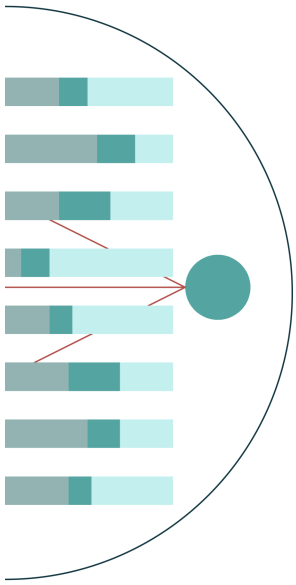
Constraint-Informed Information Systems in Space Management Optimization

Irene Rodrigues
Laboratory of Informatics, Systems and Parallelism (LISP), University of Évora, Portugal
ipr@uevora.pt

Rui Quaresma
Center for Advanced Studies in Management and Economics (CEFAGE), University of Évora, Portugal
quaresma@uevora.pt

Daniel Diaz
Centre de Recherche en Informatique (CRI), University Paris 1, Panthéon-Sorbonne, France
daniel.diaz@univ-paris1.fr

Salvador Abreu
Laboratory of Informatics, Systems and Parallelism (LISP), University of Évora, Portugal
spa@uevora.pt



ABSTRACT Declarative techniques such as Constraint Programming can be very effective in modeling and assisting management decisions. We present a method for managing university classrooms which extends the previous design of a Constraint-Informed Information System to generate the timetables while dealing with spatial resource optimization issues.

We seek to maximize space utilization along two dimensions: classroom use and occupancy rates. While we want to maximize the room use rate, we still need to satisfy the soft constraints which model students' and lecturers' preferences. We present a constraint logic programming-based local search method which relies on an evaluation function that combines room utilization and timetable soft preferences.

Based on this, we developed a tool which we applied to the improvement of classroom allocation in a University. Comparing the results to the current timetables obtained without optimizing space utilization, the initial versions of our tool manages to reach a 30% improvement in space utilization, while preserving the quality of the timetable, both for students and lecturers.

KEYWORDS: constraint-informed information systems, optimization, constraint logic programming

Introduction

Declarative methods and tools have long been a hallmark of Artificial Intelligence-enabled applications. One of these is Constraint Programming, which consists of modeling a problem in terms of variables which may take values over a specific domain, and relations amongst them which are called constraints. When dealing with certain classes of problems, Constraint Programming techniques are able to provide very efficient solvers while retaining the ability to formulate the problem declaratively, relying on high-level concepts which may be very close to the application domain. Constraints may be used to formulate two kinds of problems:

- Constraint Satisfaction Problems (CSPs),
- Constraint Optimization Problems (COPs).

The former finds a solution to a problem, while the second does so while ensuring that some form of optimum is reached. Both are search problems and can cope with encodings on arbitrarily complex domains.

The UML framework (Booch, Rumbaugh, and Jacobson,

ISSN

1999) proposes a set of methodologies for designing incremental and scalable complex applications, balancing expressiveness, incrementality and abstraction, providing a formalism for data and process modeling as well as code generation.

Space management in higher education institutions (HEIs) is a recognized research subject, with published work by several authors (Kilner Planning, 2006; Kilner Planning and London Economics, 2006; Beyrouthy, Burke, McCollum, McMullam, and Parkes, 2010; Ani, Tawil, Musa, Tahir, and Abdullah, 2012; Abdullah, *et al.*, 2012; Abdullah, Ali, and Sipan, 2012). One of the topics associated with space management in HEIs is the preparation of class schedules, which has also been the focus of research and applied work (Beyrouthy, *et al.*, 2006; Rudová, Müller, and Murray, 2011).

The preparation of class schedules is a process with a very significant impact on the teaching activities of universities, since it entails managing physical resources which are not amenable to change at will, such as the capacity of a classroom. Moreover, this process may be overconstrained by contradictory requirements, for instance: from the student's point of view one wants to concentrate classes in the same part of the day while for the lecturer one wants to minimize the number of days, which could imply having classes both in the morning and in the afternoon, thereby defeating the first constraint.

An existing framework can be used to solve this sort of problem. We claim that the inclusion of declarative features, such as Constraint processing, was instrumental in easing the development of the augmented system, including its implementation but also the potential for its formal verification.

In a previous work (Rodrigues, Matos, Abreu, Deneckère, and Diaz, 2013), we claim that Constraint Programming is a useful and effective addition to the UML framework. To do so we described an information system construction framework, based on Constraint Logic Programming (Diaz, Abreu, and Codognet, 2012) extended with persistence (Abreu and Nogueira, 2006; Abreu, 2001) and contexts (Abreu and Diaz, 2003).

Large scale information systems may be specified, modeled and designed using UML-based instruments such as Rational Rose or the Eclipse UML Tools. These techniques and tools guide the practitioner and greatly simplify and improve the reliability of IS design and maintenance. Nevertheless, one aspect remains alien to the UML perspective: how to establish arbitrarily complex relations between otherwise unrelated components of a system, so as to ensure the correctness of the global state? One may specify integrity constraints, rules which control and ensure data validity, but these appear as verifications, and cannot normally be used to actually generate data.

Including constraints among components in the UML framework is a recognized objective: there are several initiatives which specify and implement higher-level frameworks, such as the Object Constraint Language (OCL), with the intention of embedding these into the UML modeling formalism. OCL, however, is most of the time pure documentation: there are hardly any working implementations and it ends up not being used in modeling tasks.

