Course Scheduling in an Adjustable Constraint Propagation Schema

Nikolaos Pothitos, Panagiotis Stamatopoulos, and Kyriakos Zervoudakis
Department of Informatics and Telecommunications, University of Athens,
Panepistimiopolis, 157 84 Athens, Greece
{pothitos,takis,quasi}@di.uoa.gr

***Abstract*—Constraint Programming constitutes a prominent paradigm for solving time-consuming Constraint Satisfaction Problems (CSPs). In this work, at first we model a generic course scheduling problem as a CSP, that complies with the *International Timetabling Competition* (ITC) standards. Constraint Programming allowed us to search for a solution via several state-of-the-art methodologies and compare them. For the stochastic search methods, we propose new hybrid semi-random heuristics. Second, we chose to maintain bounds consistency during search to prune 'no-good' branches of the search tree. We theoretically define new lightweight consistency types, namely k-bounds-consistency, in order to speed up the overall search procedure. Eventually, we process real world data and show the efficiency of our proposal: While plain backtracking produces poor results, constraint propagation dramatically boosts the solutions quality, and can be 'fine-tuned' in our adjustable schema to make it even faster.**

***Keywords*-timetabling; bounds consistency; search.**

I. Introduction

Scheduling activities which depend on resources is a common problem addressed for almost half a century [1]. A wide spectrum of techniques to cope with it have been evolved so far [2].

A. Multidisciplinary Contributions to Timetabling

Automated timetabling is a common scheduling problem that occurs in every educational institute. There have been developed a lot of ways to solve it [3]. The similarities to the problem of graph coloring were used to invent common procedures to solve them both [4]. Timetabling was also correlated with the general class of network flow problems [5]. Other methods include clustering of the problem to smaller sub-problems [6]. The application of case-based reasoning to timetabling also gives promising results [7].

Because of the hardness of finding an optimum solution to the problem, a lot of *metaheuristic* methods have been used. In general, these methods begin from a state of the variables of the problem, and try iteratively to reach another state that is closer to a solution, i.e. that is better than the one already found, if any. The disadvantage of these methods is 'sticking' in *local* optima, but there are a lot of ways to override them. Methods widely used in Artificial Intelligence, such as simulated annealing [8], genetic algorithms [9], and tabu search [10], were applied in this problem. Local Search techniques have been recently evolved to produce near optimal solutions [11].

A common way to implement applications in Artificial Intelligence is to define the constraints of the problem in a Constraint Programming framework with a solver that uses Logic Programming [12], [13], or other environments [14]. To unify all these variations, common criteria have been suggested to measure the efficiency of automated timetabling applications [15].

There has been made great effort towards reducing the computational time needed to construct a timetable. Nevertheless, today the main obstacle for the spreading of timetabling systems is lack of flexibility; in many cases they cannot adapt to the different requirements of each educational institute and the complicated constraints that may exist. So, the main contribution of Constraint Programming in this direction is the separation of the statement of the problem and the mechanism that solves it [16]. This is the key feature that made this Artificial Intelligence paradigm popular to the programmers' community and this is how we faced automated timetabling in this work: We defined explicitly the entities of the problem and we simplified the constraints that connect them. Their simplicity makes the whole problem portable to many solvers that use Constraint Programming (CP) [17], as well as Constraint Logic Programming (CLP) [18]. Besides, this framework makes it easy to add new constraints or to modify existing ones, without affecting the search implementation.

Our first contribution is the statement of the course scheduling as a CSP. After the CSP has been defined—including all of its variables and the optimization criteria—several direct search methods are combined with other propagation techniques and well-aimed heuristics to solve it. In this context, we also propose new hybrid semi-random heuristics and compare them with the systematic ones.

The Constraint Programming paradigm is ideal for 'plugging' into our course scheduling CSP many different generic search methods and heuristics, because the statement and solution phases are completely independent, and we do not have to declare the problem and its constraints from scratch. This philosophy allowed us not only to make comparisons between many search procedures on this demanding CSP, but also to propose and test a new constraint propagation schema.

B. Constraint Propagation and Related Work

Roughly speaking, constraint propagation is a process used to transfer the modifications of the variables of a CSP across the constraint network, in order to make all the variables and their assignments compatible to each other. Intuitively, domino game is a mechanical equivalent to this technique, where the tiles falling in turn symbolize the chain variable assignments fired by constraint propagation.

In this context, a propagation algorithms series have been evolved, such as AC-3, AC-5, AC-2001, etc., where 'AC' stands for 'arc-consistency' [19]. There exists a plethora of ways to make one variable of a problem consistent to another, as there are many consistency levels, with the most prominent ones elaborated in Section V.

Propagation methods incorporate an *event* queue, containing all the previous assignments/modifications to the variables [20]. Each event in the queue is propagated to the variables, in order to make them consistent to the current assignments. For contemporary CSPs—like course timetabling—with too many variables, this 'communication' between all the variables becomes inefficient and may result into thrashing.

Related work leverages on limiting the queue itself, by preventing the insertion of events, when either the distance from the previous event is above a threshold, or the event had limited impact on the variable that created it, i.e. when it did not remove a certain proportion of possible values from its domain [21].

So, the second contribution of this work targets towards adjusting constraint propagation, in order to make it adaptable to search methods. We focused on the 'tug of war' between different consistency levels, and we proposed a compromise. Instead of tampering with the event queue itself and the propagation algorithms, we introduced a new consistency level. The algebraic evidence was verified in practice in the very challenging timetabling problem.

II. Timetabling: Entities, Properties, Relations

First of all, we typically define the critical entities involved in course timetabling, and we elaborate on their properties and the relations between them.

