A CP Approach to the Balanced Academic Curriculum Problem

Jean-Noël Monette, Pierre Schaus, Stéphane Zampelli, Yves Deville and Pierre Dupont
Department of Computing Sciences and Engineering
Université catholique de Louvain, Belgium
{jmonette,pschaus,sz,yde,pdupont}@info.ucl.ac.be

Abstract

The Balanced Academic Curriculum Problem (BACP) has received little attention in Constraint Programming. Only a few articles deals with this problem with experimental results on the three small instances publicly available in CSPLIB. The present article describes an approach to efficiently solve this challenging problem. Optimal solutions are produced on a variety of randomly generated instances which generalize the CSPLIB test cases. This work describes four contributions to the resolution of this problem: a new branching heuristic, the use of dominance relations, experiments on several balance criteria and several search strategies among which an hybridization of Constraint Programming and Local Search.

1 Introduction

The Balanced Academic Curriculum Problem (BACP) is recurrent in Universities. The goal is to schedule the courses that a student must follow in order to respect the prerequisite constraints between courses and to balance as much as possible the workload of each period. This problem has been introduced first in [Castro and Manzano, 2001] and tackled also in [Hnich *et al.*, 2002]. It has also been studied in [Hnich *et al.*, 2004] with an hybrid CP/ILP approach. More recently, [Lambert *et al.*, 2006] proposed another hybrid approach of Constraint Programming and Genetic Algorithms. Those works deal with the three small instances available on CSPLIB (http://www.csplib.org).

This work presents four contributions to address the BACP with Constraint Programming. First, we present a new value ordering heuristic guiding the search toward a balanced solution. Then, we propose to use dominance relations in order to reduce the search tree. Next, other balance criteria such as the minimization of the sum of deviations and square deviations are applied. Fourthly, different search schemes are compared, including an iterative increase of the objective value and the use of Local Search to find quickly a tight upper bound for Branch-and-Bound. Moreover, we present an instance generator that allows one to run experiments on larger instances with a structure similar to some original ones.

This paper is structured as follows. The next section presents the problem formally and describes the CP model. It describes also a new branching heuristic and the balance criteria that will be compared. Section 3 presents the dominance rules and their application. Section 4 presents the different search strategies used and Section 5 explains the instances generator. Finally, Section 6 presents the experiments and their results before concluding.

2 The BAC Problem

The goal of the BACP is to schedule courses in different periods. Each course has a workload which is expressed in number of credits and a (possibly empty) set of prerequisite courses. A solution is an assignment of courses to periods that satisfies the prerequisite constraints while balancing the workload of periods. The workload of a period is the sum of the credits of the courses taught during this period. Additional constraints limiting the minimum and maximum number of courses and credits for each period were originally introduced in [Castro and Manzano, 2001]. These additional constraints are however not considered here since balanced solutions of the instances in CSPLIB always respect them. Indeed, the limits on the number of credits by period are naturally respected by balanced solutions, unless the problem is unfeasible. The limits on the number of courses by period are respected because the number of credits for each courses does not vary much and these limits are quite large in the instances of CSPLIB. Precisely, a BACP instance is characterized by:

- n the number of courses;
- m the number of periods;
- w_i the load of course i for $1 \leq i \leq n$;
- $\mathit{prerequisites} = \{(i,j) \mid i \neq j, 1 \leq i,j \leq n\}$ a set of couples of courses stating that course i is a prerequisite of course j.

Three instances were originally proposed for this problem in CSPLIB with followings characteristics:

Instance	m	n	$\lvert prerequisites\rvert$	Values for w_i
1	8	46	38	$[1, 5]$
2	10	42	34	$[1, 5]$
3	12	66	65	$[1, 5]$

The BACP is interesting because it is at the boundary of several classes of problems: bin-packing, scheduling and balancing. The bin-packing is a class of problems where a set of objects of different sizes must hold in the smallest set of bags of finite capacity. BACP is a kind of bin-packing problem where the size of the bags is minimized rather than the number of bags. The prerequisite constraints make BACP look also like a scheduling problem. Each course is a unit-time activity whose credit is the consumption of resource and prerequisites are temporal constraints. The periods are time units and the goal is to balance the utilization of the unique resource (the student). Finally, BACP is also an excellent example of balancing problems. The need and the interest for balanced solutions increases in Constraint Programming. Maximization of the satisfaction of a set of customers, minimization of the violations in over-constrained problems, schedule of physicians, share of a scarce resource are some examples where balanced solutions are generally preferred.

