AN INTEGRATED SYSTEM FOR SCHOOL TIMETABLING*

Luisa Carpente
Department of Mathematics, University of A Coruña, Spain.
luisacar@udc.es

Ana Cerdeira-Pena, Guillermo de Bernardo, Diego Seco
Database Laboratory, University of A Coruña, Spain.
acerdeira@udc.es, gdebernardo@udc.es, dseco@udc.es

Keywords: timetable generation, planning and scheduling

Abstract: In this paper, we present an application that covers the whole complex school timetabling process, from the data introduction to the final adjustment of the automatically generated solution. On the one hand, our application interacts with the Academic Administration Official Systems (AAOS) and simplifies the hard phase of introducing the data. On the other hand, complete solutions are efficiently provided by an algorithmic engine based on different heuristic techniques, and easily updated by means of a thoroughly designed user interface.

1 INTRODUCTION

Several years ago, timetables were manually created by the educational center staff. Nowadays, this process has been simplified by semi-automatic solutions based on timetable generation applications (e.g. Kronowin). These programs build a set of timetables, but still do not solve the whole problem. The tedious tasks of data introduction and revision of usually incomplete solutions are the bottleneck in these cases. Furthermore, these applications lack complete integration capabilities that would allow the users of the system to import/export the data from/to the AAOS.

The first contribution of this paper lies in an empirical comparison of a group of algorithms, based on different heuristic techniques, that solve the optimization problem in a complete and efficient way. However, educational centers need easy-to-use systems that make lighter their work. Thus, the second contribution of this paper is a complete architecture and a fully functional application based on it that can be used to generate timetables in real scenarios (educational centers). We mainly focus on integration aspects of the system and its usability.

*Funded in part by MICINN grant TIN2009-14560-C03-02; by MICINN ref. AP2007-02484 (FPU Program), for the second author; and by MICINN ref. BES-2010-033262 (FPI program) for the third author.

2 APPLICATION DESIGN

The complete process of timetable construction is divided in several steps: *i*) *Data input*: data about teachers, groups and subjects, as well as teachers' preferences and other constraints are introduced in the system. Most of this information is automatically retrieved. *ii*) *Automatic timetable generation*: the information is processed to obtain a set of constraints and to generate the required set of timetables. *iii*) *Timetable refinement*: the generated solution is shown to the user. Although generated solutions are always complete and have good quality, they can be improved by hand. Finally, the application exports automatically the generated timetables to the AAOS.

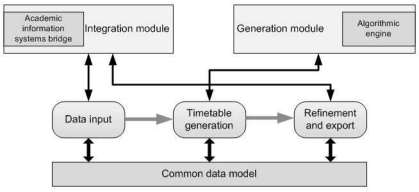

Figure 1: Application architecture.

Figure 1 shows our modular architecture designed to manage the whole process. Although a common data model is used, the import and export tasks need to interact with external systems. The generation engine has been designed to be independent of the main application to allow the evolution of the generation system. Therefore, the application is built over two

modules that hold the "external" information management, and that are integrated by the transformation of the information to the common data model. The *integration module* or data exchange module deals with the data exchange between the application and the AAOS. The *Timetable generation module* is in charge of the generation of a complete solution, according to the user-defined constraints. This generation engine can also work as a standalone system.

Constraints, preferences and other timetable elements were taken into account in the modeling of the timetable-specific domain. Most of the common timetable constraints are integrated in the generation engine used. However, the common model does not only use these restrictions, but it also manages many combinations of them. The interaction with the system is done by means of an abstraction layer that makes the structure of the model completely transparent to the user. For instance, some constraints are internally modeled as multiple simple constraints, as most of the generation tools do. Nevertheless, the user interface is based on user-defined constraints, that are easier to understand for the user. This implies that user interaction will be exhaustively checked for data integrity, given that the user actions should be limited as less as possible.

3 INTEGRATION MODULE

This module allows the application to exchange information with the AAOS that store information of all the educational centers in a region. Although at the moment just the AAOS of the Galician government (*Xade*) is supported, this module can be easily extended. An AAOS usually stores all the information about the centers and provides a web interface to introduce the final timetables. Note that, for example, Xade does not provide a web service to retrieve/store data, so this task must be performed by hand. Moreover, many times generated timetables contain elements that are not accepted by the official data model and the timetables have to be adapted. The goal of this module is to improve the dealing with these AAOS.