A. Days, Timeslots, and Periods

Let D be the number of *days* of the timetable; each day has H *timeslots.* P defines a set $\{0, 1, \ldots, D \cdot H - 1\}$ that includes all the teaching *periods.* The actual duration of a teaching period is of no interest for the timetabling application. A teaching period cannot be divided.

B. Courses and Curricula

Let C be the set of *courses,* L the set of *lectures* and G the set of *groups* of courses, also known as *curricula.* Each group $g \in G$ consists of courses ($\forall g,\ g \subseteq C$), and each course $c \in C$ consists of one or more lectures $\ell \in L$. This hierarchy supports the following conditions that often occur in academia:

1) Usually, a student attends a specific group of courses. For example courses are grouped according to the semester they belong to, or the general direction they serve; e.g. we may have the groups '1^{st} Semester Courses,' 'Courses of Mathematical Science,' etc. Two courses that are members of the same group should *not* be scheduled at the same time. The best thing that one could do is to schedule the corresponding lectures of each group as close as possible, so that the personal timetables of most students have no idle periods.
2) Normally, each course is broken up into lectures, that should be spread during the week. For each course c_i, there is a minimum desired number of days on which the lectures will be distributed over the week, namely $\mathsf{mwd}(c_i)$, where mwd stands for *minimum working days.*
3) The number of the lectures for a course is predefined and given by the property $\mathsf{dur}(c_i)$. There is also a function named 'course' that maps L to C, i.e. $\mathsf{course}(\ell)$ is the course that lecture ℓ belongs to.

It is noted that each course has the property $\mathsf{na}(c_i) \subseteq P$ which contains the periods when its lectures cannot been given; 'na' stands for 'unavailable.' E.g. $\mathsf{na}(c_2) = \{1, 5\}$ means that course c_2 will not be taught during periods 1 and 5.

The last given property for a course is the number of the students that attend it, namely $\mathsf{students}(c_i)$.

C. Teachers and Rooms

Certainly, each course should be provided a teacher/ professor and a (class)room. Let T be the set of *teachers* and R the set of *rooms;* each room is represented by a different integer. The property $\mathsf{teach}(c_i)$ gives the responsible teacher $t \in T$ for each course $c_i \in C$. The capacity of a classroom $r \in R$ is denoted $\mathsf{capacity}(r)$.

D. The Solution to the Problem

For each lecture $\ell_i \in L$, we ask to find the necessary timeslot and room. Conclusively, a *solution* to the problem refers to the full definition of a function $\mathsf{sol} : L \to P \times R$, which maps each lecture to a period and a classroom.

III. Formulating Course Timetabling as a CSP

We are now going to state the problem described above as a *Constraint Satisfaction Problem* (CSP) [22]. A CSP includes a set of *constrained variables* that may be simply called *variables;* each of them corresponds to a set of values called *domain;* we say that a constrained variable x has a domain D_x. Constrained variables are connected to each other through a set of *constraints.* In general, a constraint which refers to specific constrained variables is the set of all valid combinations of values that can be assigned to them.

E.g., for the variables x_1 and x_2 with domains $D_{x_1} = D_{x_2} = \{0, 1, 2, 3\}$, the constraint of equality can be declared as $\mathcal{C}(\{x_1, x_2\}, \{(0, 0), (1, 1), (2, 2), (3, 3)\})$. Although this notation is as general as possible, in practice, i.e. in Constraint Programming, we use simple relations to describe constraint networks. In the above example, the constraint can be simply written as $x_1 = x_2$. In this work, we will only use the constraint types implemented in many CSP solvers.

Finally, a *solution* to a CSP is a valid assignment of a value to each variable, that does not violate the constraints.

A. Variables and Domains

The set $\mathscr{X}$ of constrained variables refers to the teaching period of each lecture:

$$\mathscr{X} = \{x_i \mid D_{x_i} = P \backslash \mathsf{na}(\mathsf{course}(\ell_i)),\ \ell_i \in L\}\,. \quad (1)$$

The other critical set $\mathscr{R}$ of constrained variables for this timetabling problem includes the classrooms where the lectures are given:

$$\mathscr{R} = \{cl_i \mid D_{cl_i} = R,\ \ell_i \in L\}\,. \quad (2)$$

The rest of the constrained variables that will be used are auxiliary.

B. Constraints for Lectures

As we have already stated all critical variables and their domains, we are now going to build up the *constraint network*. When we are interested in finding more than one solution, the solver may output the same solution more than once, if the problem has symmetries. One symmetry is the 'swapping' of two lectures ℓ_i and ℓ_j, when they belong to the same course. To avoid this situation, we add the constraint:

$$x_i < x_j\,, \quad \forall\, \ell_i, \ell_j \in L\,, \text{ with } i < j \text{ and } \mathsf{course}(\ell_i) = \mathsf{course}(\ell_j)\,. \quad (3)$$

C. Resources and Constraints

To understand our approach to the problem, we will look upon teachers, classrooms, and groups of courses as *resources*. Every resource is connected to a set of lessons. In particular, for each teacher $t \in T$ we have

$$\mathscr{X}_t = \{x_i \mid x_i \in \mathscr{X},\ t = \mathsf{teach}(\mathsf{course}(\ell_i))\} \quad (4)$$

and for each group of courses $g \in G$,

$$\mathscr{X}_g = \{x_i \mid x_i \in \mathscr{X},\ \mathsf{course}(\ell_i) \in g\}\,. \quad (5)$$

The global constraint that has to be stated for all sets above is the *all-different*, a constraint that is imposed over a set of variables and is satisfied only if the variables are assigned different values.