Generically, constraints have long been recognized as a useful concept in modeling languages (e.g., for requirements engineering) but they are also used within regular applications, in the role of combinatorial problem solver or optimizer. As a consequence of its expressiveness and flexibility, Constraint Programming (CP) has been successfully used for modeling in several domains: (Rossi, van Beek, and Walsh, 2006; Van Hentenryck, 1989) air traffic flow management (Chemla, Diaz, Kerlirzin, and Manchon, 1995), planning (Garrido, Arangu, and Onaindia, 2009) or product-line models (Mazo, Salinesi, Diaz, Djebbi, and Lora-Michiels, 2012; Djebbi, Salinesi, and Diaz, 2007; Mazo, Salinesi, Diaz, and Lora-Michielis, 2011), to name but a few. Already over a decade ago, CP was even identified by the ACM as "one of the strategic directions in computer research" (Barták, 1999).

Constraint Logic Programming (CLP) is a declarative technique related to CP, which adds the possibility of constructing and driving the constraint solving process by means of logic goal satisfaction. The declarativeness of CLP makes it suitable for even very complex modeling: constraints may be thought of as a set of relations (e.g., equations) yet retaining an efficient runtime execution.

Our claim is that constraints are useful in designing and implementing components of an information system, and that the latter will benefit significantly from being constraint-informed. To make and illustrate the point, we describe an actual application – the development of a timetabling system for HEI's, which incorporates several forms of resource optimization.

The rest of the paper describes the framework and the running example – a University-wide timetabling information system – both its structure and some implementation aspects, which we proceed to critically analyze. We describe CLP and the GNU Prolog system (Diaz, Abreu, and Codognet, 2012) and continue with how we used it to model parts of the application with constraints. Finally, we put this work in

ISSN

context and make considerations regarding further evolution.

Persistent Contextual Constraint Logic Programming

The design and implementation of this web-based information system component relies on the ISCO (Abreu, 2001) language and tools, which starts from the CLP base provided by GNU Prolog (Diaz, Abreu, and Codognet, 2012) and extends it with the mechanisms necessary to enable Object-Oriented program structuring, by means of the Contextual Logic Programming language CxLP. Furthermore, the sound and expressive Prolog base is augmented with persistence in the form of the ISCO mediator extension, which resorts to external data providers such as relational databases, to persistently store and query for structured data. ISCO implements a form of Datalog (Ceri, Gottlob, and Tanca, 1989) for CxLP which may be externally represented in several ways: for instance as an object-relational database accessed in SQL or an RDF dataset queried using SPARQL. This ability makes ISCO programs very flexible, as they may query and reason about data sources of many origins. The general organization for ISCO programs is shown in Figure 1.

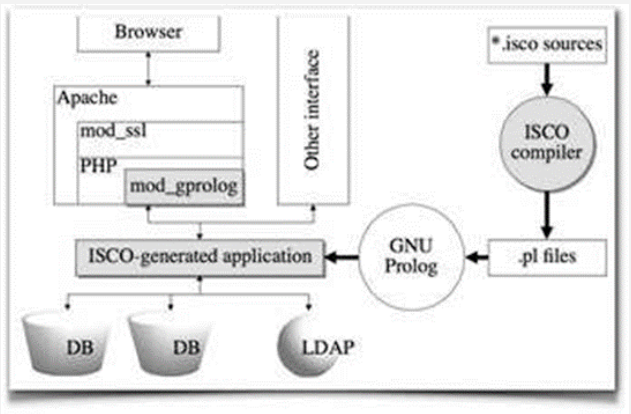


Figure 1 - Computational Environment for ISCO Applications

The basis on which ISCO applications run is the GNU Prolog implementation of CLP(FD), i.e. Constraint Logic Programming over Finite Domains, an instance of the CLP(X) scheme which is very appropriate for representing discrete problems. One important feature that CLP(X) brings is a sophisticated hybrid search procedure: on the one hand it behaves like Prolog, with variables being non-deterministically (partly) bound, relying on backtracking to enumerate all the possible values. On the other hand, we have constraints which implicitly tie variables, in a way which proactively narrows the admissible set of values for a given variable, just because it occurs in a constraint which was triggered because some other variable got narrowed. In a sense, we mix a-posteriori search-space exploration (via backtracking) with a-priori search-space pruning (via constraint propagation).

GNU Prolog provides a framework for expressing constraints which is generic and caters to many requirements: it centers on the concepts of Constraint, Variable and Domain and follows the schema depicted in Figure 2. A remarkable feature of this diagram is the four subclasses of Constraint which are sufficient to model several behaviors: arithmetic and Boolean have their usual meaning, but the other two deserve further explanation:

- a Symbolic constraint is used for global constraints: it is a sort of "hooks" for the inclusion of otherwise complex or inefficient constraints. Examples of symbolic constraints include at most which states that at most a stated number of a list of constraints will hold, and all different which stipulates that all the listed variables are pairwise different;

- a Reified constraint relates the value of a constraint variable with another constraint. This is a very powerful feature that allows one to encode meta-level or disjunctive constraints or, more generally, provide meta-level "switches" or "sensors" to enable, disable or monitor groups of constraints.