In order to support those AAOS that do not provide a web service to access the stored information, we use a web automation engine to import/export data from/to the AAOS. This engine uses a data scraper based on HTML parsing. The engine contains multiple importers/exporters that deal with the different entities in the domain model. The integrity of the individual entities is controlled by the importers. Moreover, the importers can communicate with each other to infer the relationships between the external entities, and map them to our common model.

To export data to the AAOS a more complicated process is performed because the generated timetables can contain elements that are not allowed in the AAOS. Most of the elements that can not be created in the AAOS are recognized and transformed automatically by the integration module (in fact, many "illegal" elements are internally represented as legal elements with one or more constraints). In case some of the information has changed the module uses heuristics to match the generated model with the information found at the moment in the external application, using statistics obtained from previous mappings and some domain-specific information (e.g. official acronym tables). In most of cases, the inferences done by the application do not need further adjustment by the user. Anyway, the user is allowed to manually make any changes before the export is performed.

4 TIMETABLE GENERATION MODULE

The school timetabling problem consists of finding in which period of time a given teacher has to teach a certain subject to a specified group. Additionally, two different types of constraints have to be considered during this process: *hard* and *soft* constraints. The *hard constraints* must be satisfied to obtain a feasible solution. We consider the following types: *overlaps* (a class can not be taught by more than one teacher in the same period, and classes sharing resources can not be assigned to the same period), *simultaneity* (two classes are taught by different teachers at the same time, because of the division of a group into subgroups, for example), *unavailability* (periods when a class can not be given or when a teacher can not teach), *consecutiveness* (some lessons should be taught in two or more consecutive time periods). The *soft constraints* measure the goodness of a solution. We consider: *overuse* (the number of periods per day in which a teacher gives his/her lessons, over a maximum), *underuse* (the opposite of the previous one), *holes* (the number of *empty* periods between two consecutive ones where a teacher is assigned a class), *splits* (the number of periods between two non consecutive assignments to a same class in the same day), *groups* (assuming a specified maximum of periods per day for a teacher-class association, it considers the number of exceeding periods in such day), *undesired* (number of periods in which a teacher would prefer not to teach, but is assigned a class).

4.1 Algorithms Description

The school timetabling problem can be defined by a *mathematical formulation* (Schaerf, 1999), which describes the feasible regions through a solution space; and an *objective function*, which allows to lead the search process towards an optimal solution. The constraints that define such space are the *hard con-*

straints. On the other hand, each *soft constraint* has usually an assigned weight and contributes to the value of the objective function. However, in practice, we are interested in managing both kind of constraints with a function called *cost function* that assigns a cost to each solution depending on the number of failed constraints. Each constraint has associated a weight (higher in hard constraints to lead the search process towards feasible regions). Then, the objective of the school timetabling generation problem is to find the solution that minimizes such function.

As we noted before, the timetabling problem we consider admits a mathematical programming representation, so an exact solution could be obtained by applying well-known techniques in this field. Nevertheless, it has been shown that this is a NP-complete problem (de Werra, 1985; Even et al., 1976). In practice, the high dimensionality of the problem makes it impossible to find an exact solution, and approximated methods are needed to tackle it (de Haan et al., 2006; Schaerf, 1999; Keppler and Erben, 1996). We propose and study the performance of three different algorithms based on heuristic search techniques.

We use a *teacher-oriented* representation of the problem, and therefore a timetable is represented as a two-dimensional matrix where each cell (i, j) contains the class given by the i^{th} teacher at the j^{th} period. This representation avoids implicitly the case of a teacher giving more than one class at a same time. We also define two kind of moves: *i) simple-moves*, that are obtained by swapping two distinct values in a given row of the timetable; and *ii) double-moves* that are the combination of two simple-moves when the first move leads to an unfeasible scenario.

RNA Search: Following the RNA local search technique, we have developed an algorithm which, starting from an initial solution, iteratively moves from a solution to another doing *double-moves*. It keeps track of the current best solution at each stage. The process is repeated until there are no improvements during a given number of iterations.

Genetic Algorithms: Genetic algorithms base their operation on the evolution mechanism. Starting with a *population*, a set of *individuals* or potential solutions (timetables, in our case), best candidates are selected based on their *fitness* value (given by the cost function). These will be the *parents* of a new group of individuals obtained by modifying the previous ones by using *crossovers* and *mutations*, which allow exploring the search space and guaranteeing the genetic diversity, respectively. The new population will be processed in the same way in the next iteration.