So, each resource can be supplied to at most one activity/ lecture during a teaching period. For example, Figure 1 displays a teacher t_1 that gives three lectures ℓ_1, ℓ_2, and ℓ_3. The corresponding constrained variables must not be assigned same values, in other words,

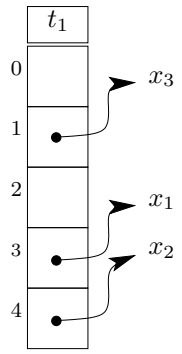

Figure 1. Teacher t_1 as a resource

$$\mathrm{AllDifferent}(\mathscr{X}_t)\,, \quad \forall t \in T\,, \quad (6)$$

$$\mathrm{AllDifferent}(\mathscr{X}_g)\,, \quad \forall g \in G\,. \quad (7)$$

On the other hand, we have only one set of constrained variables for the whole set of classrooms R. We define

$$\mathscr{X}_R = \{y_i \mid y_i = x_i + D \cdot H \cdot cl_i,\ x_i \in \mathscr{X},\ cl_i \in \mathscr{R}\}\,. \quad (8)$$

Every member of $\mathscr{X}_R$ represents the time *and* space where a lecture will take place. Expression $x_i + D \cdot H \cdot cl_i$ is the linear representation of (x_i, cl_i), as x_i cannot exceed $D \cdot H$. In order to avoid having two-dimensional constrained variables in our problem—which are more difficult to use—we did this 'trick' of linearization. So we add another constraint

$$\mathrm{AllDifferent}(\mathscr{X}_R)\,. \quad (9)$$

To better understand the above constraint, see Fig. 2. We have a two-dimensional representation of resource 'classrooms,' because one lecture can be assigned one period from P and one classroom from R. In this figure, we have time and space assignments for lectures ℓ_1 to ℓ_5; also, $D \cdot H = |P| = 5$ and $|R| = 3$. Any two lectures cannot share the same period *and* classroom.

D. Objective Variable for Quality Criteria

In many solvers, a constrained variable is used to describe the objective function. The *objective variable* as we call it, represents the cost, or, better, the quality of the timetable. In the International Timetabling Competition *ITC-2007/08* four quality criteria q_1, q_2, q_3, and q_4 were mixed [23].

1) Room Capacity: Variable q_1 quantifies the first quality factor. It concerns the students that attend a lecture (e.g. ℓ_i), when their number exceeds the capacity of the classroom

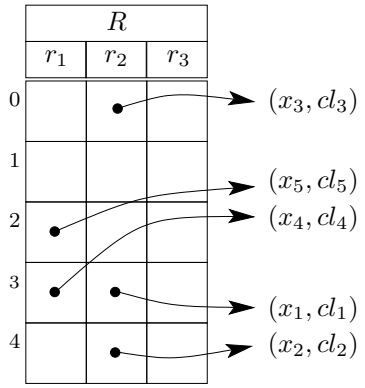

Figure 2. Two-dimensional resource for the classrooms in R

(cl_i) that hosts this lecture. This factor is expressed as

$$q_1 = \sum_{\ell_i \in L} \sum_{\substack{r \in R \\ \mathsf{students}(\mathsf{course}(\ell_i)) > \mathsf{capacity}(r)}} \Big(\big(\mathsf{students}(\mathsf{course}(\ell_i)) - \mathsf{capacity}(r) \big) \cdot \mathrm{bool}(\, cl_i = r \,) \Big) \quad (10)$$

and should be *minimized*—as every quality factor q_i. Note that we take into account the terms with $\mathsf{capacity}(r)$ being strictly less than $\mathsf{students}(\mathsf{course}(\ell_i))$, otherwise the term is not inserted into the sum. The expression '$\mathrm{bool}(condition)$' is a constrained variable with domain $\{0, 1\}$. If $condition$ is true, the variable equals to 1, otherwise it equals 0.

2) Room Stability: ITC-2007/08 specifies that every lecture ℓ_i of the same course c should be hosted at the same classroom; thus one student can easily remember where to attend a specific course. For each different classroom—except for the first—used for the course, one penalty point is added to q_2. In order to state this criterion, an auxiliary constraint is first declared for every course c:

$$\mathtt{Rooms}_c = \mathrm{Inverse}\{R_c\}\,, \quad R_c = \{cl_i \mid cl_i \in \mathscr{R},\ \mathsf{course}(\ell_i) = c\}\,. \quad (11)$$

The above constraint implies that the domain of $\mathtt{Rooms}_c[r]$ contains a positive number, if there exists a $cl_i \in R_c$, with $r \in D_{cl_i}$, else $\mathtt{Rooms}_c[r] = -1$.[1] Therefore, q_2 is easily defined as

$$q_2 = \sum_{c \in C} \left(\sum_{room \in \mathtt{Rooms}_c} \Big(\mathrm{bool}(\, room \geq 0 \,) \Big) - 1 \right). \quad (12)$$

3) Distribution of Lectures During the Week: Another goal is to evenly distribute the lectures of each course during the week. Hence, we try to minimize

$$q_3 = \sum_{c \in C} \left(\mathsf{mwd}(c) - \sum_{day \in \mathtt{Days}_c} \Big(\mathrm{bool}(\, day \geq 0 \,) \Big) \right). \quad (13)$$

The element $\mathtt{Days}_c[\delta]$ is positive only if there exists a lecture of c that is given on day δ. It is produced through the following intermediate constraint:

$$\mathtt{Days}_c = \mathrm{Inverse}\{d \mid d = \left\lfloor \frac{x_i}{H} \right\rfloor,\ x_i \in \mathscr{X},\ \mathsf{course}(\ell_i) = c\}\,. \quad (14)$$

4) Isolated Lectures: We also need a metric for the lectures belonging to a group of courses that are not adjacent to any other lecture of the same group. The more adjacent lectures we have, the better students' personal timetables will be constructed.

