International Journal on Artificial Intelligence Tools
Vol. 7, No. 4 (1998) 415—442

NEARLY OPTIMUM TIMETABLE CONSTRUCTION THROUGH CLP AND INTELLIGENT SEARCH

PANAGIOTIS STAMATOPOULOS, EFSTRATIOS VIGLAS, and SERAFEIM KARABOYAS
University of Athens, Department of Informatics
Panepistimiopolis, 157 84 Athens, Greece
E-mail: {takis,stratis,mkara}@di.uoa.gr

Received 25 January 1998
Accepted 20 July 1998

The course timetable construction is a procedure that every academic department has to carry out at least twice annually, more times if some of the requirements change. These requirements indicate that a collection of elements must be taken in mind in order for an acceptable solution to be found. They come either from the inherent constraints of the problem or from the involved parties, namely teachers and students. A requirement that is very difficult to satisfy is the one of optimality, which means that the constructed timetable should be the best among the legal ones, according to some quantified quality criteria. In this paper, a method for tackling the course timetable construction problem for academic departments is presented, which is based on Constraint Logic Programming (CLP) for the early pruning of the search space and on the usage of intelligent heuristics in order to guide the search to the generation of nearly optimum solutions. A specific system is presented, named ACTS (Automated Course Timetabling System), which has been implemented in the ECL^{i}PS^e language. This system is currently in use by the Department of Informatics of the University of Athens for the purpose of aiding the semester course timetable construction.

Keywords: scheduling, automated timetabling, constraint logic programming, heuristics

1. Introduction

One of the most common problems faced by man is the problem of *scheduling*. It is a search problem that individuals are asked to tackle almost daily, whether they do it consciously or not. For example, a student preparing for an examination has to organize a study plan. This study plan is nothing more than the scheduling of learning, practicing and revision processes. Generally, every problem that has the objective of corresponding a set of time entities (beginning or ending time, duration) to a set of activities is characterized as a *scheduling problem*.

A broad subclass of the scheduling problems is the one of the *assignment-type problems*.[1] The *timetabling problem*[2,3] is an instance of the assignment-type problem. The objective of the timetabling problem is to place in time a set of events, which might also need the employment of specific resources, in such a way that all required constraints are satisfied. Timetabling problems are faced extensively by educational organizations, e.g. academic departments, schools, etc., either for the

course or the examination scheduling requirements. Although the simple version of the timetabling problem may be solved in polynomial time, it has been proved that when someone has to deal with preassignments and unavailabilities, which is the usual non-trivial case, the problem becomes NP-complete.[4,5] This property is the one that makes the problem really hard.

The work on automatic construction of timetables has started in the early sixties[6] and a lot of progress has been done since then. Various approaches have been followed, such as graph coloring algorithms,[7] where each vertex in a graph represents a triplet (subject, teacher, room) and two vertices are connected if they cannot be scheduled simultaneously, that is, if they have a subject, teacher or room in common. The problem is reduced to find a coloring of the graph with a number of colors less than a given number, if one exists. Various other Operations Research (OR) approaches have been used as well,[8] such as mathematical programming algorithms,[9,10] where the central idea is to solve an optimization problem whose purpose is to reduce constraint violations. Quite recently, a lot of Artificial Intelligence (AI) techniques have been employed, such as simulated annealing,[11,12] tabu search[13,14] and genetic algorithms[15,16,17,18] or, in general, evolutionary approaches.[19] Finally, another AI approach, the constraint programming idea, has been used extensively for tackling the timetabling problem,[20,21,22] most of the times following a *Constraint Logic Programming (CLP)* methodology.[23,24,25,26]

The procedure of solving a timetabling problem in a constraint programming environment is usually carried out in two stages. The first stage has to do with the discovery and the statement of the involved constraints. These constraints are exported from the concerned parties and have to be imposed over a set of variables that model the problem entities. The second stage has to do with the assignment of values, and more specifically the proper values, to the above mentioned variables. The successful passage through both of the above stages results in the construction of a timetable that can be characterized as admissible by all concerned parties.

It is often argued that a timetabling problem either has a very large number of solutions or it has no solution. In the first case, the best, or a very good, solution has to be recognized among all feasible solutions. On the other hand, if the problem is stated in such a way that no feasible solution exists, this has to be identified, possibly through an exhaustive search of the problem space. In order to get an idea about the size of the search space of a real-life timetabling problem, consider a semester at a university where 40 different subjects have to be scheduled, each subject splitting into 4 teaching periods per week. Let us also assume that there are 5 working days in a week with 9 possible teaching periods in every day and that there are 4 rooms for hosting lectures. In this case, which is quite typical, the search space to be explored for the construction of a weekly timetable is as big as $(5 \times 9 \times 4)^{4 \times 40} \simeq 7 \times 10^{360}$.

The construction of a timetable is a process ever evolving and executed. This aspect puts forth another requirement of the timetabling problem, which refers to the quite often arising need for *rescheduling*. It is not uncommon that a change in the input data so slight has occurred, that it is not needed to create a new timetable from scratch. A few minor alterations are enough for an acceptable timetable to be the case again. In these situations, the previously mentioned automated timetabling procedure should be in a position to perform these alterations, thus generating a timetable that has the smallest possible number of differences to the original timetable.

A logical solution to the timetabling problem would be to find a way to directly express and impose the constraints that model the requirements of a specific instance of it. A mechanism capable of solving and satisfying these constraints has to be employed. The framework for the development of such procedures is offered by CLP. The purpose of this paper is to exhibit the adequacy of CLP, enriched with intelligent search techniques, in solving the course timetabling problems of academic departments. In addition, a working example of the method discussed is presented, as this has been developed in an instance of CLP, namely the ECL^{i}PS^e language, and is being used in the Department of Informatics of the University of Athens (DI/UoA).

In what follows, firstly a specific instance of the timetabling problem is introduced. A brief description of CLP and the ECL^{i}PS^e language comes next. Then, the representation of the problem in the above mentioned real system is put forth, accompanied with a discussion on the developed intelligent search techniques. Finally, the outcome of the system is considered, an evaluation of it is presented and various possible enhancements are outlined.

2. Overview of the DI/UoA Timetabling Problem

Before reaching the position where a solution for a specific timetabling problem is required, a few other prerequisites have first to be obtained. These prerequisites are nothing more than the gathering of the data, over which the constraints will be imposed and the solution generating process will take effect.

In what follows, a description of the data and why these are critical is given. The discussion concentrates on the university case, rather than the school one. More precisely, one aspect of the university timetabling problem is considered, which is known as *course scheduling*. The other aspect is characterized as *examination scheduling*. Generally, the course scheduling problem is far more difficult to solve, as it contains various elements not found in examination scheduling. In a nutshell, the basic differences concentrate on the fact that there can be various lectures of a course in course scheduling, while only one instance of a course in examination scheduling. In addition, some conflicts can be allowed to exist in a course schedule, while this is not the case in examination scheduling. Another factor to be considered is that a course schedule generally spreads over a limited period of time, usually a week, while an examination schedule, in the general case, can be constructed for an

arbitrary period, long enough for all the subjects to be scheduled without conflicts. This paper concentrates on weekly course scheduling. Finally, it has to be noted that although the description is tailored to the needs of DI/UoA, few modifications are needed to adapt the model to the cases of other academic departments.

2.1. *The problem data and constraints*

The first element that has to be considered in a discussion about the data involved in course scheduling is the courses themselves. For each course, a number of matters have to be taken in mind. Each course may or may not be split into lectures. The number of overall hours in one week must be equal to the total duration of the subject[1] in question. This splitting up of a course into weekly lectures must be unique for this course, but not universal for all the courses of the same duration. A substantial amount of flexibility is desired in this sector.

Other properties of a given subject come from the persons involved in its teaching, namely the teacher and the students. A number of things must be known about the faculty member in charge of teaching a given course. First of all, it must be known when he or she is available, then, which other subjects he or she teaches. If there are more than one teacher for a subject, the overall availability should be set to the intersection of the availabilities of all the involved faculty members. The student body can have constraints also. Availability is again one of them. A whole semester might be unavailable to attend the lectures of a course if, for some reason, it is otherwise occupied (for example, in a laboratory).

Returning to the courses' properties, a very important factor is that of the attendance for a given course. This is in close relation to spatial factors stemming from the available classrooms. A given classroom can hold up to a certain amount of students. Surpassing this limit is unacceptable. Classrooms, on the other hand, can themselves have availability requirements. A classroom can be unavailable for a given period in the week, for example for a faculty meeting or for seminars.