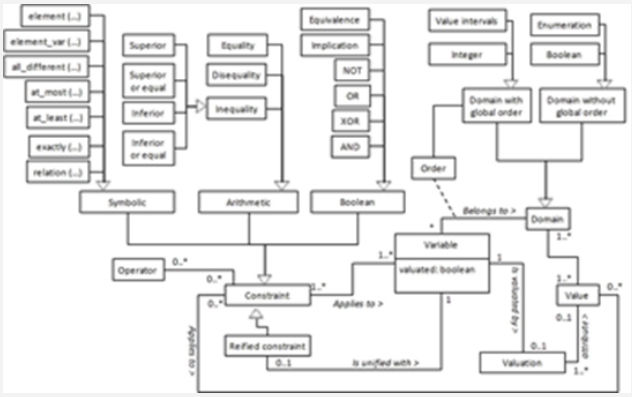


Figure 2 - Constraints in GNU Prolog

For this application we added a new built-in predicate to GNU Prolog, call_with_timeout/2, which runs a goal until success or a given time has elapsed. Doing so has proved a good choice for the situation at hand, where the web interface is not expected to have the operator wait indefinitely.

Class Scheduling in HEIs

Class scheduling is at the heart of the activity of HEI's: it is a mission-critical task, with a clear impact on the physical resource usage but also on the perceived performance of the organization. In the specific case of the University of Évora, the process of preparing the schedules is done by means of an in-house application that satisfies a set of underlying

ISSN

general requirements governing timetable preparation, which include:

- Classes within a week are organized in blocks of 90 or 120 minutes each;
- Classes are concentrated in a particular part of the day, either the morning or the afternoon, to ensure the existence of free time for students to study on their own;
- Classes occur in the 08:00-20:00 hour range, with the exception of the 13:00-14:00 period which is reserved for lunch;
- The week goes from Monday to Friday;
- From the standpoint of the lecturers, classes tend to concentrate in at most two distinct weekdays.

In our previous work, we extended the design methodology to include requirements specified as constraints, which led us to develop an application (Rodrigues, Matos, Abreu, Deneckère, and Diaz, 2013) that has proven able to meet the general objectives outlined above.

However, subsequent analysis of schedules generated by the application has revealed that, from the point of view of space management, we tend to get a relatively low level of frequency of use and rate of occupancy of classrooms. This behavior translates to an inefficient use of the available physical resources, with direct budgetary consequences.

Thus, and in line with (Kilner Planning, 2006), we worked from the basis of the implemented framework, extending it with a new utility function which aims at maximizing the use of space along the dimensions we mentioned: the temporal frequency of use, the physical rate of occupancy of the premises. Other concerns which we try to cater to are the avoidance of "holes" in students' schedules as well as a general minimization in the number of class days for students.

We show a UML class diagram for the timetabling component of the information system in Figure 3, similar to that found in (Rodrigues, Matos, Abreu, Deneckère, and Diaz, 2013).