For each group, an auxiliary two-dimensional array of boolean constrained variables named `Busy` will be built. If $\mathtt{Busy}_g[d][h] = 1$, then a lecture of group g is given at hour h on day d. Before creating this array, we construct another array for each group of courses using again the 'Inverse' constraint:

$$\mathtt{Timetable}_g = \mathrm{Inverse}(\mathscr{X}_g)\,. \quad (15)$$

The above Inverse constraint implies that the domain of $\mathtt{Timetable}_g[d \cdot H + h]$ contains a positive number, if there exists a $x_i \in \mathscr{X}_g$, with $(d \cdot H + h) \in D_{x_i}$. Therefore, we add the following constraint for each day d and hour h

$$\mathtt{Busy}_g[d][h] = \mathrm{bool}(\, \mathtt{Timetable}_g[d \cdot H + h] \geq 0 \,)\,. \quad (16)$$

We observe that the two-dimensional array $\mathtt{Busy}_g$ is a 'timetable' specific for g, that displays the hours it occupies. When $\mathtt{Busy}_g[d][h] = 1$ and the next and previous timeslots are 0, we have an isolated lecture:[2]

$$\mathtt{Isolated}_g[d][h] = \neg\mathtt{Busy}_g[d][h-1] \wedge \mathtt{Busy}_g[d][h] \wedge \neg\mathtt{Busy}_g[d][h+1]\,. \quad (17)$$

So, the following sum gives the number of gaps for all teams.

$$q_4 = \sum_{g \in G} \sum_{d=0}^{D-1} \sum_{h=0}^{H-1} \mathtt{Isolated}_g[d][h]\,. \quad (18)$$

Now, we can add the four criteria defined—possibly giving them appropriate weights—to construct the final objective variable, i.e.

$$q = q_1 + q_2 + 5 \cdot q_3 + 2 \cdot q_4\,. \quad (19)$$

[1] Practically, the *Inverse* constraint is applied over arrays. If we see the set R_c as an array, then $\mathtt{Rooms}_c[r]$ will contain the indexes of all $cl_i \in R_c$ with $r \in D_{cl_i}$.

[2] We assume $\mathtt{Busy}_g[d][h] = 0$ when h is out of range, e.g. when it equals -1 or H.

5) Optimization for School Timetables: The criteria described are focused on *groups of courses;* the 'objective' is to schedule the lectures of each of them as evenly (during the week) and continuously (during every day) as possible. This optimization goal is common in academic institutes.

When it comes to schools, the objective variable gets different. Specifically, it is constructed in the same way but *it is focused on teachers.* That is, all the formulae presented above in this section will remain the same, except that every '$g\ (\in G)$' should be replaced by '$t\ (\in T)$'. It is obvious that all the effort is now given in optimizing the personal timetables of every teacher.

IV. Search

Having defined the CSP, we should now choose a way to solve it. In general, we assign values to the variables of $\mathscr{X}$ and $\mathscr{R}$ and check whether they satisfy the constraints. First of all, heuristics are used to guide search towards a solution.

A. Heuristics

Search methods 'consult' with heuristics in order to explore as soon as possible the most promising branches of the search tree. When a search method has to choose the next variable to instantiate and the value to assign to it, it uses respectively *variable ordering heuristics* and *value ordering heuristics.*

1) Variable Ordering Heuristics: We construct a heuristic function that chooses the next uninstantiated constrained variable $x_i \in \mathscr{X}$, that corresponds to lecture ℓ_i. We use various ways to distinguish variables.

- We choose the variable x_i that has the *minimum* domain size $|D_{x_i}|$; so variables with small domains are favored, according to the first-fail heuristic [24].
- When we have a 'tie,' i.e. when the above criterion gives two or more variables with same domain size, we use another heuristic as *tie-breaker,* instead of choosing a variable among them at random. In this case, we choose the variable that is involved in the *maximum* number of constraints, in line with the *degree* criterion. We denote E_{x_i} as the number of constraints involving x_i, and e as the maximum number of constraints.

If we combine the above two heuristics, we will produce the expression

$$h_i = \Big(D \cdot H - |D_{x_i}|\Big) \cdot (e+1) + E_{x_i}\,. \tag{20}$$

$|D_{x_i}|$ takes precedence over E_{x_i}, because the latter cannot exceed e. We negated $|D_{x_i}|$, because we seek for a *minimum* value; note that it cannot exceed $D \cdot H$, so $(D \cdot H - |D_{x_i}|)$ is always positive.

2) A New Semi-random Variable Ordering Heuristic: Stochastic search methods require to choose the next variable at random, so we designed the corresponding random heuristic. Actually, we went one step further by providing a *level* of randomness, called $rand$. While $rand$ approaches zero, the heuristic becomes more random, but while $rand$ grows, the heuristic approximates the above two ones, in Section IV-A1.

More specifically, we use the metric h_i in (20), and we transform it to $h'_i = h_i^{rand}$, in order to give it more or less strength, if $rand \to \infty$ or $rand \to 0$ respectively. Then we construct the following sequence:

$$H_0 = 0\,,$$
$$H_i = \frac{h'_i}{\sum_{j=1}^{n} h'_j} + H_{i-1}\,, \quad 1 \le i \le n\,.$$

From this sequence, we can produce n mutually exclusive ranges:

$$I_i = [H_{i-1}, H_i)\,, \quad 1 \le i \le n\,.$$

The union of all of these sets is $[0,1)$. Each I_i is proportionally wide to h'_i.

