance in Timetable Manager

Multi user support

Allowing user to work locally forces to develop of a data synchronization mechanism between locally changed data and remote database. The proposed mechanism is based on idea of versioning systems like CVS or SVN. But taking into account timetable data is much harder than text files, because of complicated relations between data.
The main advantages of this mechanism are following:

- simultaneously changes are allowed and in case of conflict possibilities discard changes or introduce them are given to the user,
- user have very quick access to all timetable without blocking them for other users,
- changes can be applied for a lot of data (e.g. through locally solver) ,
- if data are not changed by one user or user has no rights to change data, there updated without inform the user,
- default values for changes are chosen in such a way, that newest changes are taken or changes with higher level of rights.

Two actions are proposed to take care of integrity of the data:

Import/update (it is required if data are changed remotely and user want make export)

1. Assume unique index for each course and resource and date of the last change. Indexes of deleted resources are remembered in separate table. *maxIndex* – the greater value of all indexes.
2. Remember locally current state (*local_UT*) and a whole state of the last imported timetable (*last_ remote_UT)*.
3. Select changes from database, which are newer than *last_ remote _UT*.
4. Introduce changes to the *last_remote_UT* and build *remote_UT*.
5. Indexes of local resource, which are greater than *last_remote_UT.maxIndex* are increased by *remote _UT.maxIndex - last_ remote _UT.maxIndex.*
6. Compare all data of the 2 timetables (*remote_UT* and *local_UT*), and check what kid of data was changed locally or remotely. Give user possibilities to accept or reject changes for data which change both locally and remote.
7. By default assume acceptance of the changes.
8. Replaced *last_remote_UT* with *remote_UT*.

Export/commit

1. Make Import to check changes and build *remote_UT*. Export is available if the *last_remote_UT* does not differ from *remote_UT*. Otherwise import is forced.
2. Compare *local_UT* with *remote_UT* based on the last change date to show user what changes will be exported
3. Assume default introduced changes to send them to database.
4. If some resources or courses are removed, store indexes with data in a special table.

Multi user support was the most desire feature of the whole timetable system. It can be solved by online working on database with multi-user access, transactions and locking tables. But this solution was rejected, because of low performance in case of simultaneous work of many users.

Automated timetabling based on Constraint Programming paradigm

The presented solver is written in ECLiPSe [ECL] using the Constraint Programming paradigm and replaced solver written in Mozart/Oz language. Main idea of the methodologies are similar to those widely presented in author's PhD thesis [Leg06], [Leg03]. The main idea of the solver were:

- effective search methods are customized for taking soft constraints into account during the search, based on the idea of value assessment [AM00]
- custom-tailored distribution strategies are developed, idea of constraining while distributing, which allowed to effectively handle constraints and search for 'good' timetables straight away,
- custom-tailored search methods were developed to enhance search effectiveness of timetabling solutions,
- integration of Local Search techniques into the Constraint Programming paradigm search enhanced optimization of timetabling solutions .

Additionally flexibility of the timetable definition was widened by the week definitions and dynamic resource assignment.

Week definitions

The idea of incorporating week definitions into the problem definition comes from the fact that scheduling an "odd" course will cause unused time in the "even" weeks and vice versa. This might cause long gaps between courses and could also render the problem unsolvable. To deal with this disadvantage we could prolong the scheduling period from one to two weeks to take account of courses with a longer cycle. This unfortunately has a drawback of doubling the domains of the courses' start variables and a necessity to add special constraints (to enforce the weekly scheduled courses happening in the same time during both weeks). The aforementioned solution would however not apply in some situations, for example in the case when some courses are required to happen only a few times in the semester or only in the second half of the semester. It would also increase the size of variable domains causing a great computational overhead. We can eliminate these drawbacks thanks to the introduction of week definitions. Week definitions are logical structures that group a certain number of time periods from the whole time-span. Referring to the previous examples a week definition of odd weeks would consist of weeks numbered <1 , 3 , 5 …15>, week definition of all weeks<1,2,3,…16> and so on , more examples below:

```
week_def{id:"A",
  weeks:[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16]
}  ( all weeks )
week_def{id:"O",   weeks:[1,3,5,7,9,11,13,15]}
  ( odd weeks )
week_def{id:"E", weeks:[2,4,6,8,10,12,14,16]}   (
even weeks )
week_def{id:"SHO", weeks:[9 ,11,13,15]}   (second
half of the semester odd weeks)
week_def{id:"F4W", weeks:[1,2,3,4]}   (first four
weeks)
```

For certain pairs of week definitions we state whether they are in conflict, which corresponds to the fact that their sets of weeks have common elements. Basing on the example above we would say that conflicts are:

```
week_def_conflict{id1:"A", id2:"O"}
week_def_conflict{id1:"A", id2:"E"}
week_def_conflict{id1:"A", id2:"SHO"}
week_def_conflict{id1:"A", id2:"F4W"}
week_def_conflict{id1:"O", id2:"SHO"}
week_def_conflict{id1:"O", id2:"F4W"}
week_def_conflict{id1:"E", id2:"F4W"}
```