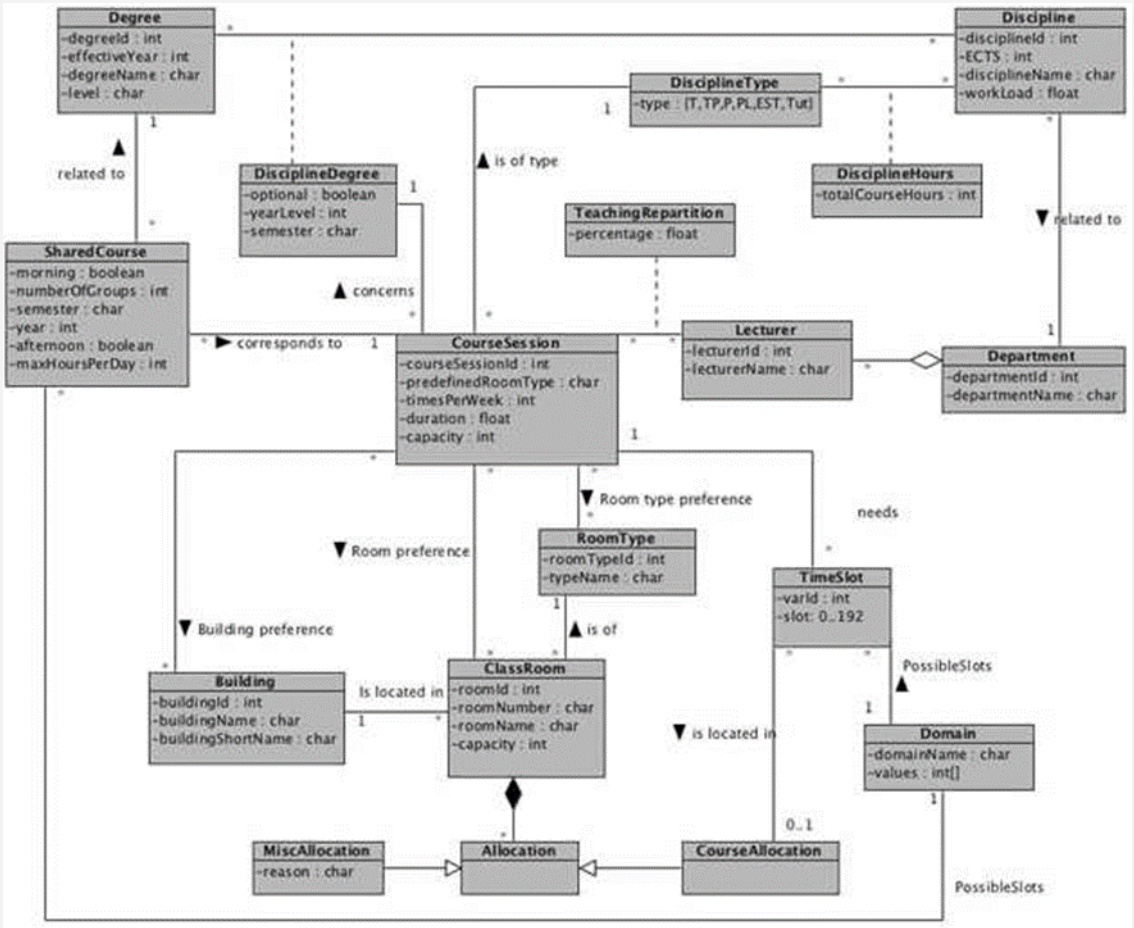


Figure 3 - Timetabling UML Class Diagram

ISSN

The Timetable Manager

The timetable manager deals with all aspects of specific timetables: the generation, editing, display and other bookkeeping tasks. The timetable manager has the option to manually edit partial timetables and to automatically generate partial or complete timetables. The automatic timetable generation can be done incrementally by collecting a set of class (course) variables, as required for a given degree curriculum.

The system enforces the timetable hard constraints both for the automatic generation and for manual editing. In the latter case, it makes no sense to take into account the soft constraints, though it is possible to subsequently evaluate the solution after it has been persistently stored.

The timetable may be incrementally built, resulting in a partial schedule that is represented in a way which is always guaranteed to be coherent. This allows us to:

1. Display the partial timetable,
2. Continue to build towards the final timetable using the values of the variables already present in the database as hard constraints,
3. Evaluate the timetable: the evaluation of a partial or completed timetable can be done by applying the utility functions that model the soft constraints,
4. Optimize the represented timetable using the soft constraints, according to a utility function.

The soft constraints model preferences, i.e., defeasible constraints, which include:

- The students' timetable ought to be compact. If a timetable has many "holes," student time may be wasted if the holes are not long enough for them to be used as self-study periods.
- In order to evaluate these preferences we count the free slots between the students' classes. We can view and represent a timetable as the set of slots it occupies in the domain [1 : : : 192][1]

$$F_{Comp}=192 - \sum_{i=1}^{N} elem(i+1) - elem(i) + 1$$

- The students' timetable ought to be either all in the morning, or all in the afternoon. This preference is important in order to allow the students to organize their self-study time.
  - Function $F_{MorningDays}$ represents the number of mornings in a student's timetable.

$$F_{MorningDays}=\#\{x/elem(i)|_{32} < 11\}$$

  - Function $F_{AfternoonDays}$ represents the number of afternoons in the student's timetable.

$$F_{AfternoonDays}=\#\{x/elem(i)|_{32} > 12\}$$

  - Function F_{Days} represents the number of days in the student's timetable.

$$F_{Days}=\#\{x/elem(i)|_{32}\}$$

  - To prefer all days in the morning or all days in the afternoon we maximize the function F_{MA}.

$$F_{MA}=max(F_{AfternoonDays}, F_{MorningDays})$$

  - This soft preference can be defined for each year of a degree, for instance first year students have their classes in the morning, second year in the afternoon. To obtain this preference we use $F_{AfternoonDays}$ or $F_{MorningDays}$ as a utility function.
- A student's timetable should be kept occupied every day of the week: let the function F_{Days} represent the number of days in which the student has classes:

$$F_{Days}=\#\{x/elem(i)|_{32}\}$$

  By maximizing F_{Days} we obtain the desired effect: we prefer timetables that have classes on all days.
- Students should have no more than 4 class hours per day. This constraint can easily be directly expressed in our representation.

Lecturers' timetables also define preferences, such as:

- The days with classes should be less than or equal to three, so as to leave time for other duties, such as research and management tasks.
- The days with classes should be consecutive. This enables the lecturer to engage in other activities which may take longer.

For classrooms we can define a utility function in order to represent the university room occupancy, which we will call *Univ_Occupancy*, and is defined as the sum of the occupancy of

[1]This representation has 192 discrete time slots which cover the entire week. See (Rodrigues, Matos, Abreu, Deneckère, and Diaz, 2013) for a more detailed description of the model.

ISSN

each relevant room:

$$Univ\_Occupancy = \sum_{i=room\ 1}^{roomN} Room\_Occupancy_i$$

The room occupancy is defined in terms of the fractional occupancy for each time slot:

$$Room\_Occupancy_i = \sum_{j=slot_i}^{slot_N} \frac{Occupancy_{(i,j)}}{\#\{S/S \in slots\ (Room_i)\}}$$

Finally, the occupancy for a given slot is given by the ratio of the number of applicable enrolled students to the number of seats:

$$Occupancy_{(i,j)} = \frac{StudentsNumber_{(i,j)}}{NumberPlaces_i}$$

Where:

- $NumberPlaces_i$ is defined for each room *i*, and indicates the capacity (in seats) of the room.
- $StudentsNumber_{(i,j)}$ is computed from the available data, using the information associated to the particular class occurrence.
- $slots(Room_i)$ is the projection of the timetable of room *i* onto *slots*, a set with values in the domain [1 : : : 192].

The *Univ_Occupancy* is the utility function that will be optimized for, in order to better use the physical (space) resources. Note that this optimization gives rise to a solution with a better use of the classroom seats, some rooms may become empty, without any classes in it.