In our case, a *crossover* consists of selecting a certain number of rows from each of the parents, given by a randomly chosen crossover point in the timetable. A *mutation* is done by means of a simple move. We have developed two different approaches: *i*) *Tournament* (GT): starting with a population randomly generated, two pairs of individuals are selected and then the best candidate of each pair is chosen. The parents of the next generation are obtained in this way; *ii*) *Four Children Tournament* (GT4C): once two individuals are chosen to form a pair they are discarded, so they will never be chosen again.

Furthermore, we have combined the previous algorithms obtaining two hybrid ones, *Tournament & RNA* (GT & RNA) and *Four Children Tournament & RNA* (GT4C & RNA). Moreover, we have created some variants of our genetic-based algorithms following the strategies: *v1)* To increase dynamically the number of mutations after a given number of iterations without improvements; *v2)* To apply a more elitist selection technique of the best candidates, so reducing its proportion to a given percentage, and *v3)* Not to eliminate the loser candidates when we work with a GT4C phase, in such a way, all the individuals could be parents at least once.

4.2 Algorithms Evaluation

To properly evaluate the algorithms performance, different experiments were run over two sets of synthetic test cases with different size and configurations. Data set A is composed of 10 files of 6 groups, 70 classes and 15 teachers. Table 1 shows the average value of unsatisfied constraints and the standard deviation for 10 runs, limited in time to 30 minutes. Results show that most algorithms obtain good results and perform quite similarly, with the exception of a small group that perform very badly: variants $v1$ and $v1+v2$ of the genetic algorithms and variant $v3$. The best choices seem to be RNA and GT4C.

Table 1: Results for the collection A.

Algorithm	Constraints				Algorithm	Constraints			
	Avg		Std			Avg		Std	
	hard	soft	hard	soft		hard	soft	hard	soft
RNA	0.16	0.60	0.09	0.39	**Var: v2**				
GT	0.22	1.45	0.22	0.32	GT	0.26	0.74	0.14	0.45
GT4C	0.31	2.05	0.29	0.68	GT4C	0.23	0.74	0.16	0.44
GT & RNA	0.44	2.30	0.50	1.35	GT & RNA	0.44	1.74	0.46	0.92
GT4C & RNA	0.38	2.46	0.33	0.93	GT4C & RNA	0.33	1.41	0.34	0.84
Var: v1					**Var: v1+v2+v3**				
GT	6.94	23.57	3.81	3.88	GT4C & RNA	0.49	2.06	0.42	0.70
GT4C	7.07	23.32	3.99	3.84	**Var: v1+v3**				
GT & RNA	0.34	2.74	0.29	0.96	GT4C & RNA	0.43	4.33	0.42	1.55
GT4C & RNA	0.28	2.74	0.25	1.20	**Var: v2+v3**				
Var: v1+v2					GT4C	0.28	0.75	0.22	0.43
GT	3.99	15.57	2.86	2.85	**Var: v3**				
GT4C	5.53	18.90	3.50	2.23	GT4C	27.62	42.19	7.52	3.97
GT & RNA	0.47	1.90	0.48	0.96					
GT4C & RNA	0.41	1.76	0.43	0.73					

After discarding the algorithms that behaved badly in the previous case, we studied the performance of the remaining over a bigger scenario (B) involving 27 groups, 333 classes and 71 teachers. The different algorithms were run 4 times, limiting the

time to 5 hours each. Table 2 shows that variant *v2* obtains very good results, especially version GT & RNA, that reduces the variability. RNA and variant *v1+v2+v3* behave also well, but they lead to a high number of unsatisfied *soft constraints*.

Table 2: Results for the scenario B.