Having defined week definitions we take from the input data assignment at least one of them to each course:

```
course {id:"act1",start_time:SA1 duration:5, … ,
week_defs:["O"] , … }
("act1" taking place in odd weeks)
course    {id:"act2",start_time:SA2    duration:7
week_defs:["F4W", "SHO"] , … },
("act2" taking place in first four weeks of the
semester and in odd weeks of the second half of
the semester )
```

These information is used at the constraint setup phase. Pairs of courses which do not contain any conflicting definitions are excluded from the constraint setup, because they occur in different weeks and therefore there is no risk that they would require the same resources during the same time.
Pairs of courses that contain at least one pair of conflicting week definitions, are potentially competitors for the same resources during the same time and need to be taken under consideration during constraint setup.
The idea of week definitions is a universalized and convenient approach of handling courses which are exceptional and do not occur regularly within each time period.

Dynamic resource assignment

We have taken the approach that resources do not have to be instantiated at the phase of problem definition, which on one hand enforces a more complex programmatic approach but on the other better reflects the nature of real timetabling problems and also allows greater flexibility at search phase (possibility of balancing resource usage, moving courses between resources might lead to further optimization of the cost function).
Normally we would assume that a course requires a fixed set of resources to take place. That would be for example, a group of teachers, a group of classrooms and a group of students, all known and stated at the time of problem definition. We extend this model by enabling the user to state how many elements from a group of resources are required , without an explicit specification which ones should be used. This flexibility is achieved thanks to the definition and management of resource groups implemented in our XML interface and processed by the solver. The data structure is such that for every course we define resources. Resources are defined by a (theoretically unlimited) number of resource groups. Each group contains indexes, that correspond to certain resources and, as a property, a number of required resources.
The number of required resources can range from one to the cardinality of the group. When the number of required resources is maximal, all the resources within the group need to be used, but for any number below the maximum we are left with a choice of resources.

```
<Course>
…
<Resources>
    <Group required=2 >
      <Resource>teacher_32</Resource>
      <Resource>teacher_78</Resource>
      <Resource>teacher_93</Resource>
    </Group>
    <Group required=1 >
      <Resource>classroom_122</Resource>
      <Resource>classroom_123</Resource>
      <Resource>classroom_144</Resource>
      <Resource>classroom_215</Resource>
    </Group>
    <Group required=1 >
      <Resource>students_group_23</Resource>
    </Group>
    <… optionally more groups>
</Resources>
</ Course >
```

This structure is translated into resource variables list in each course.

```
Course   … , resource_variables_list:[Teacher1,
Teacher2, Classroom1, StudentGroup1] , …}
```

And domains of those variables present in the list

```
domain(Teacher1)=domain(Teacher2) = [teacher_32,
teacher_78, teacher_93 ]
domain(Classroom1)=[classroom_122              ,
classroom_123 , classroom_144 , classroom_215]
```

For every group of resources we create as many resource variables as number of required resources, and give each of them a domain of all resources in a group , then constrain them to be all-different (since we cannot use any resource twice in one course). For those groups where all resources are required, variables should get instantiated right away which corresponds to the model with fixed resources:

```
StudentGroup1 = students_group_23
```

What we need to ensure now is that any two courses do not use the same resource at the same time. This is achieved for instantiated resources by imposing a constraint that prevents courses from overlapping in time, for every pair of courses that use the same instantiated resource and are in conflict according to week definitions. It is sometimes possible to set up global constraints involving more than two courses that require the same resource but only if each pair in the group is in conflict according to their week definitions, which is not always the case.

What still needs to be handled are the uninstantiated resource variables with domains. To do this we impose a suspended test on every pair of courses that have at least one common resource in their resource variables domains and are in conflict according to their week definitions. The tests wait for instantiation of both resources that could potentially be the same, and checks if they are. If the test succeeds, the constraint that prevents the pair of courses from overlapping is imposed on the courses. The invocation of tests and consequently imposing of constraints happens at the search phase when resources get instantiated by the search algorithm.

To enhance the constraint propagation it is useful to impose a second set of tests on the courses to ensure that the same resources are not chosen for courses that overlap in time. To achieve this , for each pair of courses that are in conflict according to their week definitions we impose a test checking whether the courses overlap (different conditions guarding for domain updates are acceptable here, domain bound changes as well as variable instantiation). If the test succeeds the all-different constraint is imposed on resource lists of the two courses stating that none of the variables in one list takes the same value as any variable in the other (since they can not use the same resources whilst overlapping in time and belonging to conflicting week definitions).

This second set of tests (considering courses' start times) is redundant. We notice that its declarative meaning is the same as for the first set of tests (considering resource variables) , but in the case when we proceed through the search tree both by instantiating start times for courses and resource variable, we get a better constraint propagation and avoid exploring some parts of search tree which do not contain a solution.

There is a need to use these suspended tests that set up constraints during search phase, because at the constraint setup phase we do not have the knowledge which activities will overlap in time or which will use the same resources therefore we need to wait for further instantiation of variables. This slight complication is the consequence of using dynamic resource assignment.