Room utilization can be improved if:

- The number of classes per course is optimized, taking into account the room capacity and the number of students. We are able to split or merge specific classes. At this time, the application expects that the number of classes for each course has been previously assigned and cannot be changed.
- Opening and closing university rooms: we can close some rooms, thereby freeing up the space for other activities. This can reduce the operating costs for a course, as classrooms have fixed costs which are independent of their actual utilization rate.

The application can be used to optimize for the minimal number of rooms necessary for classes. To do so, we need to optimize (maximize) with a utility function that grows with the number of rooms without any assigned classes.

Optimization

The application is able to optimize the university timetable, according to a utility function that enables us to evaluate and compare timetable instances.

The optimization can be done gradually since the current best solution is always kept in persistent storage, which allows for an incremental usage pattern.

The optimization currently has four parameters:

- The *set of (course) classes* that we want to optimize for.

  We can either choose to optimize all timetables, all classes for the entire university, or we can specify a subset of the courses, the optimization will be done just by changing the values of the classes for these courses. Similarly, we can give a set of classrooms to optimize, the optimization will be done just by adjusting the specific classes which take place in these rooms.

  The same may be done for a set of teachers, for a set of courses or, ultimately, any formulation that can be used to specify a set of timetable classes.

- The *utility function* we want to use.

  The utility function defines the preferences that we have for the timetables. At present, multiple preferences are obtained by adding the functions specified by the user.

- The *number of classes* we may want to change the value of in each step of the optimization.

  A random set of classes is computed and new solutions will be tested for this set. The obtained timetables will differ on the number of value slots which are changed from the initial configuration: these may be both times and assigned rooms.

- The *number of iterations* is a relevant input parameter due to the size of the search space for this problem, which can be very slow to explore. We can replace this parameter by a timeout.

We use Hill Climbing for the optimization algorithm. It starts with the current timetable, a solution. A set of classes is randomly chosen and their values are recalculated in order to obtain a new candidate solution. Whenever an improvement is obtained, the new timetable is updated in persistent storage, as shown in Listing 1.

ISSN

Listing 1 - Listing of the Optimization Function

```
1 optimize(Classes, UtilityF, NClasses, Iterations)
2 {
3    Value = evaluate(Classes, UtilityF);
4    for (i = 0; I < Iterations; ++i) {
5          choose (NClasses, Set, Vars);
6          Vars_Value_Max = value (Vars);
7          clear_value (Vars);
8          COP (Classes, UtilityF, ValueS);
9          if (ValueS > Value)
10                   writeDB (Classes);
11           else
12                   writeDB (Vars_Value_Max);
13   }
14 }
```

The Constraint Optimization Procedure COP (see Listing 2) uses the finite domain constraints provided by the GNU Prolog solver and ISCO to interface the persistent storage, supplied by a PostgreSQL object-relational database, interfaced via ISCO. Notice that COP operates by selecting the best configuration which may be obtained within a given time limit.

Listing 2 - Listing of the Constraint Optimization Procedure

```
1 COP(Classes, UtilityF, ValueS)
2 {
3    CollectVars (Classes, Vars);
4    HardConstraints(Vars);
5    repeat {
6          labeling(Vars, random);
7          Value = evaluate(Classes, UtilityF);
8    } until (Value > ValueS) OR timeout
9 }
```

Examples

We now consider a few examples from the actual courses carried by the School of Social Sciences of the University of Évora.

Optimizing for students' preferences

Consider the sample timetable shown in Figure 4: it has several deficiencies which are manifest as violations of soft constraints. For instance, it has classes both in the morning and in the afternoon, but also other issues which are not explicit in the printout, such as underfilled classrooms.

After one step of optimization, which may subsequently be hand-corrected, the system proposes the timetable on Figure 5 which is better, but still offers room for improvement.

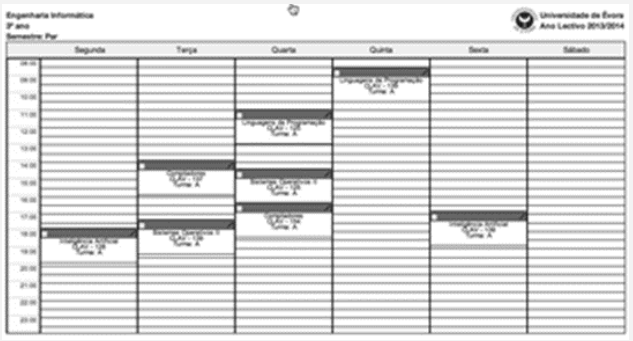


Figure 4 - Initial Timetable

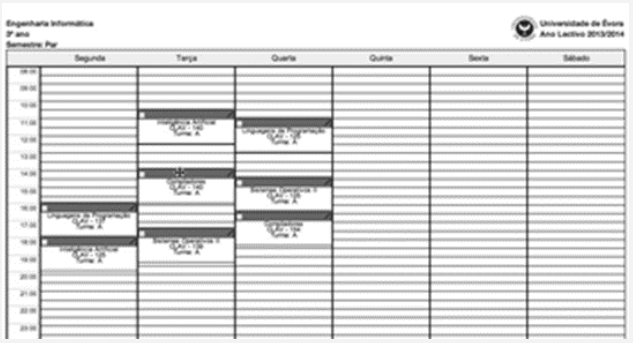


Figure 5 - Timetable after one Automated Improvement Step

By selecting the "optimization step" action in the user interface, the operator brings the timetable to that shown in figure 6, where most classes occur in the morning.