One final element revolving around the course properties is the type of the course. In some academic departments, each subject is given a type, according to the nature of the material it covers, the number of prerequisites it has and the number of subjects for which it is a prerequisite. This is also the case in DI/UoA. The type of a course is the element by which conflicts can be resolved, i.e. avoided or allowed. For example, courses of the same type cannot be scheduled on the same time. On the other hand, lectures for courses of sufficiently different types can be allowed to take place concurrently.

In order to get a more precise idea about what is given and what is needed to be computed in the specific timetabling problem which is being tackled, consider the following.

Assume that there are N subjects, M teachers and L classrooms. Consider, also, the set $Days$ of the working days in a week and the set $Periods$ of the (hourly)

[1] In the following, the words "course" and "subject" will be used to denote the same timetabling entity.

teaching periods in every day. For example, it might be the case that $Days = \{\text{mon}, \text{tue}, \ldots, \text{fri}\}$ and $Periods = \{9{:}00\text{-}10{:}00, 10{:}00\text{-}11{:}00, \ldots, 19{:}00\text{-}20{:}00\}$.

For the i-th subject ($1 \leq i \leq N$) there exists a number of given parameters:

v_i: approximate number of students following the i-th subject
l_i: number of lectures of the i-th subject
p_{ij}: number of sessions (teaching periods) of the j-th lecture of the i-th subject ($1 \leq j \leq l_i$)
t_i: number of teachers of the i-th subject
n_{ij}: j-th teacher of the i-th subject ($1 \leq j \leq t_i,\ 1 \leq n_{ij} \leq M$)
d_i: number of disjunctive (not conflicting) subjects with the i-th subject
s_{ij}: j-th disjunctive subject with the i-th subject ($1 \leq j \leq d_i,\ 1 \leq s_{ij} \leq N,\ s_{ij} \neq i$)
us_i: number of unavailability slots (u-slots) of the i-th subject
dus_{ij}: day of the j-th u-slot of the i-th subject ($1 \leq j \leq us_i,\ dus_{ij} \in Days$)
pus_{ij}: teaching period of the j-th u-slot of the i-th subject ($1 \leq j \leq us_i,\ pus_{ij} \in Periods$)

In addition, for the i-th teacher ($1 \leq i \leq M$) the existing parameters are:

ut_i: number of u-slots of the i-th teacher
dut_{ij}: day of the j-th u-slot of the i-th teacher ($1 \leq j \leq ut_i,\ dut_{ij} \in Days$)
put_{ij}: teaching period of the j-th u-slot of the i-th teacher ($1 \leq j \leq ut_i,\ put_{ij} \in Periods$)

Finally, the given parameters of the i-th classroom ($1 \leq i \leq L$) are:

c_i: capacity of the i-th classroom
uc_i: number of u-slots of the i-th classroom
duc_{ij}: day of the j-th u-slot of the i-th classroom ($1 \leq j \leq uc_i,\ duc_{ij} \in Days$)
puc_{ij}: teaching period of the j-th u-slot of the i-th classroom ($1 \leq j \leq uc_i,\ puc_{ij} \in Periods$)

What has to be computed is the time and place that every session of the lectures of each subject has to be scheduled in, that is

x_{ijk}: day of the k-th session of the j-th lecture of the i-th subject
y_{ijk}: teaching period of the k-th session of the j-th lecture of the i-th subject
z_{ijk}: classroom of the k-th session of the j-th lecture of the i-th subject

where $1 \leq i \leq N,\ 1 \leq j \leq l_i,\ 1 \leq k \leq p_{ij},\ x_{ijk} \in Days,\ y_{ijk} \in Periods$ and $1 \leq z_{ijk} \leq L$.

The constraints that the solution has to satisfy are the following:

(i) No two sessions can take place at the same time in the same classroom:

$$x_{ijk} \neq x_{i'j'k'} \quad \vee \quad y_{ijk} \neq y_{i'j'k'} \quad \vee \quad z_{ijk} \neq z_{i'j'k'}$$

$\forall\ i, j, k, i', j', k'$ with $1 \leq i \leq N,\ 1 \leq j \leq l_i,\ 1 \leq k \leq p_{ij},\ 1 \leq i' \leq N,\ 1 \leq j' \leq l_{i'},\ 1 \leq k' \leq p_{i'j'}$ and $i \neq i'$ or $j \neq j'$ or $k \neq k'$.

(ii) The sessions of a lecture have to take place in consecutive teaching periods of the same day in the same classroom:

$$x_{ijk} = x_{ij(k-1)}$$

$$y_{ijk} = next(y_{ij(k-1)})$$

$$z_{ijk} = z_{ij(k-1)}$$

$\forall \;\; i,j,k \;\;$ with $\;\; 1 \leq i \leq N, \;\; 1 \leq j \leq l_i \;\;$ and $\;\; 2 \leq k \leq p_{ij}$, where $next(tp)$ is the teaching period immediately following teaching period tp, e.g. $next(10{:}00\text{-}11{:}00) = 11{:}00\text{-}12{:}00$.

(iii) The lectures of a subject have to take place on different days:

$$x_{ijk} \neq x_{ij'k'}$$

$\forall \;\; i,j,j',k,k' \;\;$ with $\;\; 1 \leq i \leq N, \;\; 1 \leq j \leq l_i, \;\; 1 \leq j' \leq l_i, \;\; j \neq j', \;\; 1 \leq k \leq p_{ij}$ and $\;\; 1 \leq k' \leq p_{ij'}$.

(iv) The capacities of the classrooms have to be respected:

$$v_i \leq c_{z_{ijk}}$$

$\forall \;\; i,j,k \;\;$ with $\;\; 1 \leq i \leq N, \;\; 1 \leq j \leq l_i \;\;$ and $\;\; 1 \leq k \leq p_{ij}$.

(v) No teacher can teach two subjects simultaneously:

$$x_{ijk} \neq x_{i'j'k'} \quad \vee \quad y_{ijk} \neq y_{i'j'k'}$$

$\forall \;\; i,j,k,i',j',k' \;\;$ with $\;\; 1 \leq i \leq N, \;\; 1 \leq j \leq l_i, \;\; 1 \leq k \leq p_{ij}, \;\; 1 \leq i' \leq N,$ $1 \leq j' \leq l_{i'}, \;\; 1 \leq k' \leq p_{i'j'}, \;\; i \neq i' \;\;$ and $\;\; \{n_{ij''} \mid 1 \leq j'' \leq t_i\} \cap \{n_{i'j'''} \mid 1 \leq j''' \leq t_{i'}\} \neq \emptyset$.

(vi) The unavailability requirements of the teachers have to be respected:

$$dut_{n_{ijj'}} \neq x_{ij''k} \quad \vee \quad put_{n_{ijj'}} \neq y_{ij''k}$$

$\forall \;\; i,j,j',j'',k \;\;$ with $\;\; 1 \leq i \leq N, \;\; 1 \leq j \leq t_i, \;\; 1 \leq j' \leq ut_{n_{ij}}, \;\; 1 \leq j'' \leq l_i \;\;$ and $1 \leq k \leq p_{ij''}$.

(vii) The unavailability requirements for the classrooms have to be respected:

$$duc_{z_{ijkj'}} \neq x_{ijk} \quad \vee \quad puc_{z_{ijkj'}} \neq y_{ijk}$$

$\forall \;\; i,j,k,j' \;\;$ with $\;\; 1 \leq i \leq N, \;\; 1 \leq j \leq l_i, \;\; 1 \leq k \leq p_{ij} \;\;$ and $\;\; 1 \leq j' \leq uc_{z_{ijk}}$.

(viii) The unavailability requirements for the subjects have to be respected:

$$dus_{ij} \neq x_{ij'k} \quad \vee \quad pus_{ij} \neq y_{ij'k}$$

$\forall \;\; i,j,j',k \;\;$ with $\;\; 1 \leq i \leq N, \;\; 1 \leq j \leq us_i, \;\; 1 \leq j' \leq l_i \;\;$ and $\;\; 1 \leq k \leq p_{ij'}$.

(ix) The non-conflict requirements among subjects have to be respected:

$$x_{ijk} \neq x_{s_{ij'}j''k'} \quad \vee \quad y_{ijk} \neq y_{s_{ij'}j''k'}$$

$\forall \;\; i,j,k,j',j'',k' \;\;$ with $\;\; 1 \leq i \leq N, \;\; 1 \leq j \leq l_i, \;\; 1 \leq k \leq p_{ij}, \;\; 1 \leq j' \leq d_i,$ $1 \leq j'' \leq l_{s_{ij'}} \;\;$ and $\;\; 1 \leq k' \leq p_{s_{ij'}j''}$.

Note that the previous formulation does not encode the types of the courses. It is assumed that this information has been preprocessed and the result is represented by the pairs of disjunctive subjects.