Algorithm	Constraints				Algorithm	Constraints			
	Avg		Std			Avg		Std	
	hard	soft	hard	soft		hard	soft	hard	soft
Original					**var: v2**				
RNA	0.88	63.58	0.82	3.77	GT	1.94	26.04	1.39	4.17
GT	2.88	67.21	2.11	8.42	GT4C	2.69	34.08	1.53	6.53
GT4C	3.88	71.04	2.26	5.85	GT & RNA	0.94	30.38	0.98	2.77
GT & RNA	1.50	62.54	1.54	6.50	GT4C & RNA	1.94	35.71	1.44	4.12
GT4C & RNA	1.75	70.33	1.35	3.95	**var: v1+v2+v3**				
var: v1					GT4C & RNA	1.00	52.67	0.73	5.71
GT & RNA	1.75	63.75	1.19	4.71	**var: v2+v3**				
GT4C & RNA	2.25	69.04	2.12	4.49	GT4C	2.75	49.88	1.71	4.64

Algorithm	Constraints			
	Avg		Std	
	hard	soft	hard	soft
Original				
RNA	0.88	5.95	0.74	1.72
GT	1.05	6.90	1.01	2.92
GT4C	0.55	6.37	0.72	1.83
GT & RNA	1.63	8.10	1.07	2.71
Var: v2				
GT4C & RNA	2.20	8.42	1.95	3.69

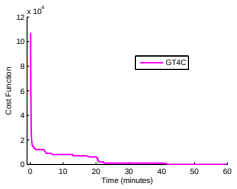

Figure 2: Results for real data.

To complete the study, we checked the performance of the best algorithms with real data obtained from a Secondary school. Left part of Figure 2 shows that all the chosen algorithms reach good results; however, we can highlight those of RNA and GT4C. Notice that the impossibility of satisfying 100% of the *hard* constraints is usually caused by the incompatibility of the constraints imposed by educational centers. In these cases, we are interested in reaching the least penalized solution. The right plot of the same figure shows the evolution of the cost function of the best solution over time.

5 USER INTERACTION

As we explained before, our application manages the complete process of school timetable construction. Most of the traditionally time-consuming tasks are performed automatically. The phases in which the process can not be automated are the data introduction and the solution refinement. To simplify these steps, we worked in collaboration with domain experts to extract the real needs of the users and the functionalities that could enhance these tasks.

Typing the data set for the timetable generation is probably the most tedious task in the existing tools. In our application, the user interface has been designed to make easier this step. It allows the user to create some arbitrary elements that do not correspond to usual situations. For instance, a teacher giving lessons of different subjects at the same time is usually a non-consistent situation, but in many centers this situation is a real problem to be solved. These actions are managed by the application by building a consistent solution based on fictitious elements. Additionally, the data input interface helps the user to debug the information introduced. Whenever a potential problem is detected the user is notified, and guided over the reasons that could have caused it. The model transformation that is applied to the user constraints keeps traceability between the underlying constraints and the original user domain objects. Only actions that cause a non-consistent state of the timetable set are blocked, while potentially malformed constraint sets are admitted. By limiting as little as possible the user capabilities in the data introduction stage, the application provides maximum flexibility to the users.

When a solution is finally generated, the final result has to be verified and validated by the user before being exported. Most of the existing tools have a basic interface for this, in some cases including an "advisor" that suggests changes. However, in most of cases any change in an existing solution usually requires many changes in several timetables to adjust a single lesson. Thus, our application proposes a different perspective of the timetable refinement: it relies on the user to make the decisions and provides an user interface that gives a global vision of the solution set. The screen is split to show at the same time groups and teachers timetables. It can display at all times a subset of relevant timetables, that is updated according to the user actions. For instance, when the user changes the timetable for a teacher, the application will automatically update the relevant subset of timetables to focus also on the group or groups that contain the conflicting time period (or those in which new conflicting periods appear). The conflicts that involve each timetable are shown using a visual code to immediately identify the problems.

User actions during the solution refinement are not limited at all. Thus the user can create non-feasible solutions. The system checks in each step the integrity of the resulting solution.

REFERENCES

de Haan, P., Landman, R., Post, G., and Ruizenaar, H. (2006). A four-phase approach to a timetabling problem in secondary schools. *PATAT'06*, pp. 423–425.

de Werra, D. (1985). An introduction to timetabling. *Eur. Journal of Operational Research*, 19(2):151–162.

Even, S., Itai, A., and Shamir, A. (1976). On the complexity of timetable and multicommodity flow problems. *SIAM Journal on Computing*, 5:691–703.

Keppler, J. and Erben, W. (1996). A genetic algorithm solving a weekly course-timetabling problem. *PATAT'96*, pp. 198–211.

Schaerf, A. (1999). Local search techniques for large high-school timetabling problems. *IEEE Transactions on Systems, Man, and Cybernetics*, 29:368–377.