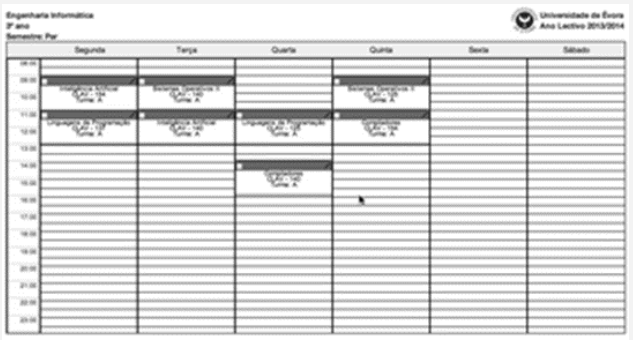


Figure 6 - Optimized Timetable

Optimizing for room occupancy and students' preferences

In order to improve room occupancy, we have to evaluate each room for each timetable as well as take into account the other utility functions for each university course, which takes us to a problem of global optimization.

Table 1 presents the data for the rooms of one of the buildings of the University, one where the courses of the School of Social Sciences hold most of their classes. These data reflect the actual timetable of these rooms in the second semester of school year 2013/14, and it was generated and

ISSN

edited with this application.

Table 1 - Rooms occupancy rate before optimization

Room	Mon	Tue	Wed	Thu	Fri	TOT
103	53%	86%	75%	86%	79%	75%
104	40%	57%	85%	62%	9%	59%
105	81%	59%	50%	59%	51%	61%
106	62%	83%	81%	86%	53%	77%
107	75%	65%	86%	70%	81%	74%
110	53%	40%	72%	62%	75%	58%
114	47%	49%	55%	30%	18%	45%
115	65%	32%	61%	48%	9%	47%
118	43%	69%	66%	51%	40%	56%
119	42%	49%	58%	45%	63%	51%
120	70%	57%	57%	56%	71%	61%
121	15%	69%	74%	67%	9%	56%
122	35%	86%	73%	69%	77%	67%
136	88%	85%	51%	88%	18%	74%
205	89%	91%	72%	89%	89%	87%
206	71%	47%	59%	63%	9%	57%
208	71%	65%	68%	81%	50%	69%
272	45%	51%	52%	68%	9%	52%
295	47%	78%	51%	81%	86%	68%
296	27%	80%	59%	65%	84%	62%
297	68%	42%	44%	55%	54%	53%
298	43%	57%	48%	72%	9%	54%
TOTAL	56%	61%	63%	68%	55%	61%

Table 2 presents the data for the same rooms after some optimization steps:

- 20 runs of the procedure:

  optimize (Classes; UtilityF; NClasses; Iterations)

- *Classes* represents the set of classrooms being optimized for. These are made explicit in the set enumerated in Table 1.

- *UtilityF* is the utility function, which in this case is defined as:

$$UtilityF = F_{Comp} + F_{MA} + F_{Univ\_Occupancy}$$

  This function takes into account the following preferences:

  ◊ From the point of view of the students: F_{Comp}, which favors a compact timetable and F_{MA}, which benefits classes either in the morning or in the afternoon, but not both.

  ◊ From the room occupancy point of view: $F_{Univ\_Occupancy}$, which maximizes room occupancy.

- *Nclasses*: 3.

  The number of variables that change from solution to solution is 3.

- *Iterations*: 100.

  The number of times a new set of variables is chosen.

The solution in table 2 increases the occupancy of those rooms from 61% to 66%, while maintaining the student's preferences in the final timetable[2].

Table 2 - Rooms occupancy rate after optimization

Room	Mon	Tue	Wed	Thu	Fri	TOT
103	58%	86%	79%	84%	79%	76%
104	41%	76%	85%	69%	9%	64%
105	81%	59%	50%	59%	51%	61%
106	66%	83%	81%	86%	53%	77%
107	75%	65%	86%	70%	81%	74%
110	83%	36%	98%	65%	75%	67%
114	47%	52%	53%	30%	18%	46%
115	80%	26%	69%	52%	9%	53%
118	64%	72%	66%	51%	40%	61%
119	42%	49%	58%	45%	63%	51%
120	70%	57%	57%	67%	71%	64%
121	15%	69%	74%	67%	9%	56%
122	38%	86%	73%	69%	77%	69%
136	88%	85%	51%	88%	18%	74%
205	89%	91%	72%	89%	89%	87%
206	71%	58%	74%	71%	9%	65%
208	71%	65%	68%	81%	50%	69%
272	50%	58%	71%	68%	9%	59%
295	57%	88%	51%	88%	86%	73%
296	76%	97%	73%	65%	84%	78%
297	99%	95%	44%	55%	54%	65%
298	9%	94%	70%	77%	9%	65%
TOTAL	62%	69%	68%	70%	55%	66%

The requirement for improvement is materialized by imposing that, in a better solution, the sum F_{Comp} + F_{MA} cannot decrease.

The final timetable is of better quality than the first one, regarding room occupancy and is equivalent to it w.r.t. the degree of satisfaction of student's preferences.

In this example, we actually obtained better solutions with an increase of 30% for rooms occupancy rate, but the stu-

[2] Should the latter not be given the same priority we can reach a room occupancy of over 85%. This is a matter for management policy, as we may tune the utility function to use different weights for each component.

ISSN

dents' preferences utility turned out lower. The decision as to which soft constraints are actually to be upheld has to be made by the course scheduling committee.