Almost every computer platform can provide us with a random number uniformly selected from $[0,1)$. Consequently, if we generate at random a value in $[0,1)$, we can take the corresponding I_i, i.e. the corresponding x_i. While $rand$ falls, the selection becomes more arbitrary.

3) Value Ordering Heuristics: Having found an uninstantiated variable $x_i \in \mathscr{X}$, using one of the above heuristics, we should assign to it a value from its domain. To do so, we choose the value that, when assigned, it will provoke the less reduction to the domains of the other values, than the rest of the possible assignments. We also favour the assignment of values that will reduce the solution cost/objective.

B. Search Methods and 'Naxos' Solver

The CSP defined took shape in our CSP solver, called 'Naxos' [25]. Naxos is a library for an object-oriented programming environment, implemented in C++. It allows the statement of CSPs having constrained variables with finite domains containing integers.

The search engine incorporated in Naxos is based on the propagation of the modifications of the domains over the constraint network. For example, the assignment of a value to a constrained variable should make other variables connected to it consistent with that value.

Moreover, the application developer that uses Naxos can create custom goals to be satisfied, and thus, he/she can make search goal-driven and control it in the way he/she likes. Special goals, namely *OR-goals,* define *choice points* in search trees, i.e. they generate two branches: one left and one right branch. If the left branch leads to a dead-end, Naxos *backtracks* to the choice point and continues to the right branch. A dead-end is reached when the domain of any variable becomes empty, e.g. when we have no time slot available for a lecture, due to current assignments.

Hence, Naxos supports a plethora of search methods such as Depth First Search, Limited Discrepancy Search

[26], etc. In this work, we mainly use a *Depth-bounded Backtrack Search* (DBS) method [27].

V. Consistency Types: Analysis and Improvements

It has been shown that for many problems *constraint propagation* procedure gives better results than ordinary backtracking search methods [28]. This methodology suggests that when we assign a value to a constrained variable, or, generally, when we shrink its domain, we should enforce some type of consistency to the constraint network, in order to a priori prune 'no-good' branches of the search-tree.

A. Arc-Consistency

Arc-Consistency is the most well-known and older consistency type.

Definition 1. *We say that an arc (X, Y)—connecting variables of the constraint network—is* arc-consistent *iff for all values $x \in D_X$, there exists a value $y \in D_Y$ such that the constraint that connects the two variables is satisfied. When every arc of the constraint network is arc-consistent, we say that the constraint network is* arc-consistent *too.*

Example 1. Let the constrained variables X and Y have the domains $D_X = \{5, 6, 8\}$ and $D_Y = \{0, 1, 2, 3\}$. Say that we have the constraint $X = Y + 5$. The arc (X, Y) is *consistent*, because for each $x \in D_X$, there is a $y \in D_Y$, with $x = y + 5$.

However, the arc (Y, X) is *inconsistent*, because for $y = 2$ there is no *support* $x \in D_X$, with $x = y + 5 = 2 + 5 = 7$. In this case, in order to *enforce* arc-consistency, we must remove 2 out of D_Y.

Lemma 1. Arc-consistency check and enforcement, also called *revision*, for an arc (X, Y), has worst time complexity $O(|D_X| \cdot |D_Y|) = O(d^2)$, where d is the maximum domain size in the CSP in question.

The proof is somehow trivial, as for each value $x \in D_X$ we seek a support $y \in D_Y$.

Arc-consistency does not lead necessarily to a solution—but if there is no arc-consistency, we are sure that we have no solution—unless we combine it with a search method. Its usefulness has to do with the reduction of the search space that the search method explores.

B. Bounds-Consistency

Bounds-consistency is a weaker type of consistency that requires that only the *bounds* of the domain of each variable should have a support value in the domains of the variables it is connected to via constraints [29].

Definition 2. *A constrained variable X is* bound-consistent[3] *with regard to variable Y, iff there exist values $y_1, y_2 \in D_Y$, such that the assignments $(X \leftarrow \min D_X,\ Y \leftarrow y_1)$ and $(X \leftarrow \max D_X,\ Y \leftarrow y_2)$ do not violate the constraints.*

[3]This definition of bounds-consistency appears in the bibliography as *bounds(D)-consistency* too.

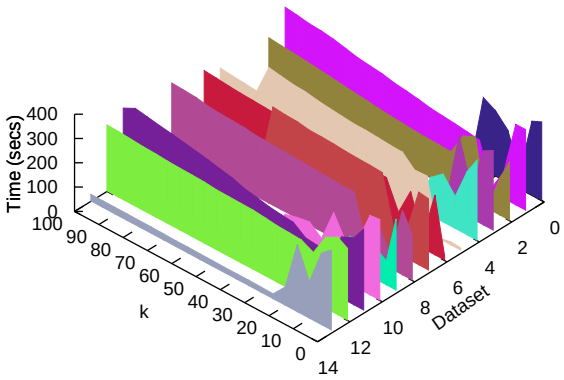

Figure 3. Time needed to solve fourteen ITC datasets

An assignment $X \leftarrow v$ designates the restriction of the domain D_X to contain only the value v, i.e. $D_X = \{v\}$.

Example 2. Let X and Y be constrained variables with domains $D_X = \{5, 6, 8\}$ and $D_Y = \{1, 2, 3\}$, and, again, the constraint $X = Y + 5$. There is *no* bounds-consistency, as there is no pair $(\min D_X,\ y)$, i.e. $(5, y)$, with $5 = y + 5$, because $0 \notin D_Y$.

But if the domain of X is shrunk to $D'_X = \{6, 8\}$, then we do have bounds-consistency enforcement. It is worth mentioning that in this case we do *not* have arc-consistency, because $2 \in D_Y$ does not have any support $x \in D_X$, with $x = 2 + 5$.