2.2. *Quality standards*

A correct timetable is one that satisfies all the constraints imposed over its data, which were outlined in the preceding paragraph. As it has been stated, this is not enough for a timetable to be complete. For a rough idea to be given, a correct timetable can be one that manages to somehow "compress" all the lectures in three days in a five-day weekly basis. But a timetable of this nature is unacceptable.

Although quality, as a concept, is purely subjective, a number of criteria exist, the meeting of which leads to an objectively qualitative and, thus, acceptable timetable.

The first of these criteria has to do with the lecture balancing for all semesters. For example, if a semester has a total of thirty teaching periods weekly, then care has to be taken, so that the daily teaching periods for this particular semester are as close as possible to the median, i.e. six. Moreover, the lectures of all the subjects in one day have to be as concentrated as possible, so no idle time between lectures of different courses exists. Teaching load criteria also exist for faculty members. For example, the number of lectures a faculty member gives daily should not exceed a given limit.

Another criterion originates from the duration of a subject. A subject with a total of five teaching periods is best to be divided in two or three lectures. If that is the case, then these lectures should have the greatest possible daily distance between them.

The fourth standard has to do with the minimization of students' movement between lectures of different subjects. That is, a particular semester should spend as much time as possible in the same classroom, for a given day. The capacity of classrooms creates another standard, which has to do with their utilization. A classroom should host courses for which their attendance is most closely to its capacity. In this way, maximum utilization is achieved.

The final two criteria have to do with the preferences of those who are affected by the timetable. These preferences usually come from faculty members and the student body. For example, although a faculty member may be available for teaching during a specific period, nevertheless prefers for some reason not to teach. On the other hand, courses of different types, that under normal circumstances might be scheduled simultaneously, are preferred by the student body not to be, perhaps because there is a great number of students who wish to attend both subjects.

At this point, it must be stressed out that all criteria mentioned are preferences and not constraints. This means that they can be relaxed or even disregarded in order for a timetable of greater quality to be constructed. However, some of the quality standards are obviously opposed. For example, better utilization might mean that students should be moved between different classrooms. It is probably

impossible for a timetable to be constructed in some way that all the criteria are respected. This brings us back to the subjective nature of quality.

2.3. *The demands from a timetabling application*

A number of matters have to be taken in mind in order for a successful timetabling application to be constructed.

First of all, the data representation should follow the nature of real-world data as close as possible. There should be sufficient representation of courses, faculty members and classrooms as well as their individual properties. These data should be able to change on the user's demand to reflect the changes of real data. Next, the constraints concerning these data should be imposed over the problem representation. This leads to a way of producing correct timetables. The following step has to do with the generation of a nearly optimum solution to the problem. This involves the satisfaction of as many as possible quality criteria. But, as any concerned user gives different significance to the quality standards, a way to assign priorities to these standards should be provided. The higher the priority the more effort for satisfaction should be made by the application.

Rescheduling is another aspect. It should be created in such a way that a given timetable could be regarded as an additional input. Then, it should attempt to reschedule this timetable, making the smallest possible number of alterations, if any.

An automated application cannot come close to the complexity, or maybe simplicity, with which a human tackles a complex problem like the timetabling one. This, in conjunction with the fact that the final solution might indeed be in need of slight alterations, makes necessary the ability of post-editing a generated timetable.

All of the above imply the construction of a complete and user friendly interface, through which the user will be in a position to input, save, retrieve and generally manipulate the problem's data and results.

Last, but certainly not least, comes the matter of execution time. This time can be regarded as the time period needed for the application to generate a timetable for real data, in volume as well as intricacy and values, starting from a zero basis. If a person needs an afternoon or a day or, even, a week to create a timetable, then an application that does not reduce this time considerably is not of much use.

An application that is built in a way to respect all the factors mentioned above is in a good path. But for an application of that kind to be developed, a consistent framework capable of accepting and resolving all the previously stated representations and problems should exist. This framework is CLP. A successful application using a specific instance of CLP has been developed. A conversation on that particular instance, namely the ECL^{i}PS^e language, as well as CLP, follows in the next section.

3. Constraint Logic Programming and ECL^{i}PS^e

Logic programming is a problem solving methodology, as opposed to traditional, procedural programming. The basic idea behind logic programming is that it is not so much needed to materialize the procedures that lead towards the solution of a problem, as it is needed to identify the entities involved and rather declare the relations between them. The solution is then found by means of an inference mechanism acting upon the previously mentioned relations. This is the notion of declarative programming and the best known representative of this class of computer languages is Prolog.

It has been argued in the literature that an algorithm has two main components, namely *logic* and *control*.[27] Logic is in charge of what the algorithm does, while control is in charge of how this is done. An ideal programming methodology should first be concerned with logic (what we want to compute) and then with control (how the solution is achieved). The superiority of logic programming stems from the fact that it provides a means of dividing the two concepts mentioned previously.

The logic programming approach is extremely applicable to solving search problems that involve entities which are related through specific constraints. An internal resolution and inference mechanism is used to find the required solution. Most commonly, the mechanism used is known as *SLD-resolution*. This leads to a *depth-first search* of the problem space, resulting in what is known as a *generate-and-test* strategy. Constraints can act as tests, eliminating from the search space paths that are dead-ends. However, this may not lead to acceptable efficiency for large problems, so what someone might want to have is a more active exploitation of constraints.

The purpose of CLP[28,29] is now clear. It is an attempt to enrich logic programming in such a way that all of its unique properties are maintained, but a new mechanism, concerned with the imposition and solution of constraints is introduced. This mechanism's objective is to enhance logic programming in such a way that efficiency is the main target. This kind of philosophy is known as $CLP(X)$ scheme, where X can be instantiated to any possible set of entities, in accordance to the application under question. For example, if a program manipulates real numbers, $CLP(\Re)$ is used.

The required enhancements for CLP come from work that is done in the areas of consistency and constraint propagation techniques. The primary objective of these methods is to avoid the generate-and-test paradigm and prune the search space by solving the imposed constraints. In this way, less effort is needed to find the solution, because the system itself rejects possible paths in the search space, by deciding that they do not lead to a solution, sometimes even prior to initializing the search procedure. This realization of search is called *constrain-and-generate*. With the introduction of CLP, the solution of a large class of problems has become feasible. This class of problems contains the scheduling problem and, thus, the timetabling problem as well.

ECL^{i}PS^e [30] is an instance of CLP. It has been created at the European Computer-Industry Research Center (ECRC) as the successor of CHIP, the first $CLP(FD)$

solver. *FD* stands for *Finite Domains*. The constraint facilities of CHIP have been integrated into ECL^{i}PS^e.

ECL^{i}PS^e is a Prolog system, enhanced with various extensions that provide sufficient flexibility for the undertaking of a number of diverse problems. This flexibility stems from a particular data type supported by the language, named *metaterm*. The metaterm facility provides a basis for a number of libraries to be developed. Examples are the constraint libraries and their specific instances, such as the finite domains library. The finite domains library has been used in the context of the work related to this paper, so a brief discussion aiming at the better understanding of its functionality follows.

The finite domains library of ECL^{i}PS^e contains a number of constraints to be imposed, mainly over integer, but also any other instance of atomic data. The basic concept to be understood is that of a *domain variable*. The domain variable is one that ranges over a finite domain. This range is defined through the built-in `::/2` predicate. For instance, the series of goals `X :: 1..5, Y :: [red, green, blue], Z :: [1, 3, 5..8]` defines the domain variables `X`, `Y` and `Z` ranging over the domains `{1, 2, 3, 4, 5}`, `{red, green, blue}` and `{1, 3, 5, 6, 7, 8}` respectively. Constraints can be one of two kinds, *arithmetic* and *symbolic*. An arithmetic constraint is nothing more than a relation between linear terms. This relation can be one of equality (`#=/2`) or inequality (`##/2`, `#</2`, `#<=/2`, `#>/2`, `#>=/2`). A linear term is an arithmetic expression composed of numbers and domain variables, in such a way that no multiplication or division between domain variables exists. For example `2*X-Z #< X-3` is an arithmetic constraint, while `X*Z #> 3` is not, as it does not respect linearity. Conjunction and disjunction of constraints is provided through the use of the `#/\` and `#\/` predicates. Symbolic constraints are implemented through specific predicates such as `element(I, List, X)`, stating that `X` should be the `I`-th element of `List`. Both `X` and `I` are domain variables, while `List` is a list consisting of ground terms. Additional predicates are provided by ECL^{i}PS^e in order to supply enumeration (`indomain/1`) and optimization (`min_max/2`, `min_max/5`) facilities.

For a CLP language such as ECL^{i}PS^e, there exists a specific strategy under which constraint satisfaction problems should be tackled. This tactic consists of three basic steps:

(i) Identify the domain variables and define them and their domains. These variables serve as the representation of the problem entities.