Related Work

In (Abdullah, *et al.*, 2012), the authors discuss and propose measures to evaluate the space performance in a HEI. For these authors, the HEI space is one the organization's most expensive assets. In (Ani, Tawil, Musa, Tahir, and Abdullah, 2012) the authors present a study to evaluate and measure the learning space usage rate based on the HEI timetables supplied by the institutions. Space management is an important issue, not only for the usage optimization, but also for the maintenance operations cost of the spaces. For these authors, the learning space must be managed effectively and efficiently so as not to become a burden and ensure there is minimal wasted space in HEIs.

The timetabling problem that we consider belongs to the class of post enrollment problems since the students' hard constraints are imposed by the degrees curriculum, this problem is included in some International Timetabling Competitions such as (Post, Di Gaspero, Kingston, McCollum, and Schaerf, 2013) and there are computational approaches to solve it that can be classified as local search (Cambazard, Hebrard, O'Sullivan, and Papadopoulos, 2012), constructive (Burke, McCollum, Meisels, Petrovic, and Qu, 2007) or combined methods (Müller, 2009). In (Rudová, Müller, and Murray, 2011) a complex course timetabling problem at a large university is discussed providing new insights into the overall timetabling process. They use search for a complete assignment of times and rooms to classes, taking all hard and soft constraints into account. They present methods for modifying a computed solution to deal with changes introduced at a later time with a minimal impact on existing assignments. Our design may be characterized as using a hybrid approach, as we combine a constructive method to build solutions that satisfy the hard constraints and proceed with a local search procedure to satisfy the soft constraints.

Closing Considerations

In this article we have taken a real-life case study: the timetabling problem for a University, which we had already shown to benefit from using constraints in the design and implementation process of the information system. We pushed the envelope by following the same design philosophy when dealing with the more complex issue of optimization for physical space management.

In the process, we mixed the power and expressiveness of the constraint-based local search optimizer for space management, with the flexibility of an incremental user-controlled application interface. It turns out that this mix of reassuring conservativeness and resource-efficient eager optimization strikes an effective balance, which is appreciated by the end-users. Moreover, the end results translate to significant savings in physical resource usage, with the ensuing economic and organizational benefits.

The timetabling application is presently managing over 60 curricula, totaling over 2500 courses taught by about 500 faculty members in more than 200 classrooms distributed over 10 different buildings. The information system and constraint solver run very well on a low-power virtual machine, meaning that response time is barely perceptible to solve a single timetabling problem.

Because it is integrated with a full CLP engine, and when compared with systems which rely on libraries such as IBM CPlex or Gecode (Tack, Lagerkvist, and Schulte, 2008) our approach has the advantage that we can conveniently do meta-reasoning, as the CSP being solved is explicitly constructed by a Logic program. So far we have developed a flexible timetabling information system, in which the characteristics of a solution are declaratively specified by constraints. This is clearly an advantage over other approaches, not least because the schedule and the criteria themselves may be the object of introspection by the application, i.e. we have an executable specification which is both convenient and efficient.

Our plans at present include enriching the type of constraints that may be specified to indicate preferences, not only w.r.t./concerning lecturers but also for entire weekly schedules. We may also include input regarding preferences such as the total distance traveled to attend a set of classes, or other criteria we may choose to optimize for. Having an incremental schedule builder is also a useful extension to perform, because the instant feedback may prove a time-saver when setting up timetables.

In what concerns the constraint solver, we are likely to evolve towards high-performance specialized solver engines such as PaCCS or MaCS (Machado, Pedro, and Abreu, 2013; Machado, Abreu, and Diaz, 2013) which will be able to efficiently deal with global optimization, largely because they are designed to work on massively parallel computational platforms.

The system has been in production for about one year, having already evolved to encompass physical resource optimization imperatives, thereby clearly demonstrating its fitness for the appointed task. Considering the good results we have been getting, we will certainly apply a constraint-informed methodology to other Information System design

ISSN

and implementation situations. We are planning to work towards a formalization for general-purpose Constraint-Informed Information Systems, as we gain more experience with concrete use-cases.

References

Abdullah, S., Ali, H. M., and Sipan, I. (2012). *Benchmarking Space Usage in Higher Education Institutes: Attaining Efficient Use*. Journal of Techno-Social, 4 (1), 11-20.

Abdullah, S., Ali, H. M., Sipan, I., Awang, M., Rahman, M. S., Shika, S. A., and Jibril, J. D. (2012). *Classroom Management: Measuring Space Usage*. Procedia - Social and Behavioral Sciences. 65, pp. 931-936. Science Direct.

Abreu, S. (2001). *Isco: A practical language for logic-based construction of heterogeneous information systems*. Proceedings of INAP'01, (pp. 229-237). Tokyo.

Abreu, S., and Diaz, D. (2003). *Objective: in Minimum Context*. In C. Palamidessi, Logic Programming (pp. 128-147). Springer Berlin Heidelberg. doi:10.1007/978-3-540-24599-5_10

Abreu, S., and Nogueira, V. (2006). *Using a Logic Programming Language with Persistence and Contexts*. In M. Umeda, A. Wolf, O. Bartenstein, U. Geske, D. Seipel, and O. Takata, Declarative Programming for Knowledge Management (pp. 38-47). Springer Berlin Heidelberg. doi:10.1007/11963578_4

Ani, A. C., Tawil, N. M., Musa, A. R., Tahir, M. M., and Abdullah, N. A. (2012). *Frequency Index for Learning Space in Higher Education Institutions*. Procedia - Social and Behavioral Sciences (pp. 587-593). Science Direct.