Lemma 2. Bounds-consistency revision of (X, Y) has worst time complexity $O(d^2)$.

Proof: Again, the *revision* of an arc/constraint includes the *check* for support values *and* the consistency *enforcement*. To check for the consistency we need $O(2 \cdot |D_Y|)$ steps, as for each one of the two D_X bounds, we try to find a support value $y \in D_Y$.

But we must also consider what happens when a D_X bound is found inconsistent. In this case we should *enforce* bounds-consistency by removing the inconsistent bound out of D_X and by repeating the above *check* for the new bound. We may have $O(|D_X|)$ removals.

As a result, the overall revision complexity is $O(|D_X|) \cdot O(2|D_Y|) = O(d^2)$, as in arc-consistency. ■

What makes bounds-consistency a strategic choice in relation to arc-consistency for CSPs with many variables, such as common course scheduling, is the low *space* complexity. While arc-consistency may modify/remove any value in D_X, bounds-consistency affects only the domain *bounds*. In the first case we *necessarily* need an array to store the domain, but in the latter one, only two values are modified

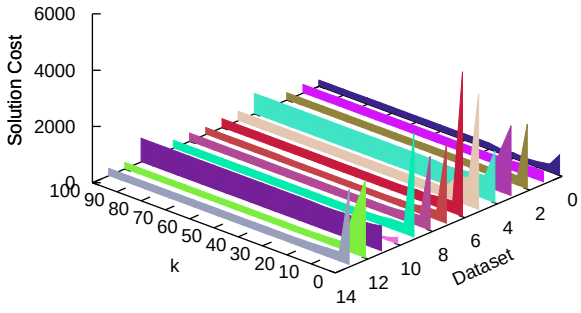

Figure 4. The objective/cost function value for the solutions found

in memory, i.e. the bounds.

C. The New k-Bounds-Consistency

We propose a looser consistency type, which enforces bounds-consistency only to the variables with domain sizes less or equal to k.

Definition 3. *The arc/constraint* (X, Y) *connecting the variables* X *and* Y *is* k-bounds-consistent, *iff* (X, Y) *is bounds-consistent* or $|D_X| > k$.

Example 3. As in Example 2, let X, Y be constrained variables with the corresponding domains $D_X = \{5, 6, 8\}$ and $D_Y = \{1, 2, 3\}$, and it holds $X = Y + 5$. The constraint is *not* 5-bounds-consistent, because $|D_X| \leq 5$ and there is not any support in D_Y for $5 \in D_X$. However, we do have 2-bounds-consistency, as $|D_X| > 2$.

Lemma 3. k-bounds-consistency enforcement on an arc (X, Y) requires at most $O(kd)$ steps.

Proof: Following the Lemma 2 proof, it takes $O(|D_X| \cdot |D_Y|)$ time to enforce bounds-consistency. Nevertheless, remember that k-bounds-consistency is enforced only when $|D_X| \leq k$. As a consequence, k-bounds-consistency has a $O(k \cdot d)$ worst case cost. ■

As k grows and approaches infinity, which is as a matter of fact equivalent to d, k-bounds-consistency becomes evidently identical to bounds-consistency. For low k values k-bounds-consistency approximates a simple constraint check.

Maintaining 1-Bounds-Consistency during search is identical with a plain backtracking method, and no constraint propagation is done. In this case 1-bounds-consistency degenerates into a way to check if a constraint is satisfied: We search to find a support value for the unique bound/ value of X; if no support is found, the unique value is removed, and an ultimate inconsistency signal is broadcast. This particular consistency type may also appear in 'lazy' propagation schemas, e.g. in local search contexts [30].

VI. Empirical Results

To verify the algebraic formulations, we had to state the problem in our generic CSP solver. All the source code is freely available at http://di.uoa.gr/~pothitos/ictai2012 with the fourteen ITC-2007/08 real-world datasets included too. The experiments were conducted on an HP computer with an Intel dual-core E6750 processor at 2.66 GHz and 2 GB of memory, running Ubuntu Linux 8.04. In accordance with ITC standards, we have only 334 seconds in this machine in order to find a solution.

A. Fine-Tuning Consistency Levels in Practice

The lightweight consistency proposed seems in theory to ease the burden of the necessary revisions. But is it competitive in demanding problems such as real-life course timetabling, in relation to other consistency levels?

Figure 3 illustrates the time it took to construct a solution and improve it as much as possible, for each of the fourteen problem instances. It is obvious that for each one of them there is a specific k, for which the Maintaining k-Bounds Consistency methodology gives the best results. For $k = 1$, the methodology actually uses no constraint propagation; it is a plain backtracking method, so the results are poor. On the other hand, while k approximates infinity, i.e. while k-bounds-consistency approaches plain bounds-consistency, the results are not so poor, but are apparently worse than using the 'golden mean' k value, which usually lies around 25.

Figure 4 displays the corresponding costs of the solutions found for each one of the fourteen datasets. Conclusively, for very small k values we not only found low quality solutions, i.e. with high cost, but we consumed a lot of time to find them. On the contrary, as k increases above the 'golden mean,' the solution quality remains almost the same, but, again, as we saw in Fig. 3, we need more time to reach it. Every methodology we applied spent all the available time, *but* constructed its own best solution at different time, as shown in Fig. 3.

B. Employing Several Search Methods: Comparisons

Finally, it is time to exploit the many different generic search methods available together with our CSP solver, in a library called AMORGOS. In Figures 5 and 6, we see how the solution costs are improved during the available time. We present the results for the first two ITC dataset instances, with the corresponding names '`Fis0506-1`' and '`Ing0203-2`.'