(ii) Impose over these variables the constraints that state the relations between them. These constraints are in fact used for the pruning of the search space and, thus, act as the limits of the territory inside which lay the solutions of the problem under consideration.

(iii) Start an enumeration procedure, which, in cooperation with the internal constraint propagation mechanism as well as the Prolog engine, returns the problem solutions, if any. This step can also be found under the CLP nomenclature as *labeling*. In case the optimum solution is required, additional care has to be taken. ECL^{i}PS^e provides a *branch-and-bound* method that computes the opti-

mum solution with respect to a given cost function. Another approach would be to use some intelligent search technique based on the problem heuristics, in order to return a nearly optimum solution. This is the case of the work presented in this paper and the topic under the following discussion.

4. ACTS – Automated Course Timetabling System

An attempt to automate the timetable construction procedure has been made at DI/UoA. The outcome of this attempt is a system called ACTS (Automated Course Timetabling System). In this section, the various elements of this attempt will be highlighted, paying particular attention to the intelligent search techniques developed, in order to construct a real-life working system.

4.1. *The system data*

For a timetabling application to be built under CLP, a number of elements have to be considered. The first of these are the data of the problem and the constraints, obvious or implied, which have to be imposed over them.

The case that had to be considered in ACTS was the one concerning course scheduling. Consequently, a representation had to be decided that would fully reflect upon the way these data are perceived by an individual that has been placed in charge of undertaking the timetable construction procedure.

- The first group of elements that will be presented is that of real-world time concepts, namely days and teaching periods. In DI/UoA, timetables are constructed on a weekly basis, therefore resulting in an encoding of days as numbers, more precisely, 1 for Monday, 2 for Tuesday and so on. The number of teaching periods per day are eleven, starting at nine in the morning and ending at eight in the evening. The encoding is quite similar in this case, resulting in one number for every teaching period, for instance 1 for the first period, from 9:00 to 10:00, 2 for the teaching period starting at 10:00 and ending at 11:00, etc. This encoding is quite important, as it is used in several of the representations of the other structures identified by the program.

- The second element considered is that of teaching classrooms. These classrooms have a number of properties. The first of these properties is their name, along with a unique number assigned to them by the program itself. The next of their properties is their capacity, which translates in the number of students they can accommodate and hence, the subjects they can host, as an expected attendance is assigned to every subject. Their last property and probably the most important one, as it leads to hard constraint imposition, is the one concerned with their availability. A classroom might be unavailable for some days and teaching periods due to external factors, such as faculty meetings or seminars. In this case, these days and teaching periods should be excluded. A list is created in order to denote the days and periods that a classroom is

unavailable. Knowing when a classroom in *not* available is selected, because it is easier to impose the constraints. A Prolog fact, `classroom/4`, is used each time a classroom has to be encoded.

- The next timetable element to be considered is that of the faculty members. Their first property is again their name, along with a unique number assigned to them by the program. The other two of their properties have to do with their unavailability. A faculty member can be unavailable, in which case the specific days and teaching periods that he or she declares as unavailable are excluded, or can be available, but for some reason prefers not to have any teaching obligations whatsoever. The difference between the above mentioned unavailability classes is that while the first one will be satisfied, whatever the cost, the second will be sacrificed if the generation of a better timetable instructs so. The latter has to do with the quality of the generated solution. A Prolog fact, `teacher/4`, is used to denote faculty members.

- The other real-life entity that should be represented is that of the student body and more specifically the semesters involved in and affected by the timetable process. In DI/UoA, there exist four undergraduate and six postgraduate academic years, the latter originating from three postgraduate curricula, two years in each curriculum. This means that a total of ten entries have to be expressed. For each entry, its coded name is needed, along with a list instructing when the semester under question is not available. This can be the case, because some of the above mentioned postgraduate academic years belong to joint programs with other departments, which means that a fraction of their timetable can be fixed and thus unchangeable throughout the timetable process. For example, a postgraduate semester can give the requirement that it can only attend lectures on two or three specific days of the week. This has to be integrated into the knowledge of the program. The Prolog fact `semester/2` is used for semester information entries.

- The last of the timetable elements that is needed is the one concerning the taught courses in one semester. A number of properties have to be identified for one course. Again, the first group of these has to do with its unique identification, consisting of a number assigned to the course by the program, as well as the course's name and code, as they appear in the department's curriculum. The next property is the one concerning the expected audience to the lectures of the course. For instance, this can be the median of students attending the course in the last years. The following property has to do with the weekly duration of a subject and how this duration should be split into a number of daily lectures. In DI/UoA, the total duration of a subject varies from three to five weekly hours. This duration can be split in a number of ways. For example a five-hour course can be split either in two two-hour lectures and a single one-hour one, or it can be split into one two-hour and

one three-hour lecture. In any case, the program is indeed concerned with the way the duration is split into lectures and not how long this duration is.

An interesting property of a subject is the one concerned with what is known as the subject's type. It is the case in DI/UoA, as well as other academic departments, that a characterization is given to each subject, according to the material it covers, the semester it is offered and the internal structure of the department. For example, a subject of DI/UoA's undergraduate curriculum can be compulsory, basic or optional. In the two latter cases, it belongs to one of the department's three divisions. Moreover, it might be the case, that it is considered as basic or optional for more than one division and even belong to more than one semester, furthermore, appearing in both undergraduate and postgraduate curricula. It is clear that a very complex situation has to be acknowledged and integrated into the program. For these difficulties to be overcome, a demonstrable amount of flexibility was chosen. In fact, a subject can appear under almost every category that was previously mentioned. The program itself, is in such a position that, if asked, can clear the situation, deciding on the strongest of the categories, even finding with which other courses a particular course should not be scheduled simultaneously. A list is generated consisting of subject codes, denoting that there should exist no conflicts between the subject under consideration and the members of the list. A similar list denotes a preference of avoiding conflicts between particular courses. Both of the above mentioned lists can be universally created by means of default rules integrated into the program.

A consequence of the subject's belonging to one or more curricula is the fact that unavailability factors have to be taken into consideration. This unavailability of a subject is nothing more than the union of the unavailabilities of all the semesters and academic years to which the subject belongs.

A subject is, of course, taught by some faculty member. This can be one, or more than one. Either case, it has to be reflected upon the subject's properties. The final property is a flag. It denotes whether the subject will be scheduled or not. It may be the case, under special circumstances, that a subject, although it appears in the department's curriculum, should not be scheduled.

A subject is denoted by the Prolog fact `subject/11`.

To sum up all the information given above, the following are examples of every data entry to the program.

- ```
  classroom(1, 'Informatics Room', 120,
            [na(1,9), na(1,10), na(1,11)])
  ```