Barták, R. (1999). *Constraint programming: In pursuit of the holy grail*. Proceedings of Week of Doctoral Students (WDS99), (pp. 555-564).

Beyrouthy, C., Burke, E. K., Landa-Silva, J. D., McCollum, B., McMullan, P., and Parkes, A. J. (2006). *Understanding the Role of UFOs Within Space Exploitation*. Proceedings of PATAT, (pp. 359-362).

Beyrouthy, C., Burke, E. K., McCollum, B., McMullam, P., and Parkes, A. J. (2010). *University space planning and space type profiles*. Journal of Scheduling, 13, 363-374. doi:10.1007/s10951-010-0178-9

Booch, G., Rumbaugh, J., and Jacobson, I. (1999). *Unified Modelling Language– Users Guide*. MA: Addison-Wesley Reading.

Burke, E. K., McCollum, B., Meisels, A., Petrovic, S., and Qu, R. (2007). *A graph-based hyper-heuristic for educational timetabling problems*. European Journal of Operational Research, 177-192. doi:10.1016/j.ejor.2005.08.012

Cambazard, H., Hebrard, E., O'Sullivan, B., and Papadopoulos, A. (2012). *Local search and constraint programming for the post enrolment-based course timetabling problem*. Annals of Operations Research , 194, 111-135. doi:10.1007/s10479-010-0737-7

Ceri, S., Gottlob, G., and Tanca, L. (1989). *What you always wanted to know about datalog (and never dared to ask)*. IEEE Transactions Knowledge and Data Engineering, 146-166. doi:10.1109/69.43410

Chemla, D., Diaz, D., Kerlirzin, P., and Manchon, S. (1995). *Using clp (fd) to support air traffic flow management*. Proceedings of the Third International Conference on the Practical Application of Prolog, (pp. 69-83). Paris.

Diaz, D., Abreu, S., and Codognet, P. (2012). *On the implementation of GNU Prolog*. Theory and Practice of Logic Programming, 12, 253-282. doi:10.1017/S1471068411000470

Djebbi, O., Salinesi, C., and Diaz, D. (2007). *Deriving product line requirements: the red-pl guidance approach*. Software Engineering Conference, 2007 (pp. 494-501). Aichi: IEEE. doi:10.1109/ASPEC.2007.63

Garrido, A., Arangu, M., and Onaindia, E. (2009). *A constraint programming formulation for planning: from plan scheduling to plan generation*. Journal of Scheduling, 12, 227-256. doi:10.1007/s10951-008-0083-7

Kilner Planning. (2006). *Space utilization practice, performance and guidelines*. Technical report 38. Obtido em Julho de 2014, de Space Management Group: http://www.smg.ac.uk/documents/utilisation.pdf

Kilner Planning and London Economics. (2006). *Managing space: a review of English further education and HE overseas*. Obtained in July, 2014, at Space Management Group: http://www.smg.ac.uk/documents/FEandoverseas.pdf

Machado, R., Abreu, S., and Diaz, D. (2013). *Parallel performance of declarative programming using a pgas model*. In K. Sagonas, Practical Aspects of Declarative Languages (pp. 244-260). Springer Berlin Heidelberg. doi:10.1007/978-3-642-45284-0_17

Machado, R., Pedro, V., and Abreu, S. (2013). *On the scalability of constraint programming on hierarchical multiprocessor systems*. Proceedings of the 42nd International Conference onParallel Processing (ICPP) (pp. 530-535). Lyon: IEEE. doi:10.1109/ICPP.2013.66

Mazo, R., Salinesi, C., Diaz, D., and Lora-Michielis, A. (2011). *Transforming attribute and clone-enabled feature models into constraint programs over finite domains*. 6th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE) (pp. 188-199). SciTePress.

Mazo, R., Salinesi, C., Diaz, D., Djebbi, O., and Lora-Michiels, A. (2012). *Constraints: The heart of domain and application engineering in the product lines engineering strategy*. International Journal of Information System Modeling and Design, 3, 33-68. doi:10.4018/jismd.2012040102

Müller, T. (2009). *ITC2007 solver description: a hybrid approach*. Annals of Operations Research, 172, 429-446. doi:10.1007/s10479-009-0644-y

ISSN

Post, G., Di Gaspero, L., Kingston, J. H., McCollum, B., and Schaerf, A. (2013). *The Third International Timetabling Competition*. Annals of Operations Research. doi:10.1007/s10479-013-1340-5

Rodrigues, I. P., Matos, N., Abreu, S., Deneckère, R., and Diaz, D. (2013). *Towards constraint-informed information systems*. Proceedings Research Challenges in Information Science (RCIS) (pp. 1-10). Paris: IEEE. doi:10.1109/RCIS.2013.6577690

Rossi, F., van Beek, P., and Walsh, T. (2006). *Handbook of Constraint Programming* (Foundations of Artificial Intelligence). New York: Elsevier Science.

Rudová, H., Müller, T., and Murray, K. (2011). *Complex university course timetabling. Journal of Scheduling*, 14, 187-207. doi:10.1007/s10951-010-0171-3

Tack, G., Lagerkvist, M., and Schulte, C. (2008). *Gecode: An Open Constraint Solving Library*. Workshop on Open-Source Software for Integer and Constraint Programming (OSSICP). Paris.

Van Hentenryck, P. (1989). *Constraint satisfaction in logic programming*. Cambridge: The MIT Press.

ISSN