Results

The final results cannot be presented, because of the implementation stage of the whole system. Some results are taken from previous solver written in Mozart/Oz language for two small real problem – one from high-school and departure at the Silesian University of Technology. Results presented in Figure 4 shows that using a too complicated propagator can twice increase time and memory consumption.

Timetable for	Parameters	Propagators	
		serialize	disjoint
high-school	time [s]	73	36
(253 courses)	memory [MB]	723	337
university	time [s]	32	12.5
(223 courses)	memory [MB]	269	156

Figure 4. Comparison of two types of no overlap constraints.

Schedule.serialize is a strong propagator to implement capacity constraints. It employs edge-finding, an algorithm which takes into consideration all tasks using any resource. This propagator is very effective for job-shop problems. However, for the analyzed cases this propagator is not suitable, because most tasks have frequently small durations and the computational effort is too heavy as compared with the rather disappointing results. FD.disjoint which although may cut holes into domains, must be applied for each two courses that cannot overlap. Those constraints enable also the handling of some special situations connecting with week definitions described in previous section.

Popular first-fail (FF) strategy was compared with custom-tailored distributed strategy (CTDS) based on constraining while distributing and choosing those values for variables, which have smallest assessment (assessment for value was increased when soft constraints were violated). Optimization was checked for popular branch-and-bound and idea of incorporation local search into constraint programming. This idea based on following steps after finding feasible solution:

1. Finds a course which introduced highest cost (e.g. makes gaps between courses)
2. Finds a second course to swap with the first one.
3. Creates a new search for the original problem from memorized initial space.
4. Instantiates all courses (besides these two previously chosen) to the values from solution. It can be made in one step because they surely do not violate constraints.
5. Schedules first course in the place of the second one.
6. Finds the best start time for the second course.

7. Computes the cost function. If it has improved, the solution is memorized, else another swap is performed.

Results of comparisons are presented in Figure 5.

	University			High-school		
	time [s]	mem. [MB]	cost	time [s]	mem. [MB]	cost
FF	19	258	7185,5	7,5	119	29909
FF+BAB		no improvement		28	450	29908
CTDS	29,5	340	429,0	10,5	177	18657
CTDS+BAB		no improvement			no improvement	
CTDS+LS	36	340	180,5	17	177	17158

Figure 5. Comparison of two types of distribution strategy and optimisation methods.

Conclusion and future work

Presented system describing comprehensive approach to real-world University Timetabling problem is still during implementation at the Silesian University of Technology. Most of the parts system has been already implemented, but it is still not used in full range. Multi-user paradigm has been already implemented and tested. It is one of the most important feature appreciated by the user, which use nowadays only manual assistance of the presented system. Authors plan test different methodologies based on Constraint Programming and Local Search after gathering data from whole university. The different search methods will be tested similar to Iterative Forward Search presented in [M05].

References

[AM00] S. Abdennadher and M. Marte. *University course timetablingusing constraint handling rules*. Journal of Applied Artificial Intelligence, 14(4):311–326, 2000.

[ECL] The ECLiPSe Constraint Programming System, http://eclipse.crosscoreop.com/

[Leg03] W. Legierski. *Search strategy for constraint-based class-teacher timetabling*. In Practice and Theory of Automated Timetabling IV, volume 2740 of Lecture Notes in Computer Science, pages 247–261. Springer-Verlag, 2003.

[LW03] W. Legierski and R. Widawski. *System of automated timetabling*. In Proceedings of the 25th International Conference Information Technology Interfaces ITI 2003, Lecture Notes in Computer Science, pages 495–500, 2003.

[Leg06] W. Legierski. Automated timetabling via Constraint Programming, PhD Thesis, Silesian University of Technology, Gliwice, 2006.

[Mar02] M. Marte. *Models and Algorithms for School Timetabling A Constraint-Programming Approach*. PhD thesis, Ludwig-Maximilians-Universitat Munchen, 2002.

[M05] T. Muller. *Constraint-based Timetabling*. PhD thesis, Charles University in Prague, Faculty of Mathematics and Physics, 2005.

[PB04] S. Petrovic and E.K. Burke, Edmund K, *Handbook of Scheduling: Algorithms, Models, and Performance Analysis, Chapter 45: University Timetabling*,CRC Press,Edt: J. Leung, 2004

[RL01] Oliveira E. Reise L.P. *A language for specifying complete timetabling problem*. In Practice and Theory of Automated Timetabling III, volume 2079 of Lecture Notes in Computer Science, pages 322–341. Springer-Verlag, 2001.

[Rud01] H. Rudova. *Constraint Satisfaction with Preferences*. PhD thesis, Masaryk University Brno, 2001.

[Sch95] A. Schaerf. A survey of automated timetabling. In Wiskunde en Informatica, TR CS-R9567. CWICent, 1995.

[Wer86] J. Werner. *Timetabling in Germany: A survey*. Interfaces, 16(4):66 74, 1986.