  A classroom with a code of 1, named Informatics Room, a capacity of 120 students, which is not available on Monday for the last three teaching periods.
  ```

- `teacher(1, 'Stamatopoulos', [na(1,9), na(1,10), na(1,11)], [np(2,1), np(2,2)])`

  A faculty member with a code of 1, named Stamatopoulos, who is not available the last three teaching periods of Monday and who prefers not to teach the first two periods of Tuesday.

- `semester('u/3', [na(2,1), na(2,2)])`

  The third undergraduate semester, which is not available on the first two teaching periods of Tuesday.

- `subject(1, 'CS10', 'Expert Systems', 80, 2+2, [['u/2', 'b/2'], ['u/3, 'b/2']], [1], [5,6,7], [], [na(3,1), na(3,2)], 1)`

  A subject with a code of 1, the curriculum code of CS10, named Expert Systems, an expected audience of 80 students, a total duration of four hours split in two two-hour lectures, characterized as basic of the second division for the second and third academic years, taught be the teacher with a code of 1, not to be scheduled on the same time with subject under codes 5, 6 and 7, no preference for avoidance of simultaneous scheduling, that should not be scheduled on the first two teaching periods of Wednesday and which should appear in the timetable under construction.

4.2. *The problem modeling*

As it has been stated in a previous paragraph, the first step in tackling a constraint satisfaction problem under a finite domains implementation is to identify the domain variables involved and define their domains.

In this specific problem, the problem variables constitute an hourly lecture, or a session, of a given subject. This modeling was favored, because this is what is presented in a timetable. When someone reads a given timetable, he does not so much see lectures of subjects, as much as he sees sessions of a subject, the combination and sequence of which create lectures. The creation of lectures out of sessions is a matter of correct constraint imposition and it will be discussed shortly. In addition, this representation gives additional flexibility to the user, in order to evade some of the constraints imposed by the program, in the process of post-editing. However, this is not recommended, as such an action may create potential problems in a later executed rescheduling process. Another advantage of the selected decision stems from the fact that a representation of this kind, although more expensive in terms of memory than a representation that would concentrate on starting hours of the lectures (the ending hour can be derived from the duration of the lecture), makes constraint imposition easier. For instance, a required constraint is that all the sessions are assigned a different triplet of day, teaching period and classroom. This kind of constraint is provided by ECL^{i}PS^e. In any other case, programming would have to be lowered to an inferior level, the level of designing and developing new constraints.

The number of the variables used is in accordance to the number of courses appearing in the timetable under construction and their duration. Example granted, if there are five subjects, each with a three-hour duration, fifteen domain variables have to be created.

Now that the variables are identified, their domains have to be defined. These domains are in close relation to the entity that the domain variables represent. It was mentioned before that every variable is assigned a triplet consisting of a day, a classroom and a teaching period. This is in fact the representation used, but not in this original form. The language has the ability to handle constraints imposed over integer and atomic domains. A triplet of this kind is neither integer nor atomic. So, this information had to be somehow compressed in a different form, in order to use the high-end constraint mechanism of ECL^{i}PS^e. For that reason, every triplet, that is, of course, unique for a specific instance of the problem, was unified into an integer number. This approach seems to be slightly different from the mathematical model presented earlier, where for each session of every lecture of all subjects three distinct variables are defined, however, it is easy to see that this, rather syntactic, change will serve a more efficient encoding of the problem constraints. Indeed, constraints that should be imposed over such a triplet were easily programmed under this representation. For instance, a constraint of this kind would be that every session should be held on a different combination of day, classroom and teaching period. This could be done with only one call, by use of the provided `alldifferent/1` predicate. On the other hand, constraints that had to be imposed over only one element of such a triplet could be programmed if the triplet would be broken down in its constituents. A linear term can be assembled that can analyze any number into a combination of days, classrooms and teaching periods. This linear relation is

$$N = (D-1) \times Rooms \times Periods + (R-1) \times Periods + P$$

where N is the triplet number, D is the day number, R is the classroom number as it has been assigned to the classroom by the program itself, P is the teaching period number, $Rooms$ is the total number of classrooms and $Periods$ is the overall number of teaching periods in one day.

For instance, if there were five days, three classrooms and seven teaching periods, then the converted numbers would be as shown in Table 1.

Table 1. Correspondence of (D, R, P) triplets to triplet numbers.

	1st classroom	**2nd classroom**	**3rd classroom**
1st day	1–7	8–14	15–21
2nd day	22–28	29–35	36–42
3rd day	43–49	50–56	57–63
4th day	64–70	71–77	78–84
5th day	85–91	92–98	99–105

It becomes obvious from the previous example that the domain for every program

variable ranges from 1 up to the product of days, classrooms and teaching periods (in the case above, $5 \times 3 \times 7 = 105$). This is the original domain of every variable, since after the imposition of the availability constraints a number of elements will be excluded. More specifically, the first values to be excluded have to do with the fact that some classrooms might not be available for individual combinations of days and teaching periods. Other constraints are special for every course to be scheduled (for instance, faculty member unavailability).

4.3. *Imposed constraints*

The next step in dealing with a constraint satisfaction problem is the imposition of constraints over the identified and defined variables. These constraints come from the nature of the problem faced. In the timetable construction case, the constraints originate from the various discrete structures that constitute the data of the problem.

- The first of these constraints is derived from the teaching facilities of an academic department, namely the classrooms. A classroom can be unavailable for specific combinations of days and teaching periods. If that is the case, these values will have to be cleared off the domains of the problem variables. In fact, in the actual implementation, these values do not even make it to the domains. After computing all the possible values for a domain variable, these values are inserted into a list. All the combinations of unavailable days and teaching periods, for every classroom, are deleted from the contents of this list. This list is later used to define the variables' domains. If all the variables were to be initialized to the same domain and later, by means of constraint imposition, have some of the possible values excluded, it would result in unnecessary aggravation of the whole system. These values are universal and are known *a priori*, so there is no need for them to be inserted into the domains.

- The second type of constraints stems from the faculty members. As long as there exists teacher unavailability, then the appropriate constraints have to be imposed. A course, whose teacher denotes a specific day and teaching period as unavailable, cannot have any session take place at that time. In the event where more than one teacher exist for a subject, then the union of their unavailabilities is needed. That is because although not both faculty members are together in the classroom, it is common practice that they should both be available in case a swapping of sessions has to occur for some reason. Given the fact that a faculty member characterizes only days and teaching periods as unavailable, it is obvious that not only one value will be excluded from the possible solution set, but a number of values equal to the number of available classrooms for the given combination. For instance, if there are five classrooms available for a session, for which the faculty member in charge of teaching denotes a combination of day and teaching period as unavailable, then five possible values will be cleared.

The next constraint that faculty members introduce arises from their availability to teach a subject. This is because when a faculty member has a teaching session, he or she is unavailable for any other session. The former is clearly an implied constraint and one that can be missed if the problem is tackled with inappropriate diligence. In this case, the possible value of this session has to be excluded from the possible values of other sessions held by the same faculty member. Propagation techniques are used in analogous circumstances in order to clear the situation. These constraints are usually solved only after a few values have been assigned to domain variables.

- Courses are the source of another group of constraints. Given the fact that a course has a specific type, this implies that it should not be scheduled on the same time with some other courses. This translates to the constraint that the domain variables of these subjects should be different in the aspect of the combination of day and teaching period. It was stated in a previous paragraph that the list of all the courses with which no conflict is allowed for a given subject is known to the program and in any case can be created by the program, according to the subject's type. These constraints are imposed only over one of the subjects involved due to the reflexive nature of the difference relation.

  The second constraint has to do with the splitting of a subject into lectures. Special constraints have to be imposed in order for the subject to be correctly split (each lecture on a different day) as well as for the continuity of the sessions for a given lecture to be assured (each session follows the other on the same day and classroom).