Except for the *Depth-bounded Backtrack Search* (DBS) method [27], we also utilized *Iterative Broadening* [31], as well as the classic *Depth First Search* (DFS), along with its stochastic variations that used the random heuristics in Section IV-A2, for different $rand$ values.

First of all, as $rand$ increases, stochastic DFS may give better results than normal DFS, and this is achieved by our

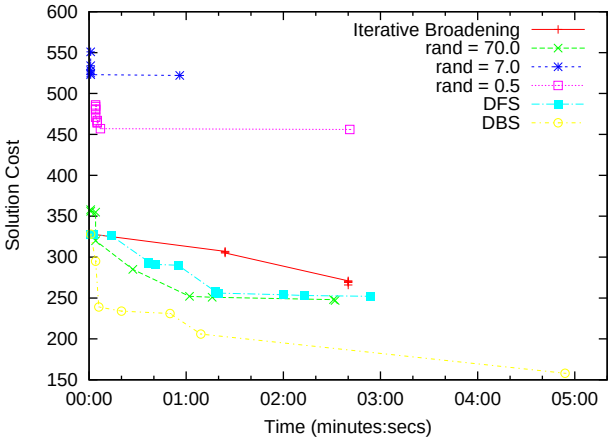

Figure 5. Objective optimization progress for the *first* ITC instance

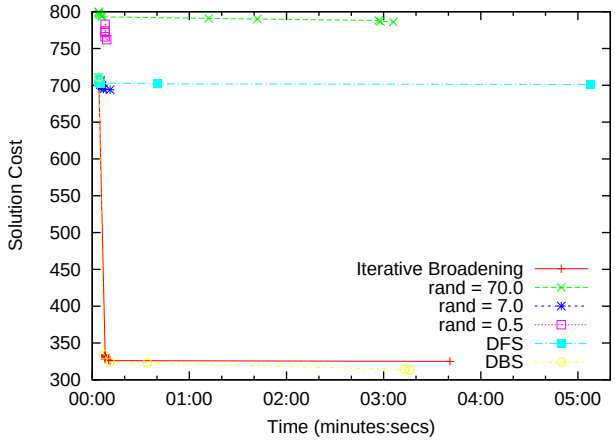

Figure 6. Objective optimization during time for the *second* instance

semi-random heuristics. For small $rand$ values, like 0.5, i.e. when we have more randomness, the solutions are regularly worse. In any case, DBS seems to outperform the rest of the methods.

Note that the above results for the two ITC instances were generated while maintaining 25-bounds-consistency during search. Besides, 25 appeared to be the best k value in many instances.

VII. Conclusions and Perspectives

Constraint Programming is a tried paradigm to solve optimization problems. We presented an effective application to the timetabling problem. The CSP formulation allowed us to exploit many generic search techniques, as the statement of the problem and the search of the solution are independent steps in the philosophy of Constraint Programming. So, one can even further experiment with other, modern or old, stochastic or systematic search methods and combine them. The statement of the problem will remain the same, unless one wants to insert new constraints, or make it more generic, if possible.

Amongst the methodologies we used, we focused on parameterizing bounds-consistency. The k-bounds-consistency was proved more efficient for small k values. Still, the exact specification of the best k is different for each different CSP instance. In future, it would be interesting to automate the process of finding the 'golden mean' k. Another tempting question is: When it is better to exploit arc-consistency and to what extend?

Future perspectives also include proposing even looser consistency types for the individual problems with too many variables, for which only local search methods seem nowadays efficient [32]. Finally, except for the domain size, are there any other ways to limit—or augment—the constraint propagation level?

Acknowledgments

Nikolaos Pothitos is financially supported by a Bodossaki Foundation Ph.D. scholarship.

This research was partially funded by the University of Athens Special Account of Research Grants no 10812.

We would also like to thank Foivos Theocharis for the implementation of the programming library Amorgos which supports the search methods we used and is freely available together with Naxos solver [25]. Amorgos is an extension to Naxos, that kept the convention to label our Constraint Programming Libraries using Greek islands names.

References

[1] P. Mellor, "A review of job shop scheduling," *OR*, vol. 17, no. 2, pp. 161–171, 1966.

[2] M. L. Pinedo, *Scheduling: Theory, Algorithms, and Systems*, 4th ed. New York: Springer, 2012.

[3] E. K. Burke and S. Petrovic, "Recent research directions in automated timetabling," *European Journal of Operational Research*, vol. 140, no. 2, pp. 266–280, 2002.

[4] Ö. Ülker, E. Özcan, and E. Korkmaz, "Linear linkage encoding in grouping problems: Applications on graph coloring and timetabling," in *PATAT 2006: 6th International Conference on the Practice and Theory of Automated Timetabling, Brno, Czech Republic*, E. K. Burke and H. Rudová, Eds., vol. 3867. Heidelberg: Springer, 2007, pp. 347–363.

[5] J. van den Broek, C. Hurkens, and G. Woeginger, "Timetabling problems at the TU Eindhoven," *European Journal of Operational Research*, vol. 196, no. 3, pp. 877–885, 2009.

[6] S. Shatnawi, K. Al-Rababah, and B. Bani-Ismail, "Applying a novel clustering technique based on FP-tree to university timetabling problem: A case study," in *ICCES 2010: 6th International Conference on Computer Engineering and Systems*, 2010, pp. 314–319.

[7] E. K. Burke, B. MacCarthy, S. Petrovic, and R. Qu, "Multiple-retrieval case-based reasoning for course timetabling problems," *JORS: Journal of the Operational Research Society*, vol. 57, no. 2, pp. 148–162, 2005.