  The last constraint comes from the expected attendance of the subjects. The lectures of every subject have to take place in classrooms that are large enough to host them. All the values corresponding to classrooms of a smaller capacity than the expected audience are cleared off the variables' domains.

- The last type of constraints is derived from the student body and has to do with the individual semesters' unavailability. A subject belonging to a semester that has declared some days and teaching periods unavailable should not have any of its lectures held on these specific combinations.

After the imposition of constraints, constraint solving techniques are initialized, in order to prune the search space. In special cases, of hard-constrained variables, this means that their domains have dramatically decreased, or even have been initialized to some value. This tactic reduces substantially the workload of the intelligent search procedure, analyzed in the next paragraphs.

4.4. *Solution generation – The scheduling process*

The final step towards the solution of a CLP problem is to trigger a search procedure in order to locate and produce the desired solution (or solutions). The choice of a

search procedure is undoubtedly the most sensitive and important of all the choices made during the development cycle of the application. And that is because with a correct decision, there can be an undisputed increase in the program's efficiency. Note here, that in most cases only one solution is required, the one that responds better to a specific collection of criteria. This solution, in the general case, is not the first one to be found. In the timetabling problem, the difference between the first and the final solution is what might be characterized as the transition from a correct to an acceptable solution.

The first attempts toward the generation of an acceptable solution were based on the provided facilities of the ECL^{i}PS^e language. These facilities are nothing more than an implementation of blind search techniques (predicates `indomain/1`, `deleteff/3`, `deleteffc/3`, `labeling/1`) as well as a class of predicates aiming at the blind generation of optimum solutions (predicates belonging in the `min_max` and `minimize` classes). The philosophy of such methods revolves around the creation of a linear expression, consisting of arithmetic constants and program variables, common for all solutions, that represents the cost of a given solution. An optimization predicate is then executed with a number of parameters, two of which are the cost expression and the labeling procedure. Lastly, a *branch-and-bound* process is initialized, aiming to supply the lowest cost solution.

Such methods proved to be inadequate for the solution of the timetabling problem. Their failure was mainly due to the problem's complexity. As it has been established, a number of quality standards exist. The above mentioned cost expression had to encompass all of these standards. An optimization technique, like the one provided by ECL^{i}PS^e, implies that the cost expression has to be helpful toward the separation of the solutions. But, because the problem was so complex and a large number of criteria existed, the cost expression proved to be quite an inflexible one. This had a dramatic effect on the execution time of the scheduling process. For the sake of argument, the program could blindly generate the first solution in, roughly, a few seconds. But when the cost expression and the optimization method were involved, the program needed about twenty minutes to calculate the cost expression and would generate no more than four solutions in two hours. Clearly, a new search procedure had to be developed. This procedure should be one that would take into account all the special characteristics of the timetabling problem and would lead directly to the optimum solution. This was nothing more than an intelligent search procedure based on the problem heuristics.

By developing an intelligent search procedure, an attempt was made to use the human knowledge and understanding of the timetabling problem to the highest degree possible. The basic idea is that, if after the imposition of the constraints and the application of consistency techniques, the possible values of every variable are known, then the problem heuristics can be energized in order to select the most promising value. Moreover, even the variable to be instantiated next can be selected. Additional information can be supplied during the scheduling process, as every value assigned is known to the subsequent selections and assignments.

The developed scheduling process has two modes. It can either act upon individual semesters or spread over all of the courses. In any case, a selection is made aiming to find which subject's variables will be instantiated next. The selection is made as follows. First of all, each day is divided into morning (9:00 – 14:00) and afternoon (14:00 – 20:00) fragments. This is done because when people are asked to declare their availability they tend to think in the more abstract level of mornings and afternoons. Next, the total number of mornings and afternoons where each subject can be scheduled is calculated and, after that, the subject with the least possible alternatives is selected. This is an implementation of what is known as the *first-fail* tactic. When a subject has fewer alternatives, then the chance of an incorrect value assignment is minimized. Moreover, if a subject with more alternatives were to be selected, then its value assignment might have an unwanted effect on the possible values of the more constrained subject, making impossible its value assignment and thus causing unnecessary, as well as predictable, backtracking in the whole process. In case two subjects belong to the same category, then a tiebreaker procedure is put in effect. This procedure shows a preference for stronger typed subjects, i.e. compulsory subjects are to be scheduled before the division basics and so on.

After the subject that will be scheduled is identified, the scheduling process begins. The first step is to retrieve all the possible values for a given lecture (the provided predicate `dvar_domain_list/2` is used for this purpose). One of these values will be chosen as the most promising one. This choice is made according to the quality standards:

(i) Semesters' lecture balancing
(ii) Faculty members' lecture balancing
(iii) Student movement between lectures
(iv) Idle time between lectures
(v) Classroom utilization
(vi) Distance between lectures of the same subject
(vii) Faculty members' preference of not teaching
(viii) No simultaneous subject scheduling preference

The decision process is quite simple. All possible values are sorted with the above mentioned quality standards acting as the criterion. For each value, eight scores are assigned, denoting the corresponding criterion satisfaction degree. The time when the sorting process will decide that one value is better than the other is when it becomes possible to sever the values according to the criteria. This means that it might be the case for not all of the above mentioned criteria to be used. An example can clear this situation. If two values have scores `X=[1,2,3,4,5,6,7,8]` and `Y=[1,2,3,5,6,7,8,9]`, then the sorting process will be able to decide once it reaches the fourth criterion. For the score calculation, knowledge extracted from the program data, as well as knowledge gathered during the scheduling process, is used.

From the discussion above, it becomes obvious that the order in which these criteria are declared in the scheduling process plays a significant role. That is why no static order of criteria exists. The user is in the position to assign priorities to these criteria. A direct consequence of this is that different orders of criteria generate different timetables for the same data. Experimentation in this sector can be a guide by which the user can decide which order produces the best timetables for his or her purposes. Furthermore, when a timetable is stored, it contains the order of criteria with which it was generated.

4.5. *The rescheduling process*

The rescheduling process is quite similar to the scheduling one, as it is based on the same heuristics. However, what is admitted here is the fact that, in order for a rescheduling process to be efficient, a minimal number of alterations have to take place in the program's data. In case there are drastic changes in the input, then a rescheduling process is in danger of degenerating into a full-scale scheduling one, but somehow aggravated by all these actions that aim to construct a timetable with a minimum number of differences to the original one.

In a nutshell, the course of a rescheduling process is the following. Again, variables have to be identified, have their domains defined and constrained. But, as now there might be courses that appear in the new curriculum and not appear in the former (i.e. one of the changes might be the addition of courses to the curriculum), a rescheduling process has to take effect based on the common data of the two problems. This process will create a timetable with the least possible number of changes, which could be zero, if the only alterations on the input are limited to the addition of new subjects. When the rescheduling process comes to an end, a scheduling process is initialized for the remaining subjects, obviously taking into account the already rescheduled timetable. This process has the goal of inserting the lectures of the new courses in specific voids in the rescheduled timetable. This approximates the actions of a human if he or she was facing a similar problem.

As stated, the rescheduling process is based on the admittance of a minimum number of changes. In the other aspects, it follows a process resembling the one of the scheduling process. Again, a selection of the next subject to be processed is made and, again, this selection can be made between subjects of the same semester, or between the heap of all of the subjects, regardless of semester. Once a subject is selected, the main procedure is put into effect. The algorithm used is quite simple. If the lectures of a given course can be held on the same time, then this is the choice made. No changes are made. If for some reason this cannot be the case, then the scheduling procedure is triggered, sorting all the possible values, in order to find the most promising one. Obviously, this alteration will create a difference to the original program. The heuristic here is that if a small number of changes in the data exist, small enough for a rescheduling process to make sense, then the majority of the rest of the timetable will have the minimum number of changes. Furthermore, in case a change has to be made, then the use of the same heuristics

tends to ensure the least possible actual distance between the two timetables, the original and the rescheduled one. A faculty member can accept the fact that one of his or her lectures has shifted an hour earlier or later. Although this is a change to the original timetable, it is not a big change. But if a faculty member is asked to adhere to a completely different timetable, this cannot be easily accepted. The various tests given on this sector to the rescheduling procedure have proved that it behaves well enough under circumstances of this kind.

The next step in the rescheduling process is the course addition process, which is a partial scheduling one. It is partial, because a timetable already exists, stemming from the rescheduled version of the original timetable. The heuristics are the same as in the scheduling process. In case the change in the application's input is only this addition of courses, then there is a great possibility that these courses will be scheduled in existing voids of the timetable. This is actually what someone would hope for, that is the same timetable only now enriched with the lectures of the new subjects. Of course, for that to be the case, the new courses must be accompanied with a real flexible set of constraints. If changes have to be made, then the rescheduling process will make them, trying to keep the smallest possible distance to the original timetable.

The rescheduling process seems quite naive in its concept. But the fact is that it works under real-world circumstances. Moreover, the time needed to reschedule an existing timetable, if the principle of a minimum number of changes is respected, can be limited to half of the time needed to construct a timetable from scratch with the help of the previously mentioned scheduling process.

5. ACTS at Work

For an application to be used and widely accepted, a number of properties have to exist. The first of these properties is the implementation of an easy to use as well as complete and flexible interface. ACTS provides this interface, developed on a particular extension of the ECL^{i}PS^e language, the ProTcXl library, based on the popular scripting language Tcl and the Tk toolkit. This interface comes with a number of interesting facilities.

The first of these facilities is the fact that the user can input new data in an easy way. These data can be stored, retrieved and manipulated in any way possible. All of the processes presented in the previous paragraphs are within the user's grasp. He can also change the order of the specified quality criteria at will. When all the timetable parameters are set, a scheduling process can be initialized (Fig. 1). During this process, the user receives substantial information on what the program is doing at a given time (i.e. selecting a subject to be scheduled, imposing constraints or assigning values). Additional flexibility is provided by the fact that the user can combine a given set of data with different timetables generated from this set of data, by different criteria orders.

When a timetable is created, it is presented to the user (Fig. 2). ACTS comes equipped with a complete, fully functional timetable editor. In this process, which

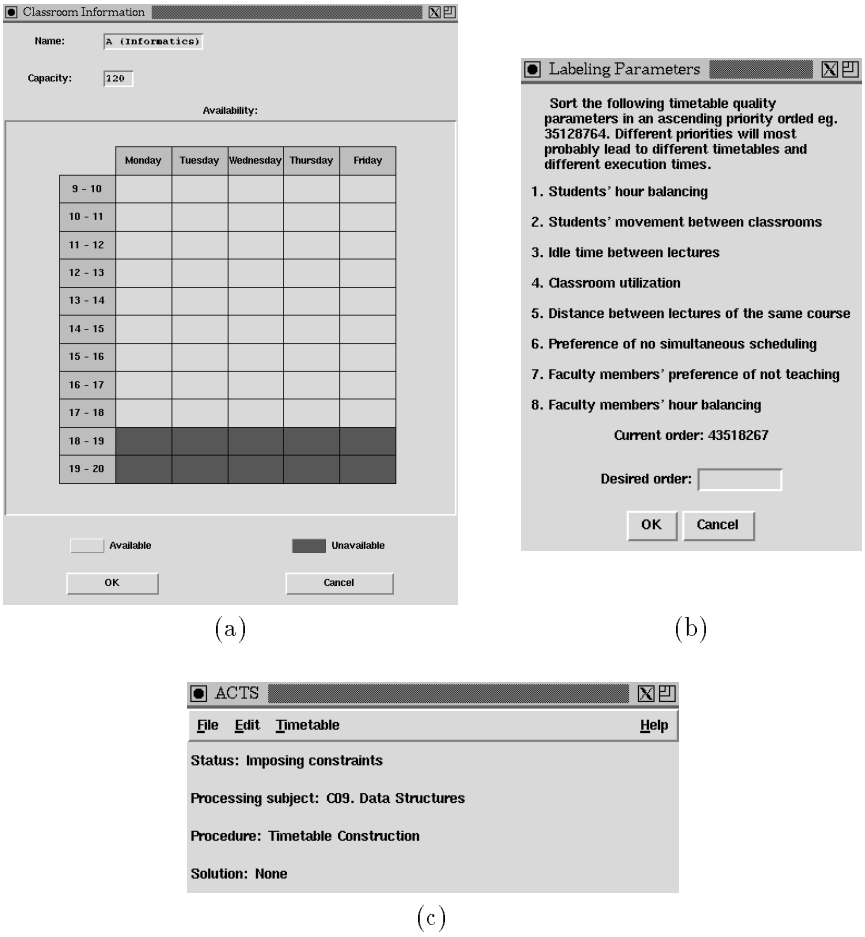

Fig. 1. A data file is opened. ACTS awaits for classroom availability (a) and timetable parameters to be set (b). After that, the scheduling process is executed (c).

was previously referred to as post-editing, the user can make changes to the generated timetable, if he or she thinks that the outcome was not satisfactory. This editor can also be used in order to create manually a timetable from scratch. However, the most unique feature of the timetable editor is the fact that it supplies a timetable verifier, a process that checks the timetable for possible errors or misjudgments. If any errors are found, the user is informed about the number of errors and, for each individual error, the day, teaching period, classroom and subject session that generated it, accompanied by an explanation of the error and what must be done in order to fix it.

Another important feature of ACTS is its ability to create ready-to-publish timetables. After the user is satisfied with the generated or altered timetables, he or she can save the timetables both in text format, as well as the more popular and

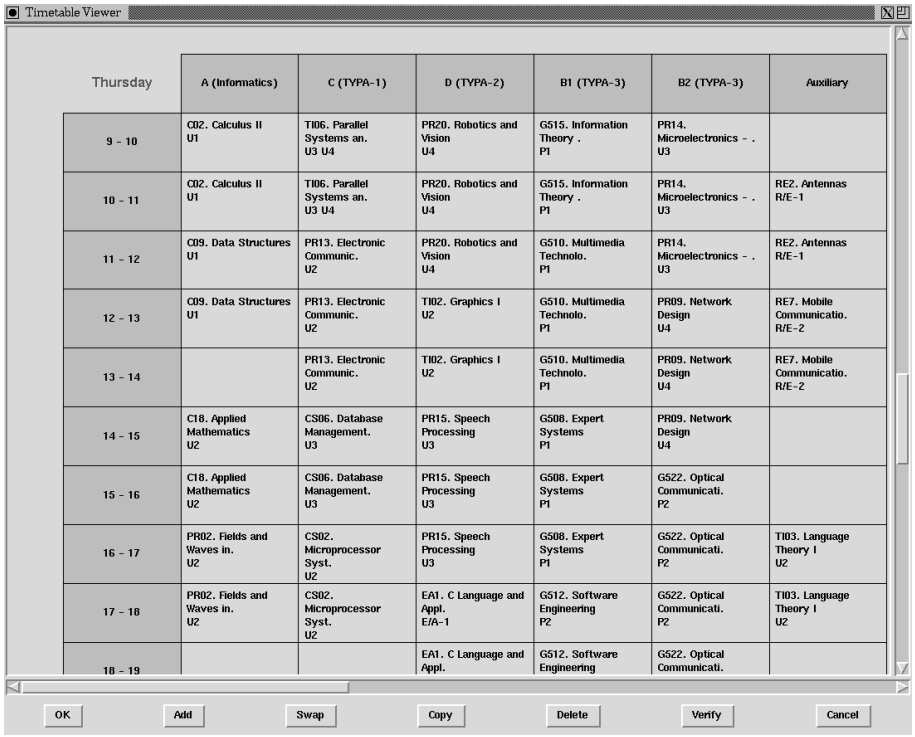

Fig. 2. Timetable viewer and editor. The edit commands and the verification facility are also visible.

more detailed HTML format. A timetable can be saved in a "by day" or a "by classroom" way. In any case, the generated timetable can be easily published both on a bulletin board, if printed, or directly on the World Wide Web.

The final feature of ACTS is that it comes with an on-line help facility, ready to guide the user through the whole process.

6. ACTS Evaluation

The program's efficiency is very high. It has been applied to real-world data sets, such as the ones of DI/UoA, consisting of five to six classrooms, roughly sixty courses and fifty faculty members and teaching assistants. By use of a quite satisfactory priority assignment to the quality criteria, the developed application is in the position of generating the final solution in about twenty to thirty minutes running on a Sun SparcServer 1000. The timetable generated is a nearly optimum one, as it satisfies the quality standards to the highest possible degree.

It should be stressed that the most sensitive of the decisions made during the data inputting for a timetable under construction is the priority assignment to the quality criteria. These are the guides by which the intelligent search procedure

is directed. A "bad" choice in this sector is possible to result in a continuous backtracking of the solution generating procedure, thus substantially increasing the program's execution time. There is no doubt that a solution will be generated eventually, but it will not be such a good one and it will not be found in a reasonable amount of time. The system's monitoring facilities can be helpful in acknowledging and overcoming such unfortunate decisions. As it was previously mentioned, the user has complete knowledge of the program's actions. Backtracking can be easily realized. After experimentation, an empirical method of acknowledging such problems can be the fact that if backtracking occurs at the early or middle stages of the search process, then no real problems exist. However, if the system starts to fluctuate near the end of the search process (i.e. only with a couple of subjects left to be scheduled), this means that it is in trouble and most likely it will not overcome these difficulties. Experimentation acted in a useful way in finding the default criteria order which the program starts with. It is an order that almost always yields successful results, no matter how complex the data are and how hard to solve the constraints implied appear.

The program's main disadvantage is its cost in system resources, both in memory and in computational aspects. It must be understood that the timetabling problem is quite a difficult one. A program made to automate this process has to deal with an extremely large number of facts, an even larger number of variables arising from these facts, as well as a number of constraints that have to be imposed and solved. On the other hand, the quality of the generated solutions is quite high. The timetables generated can be used, either identical or with minor changes, with no apparent second thoughts. The program's ability to create different solutions through different criteria orders allows the user a wide space for experimentation. Its monitoring facilities are quite helpful in order for the user to decide which criteria order suits him or her best.

The rescheduling process has a solid and satisfactory behavior most of the time. For a minimum number of changes, the program resembles human action. Its flexibility and its ability to deal with almost any rescheduling requirement (i.e. data alterations, curriculum changes, etc.) makes it quite a powerful tool in order to tackle specific instances of the rescheduling problem in the construction of timetables.

The existence of an easy to use and substantially flexible interface is another plus. The program's provided facilities of editing an already generated timetable, as well as the ability to verify any changes made, are two of its strongest properties. Particularly, the verifier module is probably the most important one, as it gives the user the chance of checking without problems all the changes made, as well as being able to acknowledge all the potential errors and the reasons that led to their existence. The timetable editor can in fact be regarded as an autonomous feature of the application, as it gives the user the framework for creating timetables even from scratch.

But the most powerful feature of ACTS is its ability to handle and work with

real-world data. Not only does it generate solutions, but also it generates them in a very short period of time. And what is more, these solutions are either completely or nearly optimum ones, meaning that they are ready to be published. Its use in the creation of the current academic year's timetable in DI/UoA offers evidence conducive of the program's success. It is the authors' opinion that this success originates from the use of CLP in the solution process, as well as the close approximation of the human expert's actions, by use of the developed intelligent search procedure. All of the above make the authors believe that ACTS is on the right path toward fully automating the timetabling process.

7. Conclusions and Further Work