[8] A. Gunawan, K. M. Ng, and K. L. Poh, "A hybridized Lagrangian relaxation and simulated annealing method for the course timetabling problem," *Computers & Operations Research*, vol. 39, no. 12, pp. 3074–3088, 2012.

[9] S. Yang and S. N. Jat, "Genetic algorithms with guided and local search strategies for university course timetabling," *IEEE Trans. Syst., Man, Cybern. C*, vol. 41, no. 1, pp. 93–106, 2011.

[10] Z. Lü and J.-K. Hao, "Adaptive tabu search for course timetabling," *European Journal of Operational Research*, vol. 200, no. 1, pp. 235–244, 2010.

[11] T. Müller, "ITC-2007 solver description: A hybrid approach," *Annals of Operations Research*, vol. 172, no. 1, pp. 429–446, 2009.

[12] H. Frangouli, V. Harmandas, and P. Stamatopoulos, "UTSE: Construction of optimum timetables for university courses – A CLP based approach," in *PAP 1995: 3rd International Conference on the Practical Applications of Prolog, Paris*. Alinmead Software Ltd, 1995, pp. 225–243.

[13] P. Stamatopoulos, E. Viglas, and S. Karaboyas, "Nearly optimum timetable construction through CLP and intelligent search," *International Journal on Artificial Intelligence Tools*, vol. 7, no. 4, pp. 415–442, 1998.

[14] K. Zervoudakis and P. Stamatopoulos, "A generic object-oriented constraint based model for university course timetabling," in *PATAT 2000: 3rd International Conference on the Practice and Theory of Automated Timetabling, Konstanz, Germany*, ser. LNCS, E. K. Burke and W. Erben, Eds., vol. 2079. Heidelberg: Springer, 2001, pp. 28–47.

[15] A. Schaerf and L. Di Gaspero, "Measurability and reproducibility in university timetabling research: Discussion and proposals," in *PATAT 2006: 6th International Conference on the Practice and Theory of Automated Timetabling, Brno*, E. K. Burke and H. Rudová, Eds., vol. 3867. Heidelberg: Springer, 2007, pp. 40–49.

[16] E. C. Freuder, "In pursuit of the holy grail," *ACM Computing Surveys*, vol. 28, no. 4es, p. 63, 1996.

[17] "ILOG SOLVER," http://ilog.com/products/cp, 2010.

[18] "ECL^{i}PS^e constraint programming system," http://eclipseclp.org, 2012.

[19] C. Bessière, J.-C. Régin, R. H. C. Yap, and Y. Zhang, "An optimal coarse-grained arc consistency algorithm," *Artificial Intelligence*, vol. 165, no. 2, pp. 165–185, 2005.

[20] C. Schulte and M. Carlsson, "Finite domain constraint programming systems," in *Handbook of Constraint Programming*, ser. Foundations of Artificial Intelligence, F. Rossi, P. van Beek, and T. Walsh, Eds. Amsterdam: Elsevier Science, 2006, ch. 14, pp. 495–526.

[21] E. C. Freuder and R. Wallace, "Selective relaxation for constraint satisfaction problems," in *ICTAI 1991: 3rd International Conference on Tools for Artificial Intelligence, San Jose CA*. IEEE, 1991, pp. 332–339.

[22] E. Tsang, "A glimpse of constraint satisfaction," *Artificial Intelligence Review*, vol. 13, no. 3, pp. 215–227, 1999.

[23] B. McCollum, A. Schaerf, B. Paechter, P. McMullan, R. Lewis, A. J. Parkes, L. D. Gaspero, R. Qu, and E. K. Burke, "Setting the research agenda in automated timetabling: The second international timetabling competition," *INFORMS Journal on Computing*, vol. 22, no. 1, pp. 120–130, 2010.

[24] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. New Jersey: Prentice Hall, 2009, ch. 6, pp. 202–233.

[25] N. Pothitos, "NAXOS SOLVER," http://di.uoa.gr/~pothitos/naxos, 2012.

[26] P. Prosser and C. Unsworth, "Limited discrepancy search revisited," *Journal of Experimental Algorithmics*, vol. 16, no. 1, pp. 1.6:1–1.6:18, 2011.

[27] R. Barták, "Incomplete depth-first search techniques: A short survey," in *CPDC 2004: 6th Workshop on Constraint Programming for Decision and Control, Gliwice, Poland*, J. Figwer, Ed., 2004, pp. 7–14.

[28] D. Sabin and E. C. Freuder, "Contradicting conventional wisdom in constraint satisfaction," in *PPCP 1994: 2nd International Workshop on Principles and Practice of Constraint Programming, Washington*, ser. LNCS, A. Borning, Ed., vol. 874. Heidelberg: Springer, 1994, pp. 10–20.

[29] C. Bessière, "Constraint propagation," in *Handbook of Constraint Programming*, ser. Foundations of Artificial Intelligence, F. Rossi, P. van Beek, and T. Walsh, Eds. Amsterdam: Elsevier Science, 2006, ch. 3, pp. 29–83.

[30] N. Pothitos, G. Kastrinis, and P. Stamatopoulos, "Constraint propagation as the core of local search," in *SETN 2012: 7th Hellenic Conference on Artificial Intelligence, Lamia, Greece*, ser. LNCS (LNAI), I. Maglogiannis, V. P. Plagianakos, and I. P. Vlahavas, Eds., vol. 7297. Heidelberg: Springer, 2012, pp. 9–16.

[31] M. L. Ginsberg and W. D. Harvey, "Iterative broadening," *Artificial Intelligence*, vol. 55, no. 2-3, pp. 367–383, 1992.

[32] P. Van Hentenryck and L. Michel, *Constraint-Based Local Search*, 2nd ed., ser. Logic Programming. Cambridge: The MIT Press, 2009.