In this paper, a CLP based approach for tackling the university course timetabling problem is presented. The aim has been to define the set of feasible solutions through a number of constraints and to search for a near optimum solution, according to some quality criteria, by applying intelligent techniques borrowed from the way human experts deal with the problem. A specific system has been built, named ACTS (Automated Course Timetabling System), using the CLP language ECL^{i}PS^e. This system has been used in real-life for the satisfaction of the course timetabling needs of the Department of Informatics of the University of Athens (DI/UoA). A number of possible enhancements can be made, though, to what has already been established as a successful application.

The first of these enhancements has to do with overcoming the program's "greedy" computational nature. A possible breakthrough in this area could be the change of the program's lecture representation. Instead of using hourly sessions as the guide, lectures lasting more than one hour can be used. This means a demonstrable decrease in the space needed to represent courses. For instance, if three four-hour courses, to be split in two two-hour lectures, existed, then, while in the existing representation twelve variables were needed, in the new one six would be used, meaning a fifty percent reduction in the memory requirements. However, this means that a whole new set of constraints would have to be developed in order to handle the new representation. This would also result in changes in the search procedure. Whether a change this drastic will yield better results remains to be seen.

The search procedure offers ground for substantial improvement. It has been stated that the program offers a unique solution, the one that is characterized as the best according to its integrated heuristics. However, such problems do not necessarily have one solution that is far better by the others, but they rather have a group of satisfactory solutions. The backtracking mechanism of the Prolog engine can be used in order to provide the possibility of other solutions making the way to the user's knowledge. But this is not necessarily the case. In most cases, what is needed is not just a different solution. A difference can be just the swapping of two lectures, but that does not mean that there is a substantial difference between the presentation of the two solutions. The quest is for a solution that is *sufficiently different* to the original one. This constitutes a solution that has a maximized

distance from a given one, but has similar properties in the quality aspect of the timetable.

It must have become obvious by now that ACTS is strongly bound to the requirements of a specific academic department, namely DI/UoA. Nevertheless, it has been integrated with principles found in any instance of the course scheduling problem, for any particular department. These principles can be used as the framework on which specialized timetable generators would be built.

A final enhancement, and certainly the most ambitious one, is the elevation of the developed application to another level, in order to construct a machine learning system, a system that would be more than a "simple" timetable construction procedure. What someone would like to have is a system that would learn a little each time the timetable construction process was carried out and would keep this knowledge in order to overcome potential difficulties it would face in the future. This system should not just use static heuristic principles, but it would use past experience in order to create better timetables and it would eventually evolve with the same rhythm as all the parameters of a timetabling procedure. But a system of this kind cannot be visible for some time, certainly not in the ten months that were needed in order for ACTS to be developed. Nevertheless, that system would certainly denote a new era in the area of real-world problem solving.

References

[1] J. A. Ferland, *Generalized assignment-type problems: A powerful modeling scheme*, in Proceedings of the 2nd International Conference on the Practice and Theory of Automated Timetabling PATAT '97, University of Toronto (1997) 27–54.

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

[3] A. Schaerf, *A survey of automated timetabling*, Technical Report CS-R9567, CWI, (1995).

[4] T. B. Cooper and J. H. Kingston, *The complexity of timetable construction problems*, in Proceedings of the 1st International Conference on the Practice and Theory of Automated Timetabling PATAT '95, volume 1153 of Lecture Notes in Computer Science, eds. E. Burke and P. Ross, Springer-Verlag (1995) 283–295.

[5] D. de Werra, *The combinatorics of timetabling*, European Journal of Operational Research **96** (1997) 504–513.

[6] C. Gotlieb, *The construction of class-teacher timetables*, in Proceedings of the IFIP Congress (1962) 73–77.

[7] N. Mehta, *The application of a graph coloring method to an examination scheduling problem*, Interfaces **11** (1981) 57–64.

[8] M. Badri, *A two-stage multiobjective scheduling model for [faculty-course-time] assignments*, European Journal of Operational Research **94** (1996) 16–28.

[9] A. Tripathy, *A lagrangian relaxation approach to course timetabling*, Journal of the Operational Research Society **31** (1980) 599–603.

[10] J. Ferland and S. Roy, *Timetabling problem for university as assignment of activities to resources*, Computers and Operations Research **12(2)** (1985) 207–218.

[11] J. Thompson and K. A. Dowsland, *General cooling schedules for a simulated annealing based timetabling system*, in Proceedings of the 1st International Conference on the

Practice and Theory of Automated Timetabling PATAT '95, volume 1153 of Lecture Notes in Computer Science, eds. E. Burke and P. Ross, Springer-Verlag (1995) 345–363.

[12] S. Elmohamed, P. Coddington and G. Fox, *A comparison of annealing techniques for academic course scheduling*, in Proceedings of the 2nd International Conference on the Practice and Theory of Automated Timetabling PATAT '97, University of Toronto (1997) 146–166.

[13] J. P. Boufflet and S. Nègre, *Three methods used to solve an examination timetabling problem*, in Proceedings of the 1st International Conference on the Practice and Theory of Automated Timetabling PATAT '95, volume 1153 of Lecture Notes in Computer Science, eds. E. Burke and P. Ross, Springer-Verlag (1995) 327–344.

[14] G. M. White and J. Zhang, *Generating complete university timetables by combining tabu search with constraint logic*, in Proceedings of the 2nd International Conference on the Practice and Theory of Automated Timetabling PATAT '97, University of Toronto (1997) 268–277.

[15] A. Colorni, M. Dorigo and V. Manniezo, *Genetic algorithms: A new approach to the time-table problem*, in Combinatorial Optimization, volume F82 of Lecture Notes in Computer Science — NATO ASI Series, Springer-Verlag (1990) 235–239.

[16] W. Erben and J. Keppler, *A genetic algorithm solving a weekly course-timetabling problem*, in Proceedings of the 1st International Conference on the Practice and Theory of Automated Timetabling PATAT '95, volume 1153 of Lecture Notes in Computer Science, eds. E. Burke and P. Ross, Springer-Verlag (1995) 198–211.

[17] D. Corne and P. Ross, *Peckish initialisation strategies for evolutionary timetabling*, in Proceedings of the 1st International Conference on the Practice and Theory of Automated Timetabling PATAT '95, volume 1153 of Lecture Notes in Computer Science, eds. E. Burke and P. Ross, Springer-Verlag (1995) 227–240.

[18] P. Ross, E. Hart and D. Corne, *Some observations about GA-based exam timetabling*, in Proceedings of the 2nd International Conference on the Practice and Theory of Automated Timetabling PATAT '97, University of Toronto (1997) 55–71.

[19] E. K. Burke, J. P. Newall and R. F. Weare, *A memetic algorithm for university exam timetabling*, in Proceedings of the 1st International Conference on the Practice and Theory of Automated Timetabling PATAT '95, volume 1153 of Lecture Notes in Computer Science, eds. E. Burke and P. Ross, Springer-Verlag (1995) 241–250.

[20] M. Yoshikawa, K. Kaneko, Y. Nomura and M. Watanabe, *A constraint-based approach to high-school timetabling problems: A case study*, in Proceedings of the 12th National Conference on Artificial Intelligence AAAI '94 (1994) 111–116.

[21] M. Henz and J. Würtz, *Using Oz for college timetabling*, in Proceedings of the 1st International Conference on the Practice and Theory of Automated Timetabling PATAT '95, volume 1153 of Lecture Notes in Computer Science, eds. E. Burke and P. Ross, Springer-Verlag (1995) 162–177.

[22] P. David, *A constraint-based approach for examination timetabling using local repair techniques*, in Proceedings of the 2nd International Conference on the Practice and Theory of Automated Timetabling PATAT '97, University of Toronto (1997) 132–145.

[23] P. Boizumault, Y. Delon and L. Peridy, *Constraint logic programming for examination timetabling*, The Journal of Logic Programming **26** (1996) 217–233.

[24] H. Frangouli, V. Harmandas and P. Stamatopoulos, *UTSE: Construction of optimum timetables for university courses — A CLP based approach*, in Proceedings of the 3rd International Conference on the Practical Applications of Prolog PAP '95 (1995) 225–243.

[25] C. Cheng, L. Kang, N. Leung and G. M. White, *Investigations of a constraint logic programming approach to university timetabling*, in Proceedings of the 1st International Conference on the Practice and Theory of Automated Timetabling PATAT '95, volume 1153 of Lecture Notes in Computer Science, eds. E. Burke and P. Ross, Springer-Verlag (1995) 112–129.

[26] G. Lajos, *Complete university modular timetabling using constraint logic programming*, in Proceedings of the 1st International Conference on the Practice and Theory of Automated Timetabling PATAT '95, volume 1153 of Lecture Notes in Computer Science, eds. E. Burke and P. Ross, Springer-Verlag (1995) 146–161.

[27] R. Kowalski, *Algorithm = Logic + Control*, Communications of the ACM **22(7)** (1979) 424–436.

[28] P. van Hentenryck, *Constraint Satisfaction in Logic Programming*, The MIT Press (1989).

[29] T. Frühwirth, A. Herold, V. Küchenhoff, T. Le Provost, P. Lim, E. Monfroy and M. Wallace, *Constraint logic programming — An informal introduction*, in Logic Programming in Action, eds. G. Comyn, N. E. Fuchs and M. J. Ratcliffe, Springer-Verlag (1992) 3–35.

[30] *ECL^{i}PS^e 3.5: User Manual*, ECRC (1995).