BACP is also a hard problem. The satisfaction version of BACP ("Does it exist a solution under a given balance value?") is a NP-complete problem. This can be shown by reduction from the satisfaction version of the bin-packing problem ("Does it exist a solution with at most a given number of bags?") that is known to be NP-complete. Objects become courses, credits are sizes and bins are periods. There is no prerequisite. Looking for a solution for BACP solves the corresponding bin-packing problem. As shown in Section 5, the problem becomes easier with the addition of prerequisites. Indeed the number of available periods for each course is reduced.

2.1 CP Model

The CP model has initially been proposed in [Hnich *et al.*, 2002]. Each course is identified by an integer in the interval $[1, n]$. There are three sets of variables:

- $\forall i \in [1, n], P_i$ represents the period of course i.
- $\forall i \in [1, m], L_i$ is the load of the period.
- $\forall (i, j) \in [1, n] \times [1, m], B_{ij}$ makes the link between the two other sets of variables. In particular $B_{ij} = 1$ if and only if course i is given in period j.

The prerequisites constraints are easily stated with the first set of variables using "less than" constraints. The load variables are linked to the binary variables with weighted sums while channeling constraints link the period of the courses and the binary variables.

$$\forall (i, j) \in prerequisites : P_i < P_j.$$

$$\forall 1 \leq j \leq m : L_j = \sum_{i=1}^{n} B_{ij}.w_i.$$

$$\forall 1 \leq i \leq n, 1 \leq j \leq m : (P_i = j) \Leftrightarrow (B_{ij} = 1).$$

2.2 The Balance Criteria

In previous works [Castro and Manzano, 2001; Hnich *et al.*, 2002], the only balance criterion used for this problem is the minimization of the maximum load that will be denoted by $C^{\max}$. Mathematically, it is defined as

$$C^{\max} = \max_{1 \leq i \leq m} L_i.$$

The objective of the BACP is to minimize $C^{\max}$.

As shown in [Schaus *et al.*, 2007], other criteria can be used. The balance can be defined in a generic way by the criterion

$$C(p) = \sum_{i=1}^{m} |m.L_i - w|^p.$$

where $w = \sum_{i=1}^{m} L_i = \sum_{i=1}^{n} w_i$ is the total workload. The objective is now to minimize $C(p)$, i.e. to minimize the sum of the measures of the distance between the load of each period and the average load. In particular, instantiating the parameter p to 1, 2 and ∞ gives the following interpretations:

- $C(1) = \sum_{i=1}^{m} |m.L_i - w|$ is the sum of deviations from the mean.
- $C(2) = \sum_{i=1}^{m} (m.L_i - w)^2$ is the sum of square deviations from the mean.
- $C(\infty) = \max_{1 \leq i \leq m} |m.L_i - w|$ is the maximum deviation from the mean.

It as been shown in [Schaus *et al.*, 2007] that neither criterion subsumes the others and there is no a priori reason to prefer one of them. Section 6.1 evaluates how well each criterion approximates the others.

2.3 Variable and Value Ordering Heuristics

The previous work [Hnich *et al.*, 2002] on BACP branches on the variables P_i and uses a classical *first-fail* heuristic that chooses the unassigned variable with the smallest domain. The value heuristic picks the smallest value of its domain.

As the goal is to obtain the most balanced solution, choosing as value the period which is the less heavily loaded ensures that the first solution found will be already quite balanced. Formally, to post the constraint $P_i = v$, v is chosen such that

$$v = \underset{v' \in dom(P_i)}{\text{argmin}} \; \underline{L}_{v'}$$

where $dom(P_i)$ denotes the domain of the variable P_i and $\underline{L}_v$ is the minimum value in $dom(L_v)$.

3 Dominance Rules

The BACP instances contain many symmetrical solutions. For instances, consider two courses that have the same workload and that are prerequisites of the same courses and have the same courses as prerequisites. In a solution, these two courses can be exchanged giving another solution with the same balance (independently of the balance criterion used). Situations of this kind may be discovered by the use of dominance rules.

Dominance relations have been shown to be powerful for solving hard problems [Prestwich and Beck, 2004]. The concept of dominance can be seen as a symmetry that is unidirectional. A state s_i of the search tree is said to dominate another state s_j if the best solution that can be found from s_j is no better than the best solution that can be found from s_i. It is useless to explore the subtree rooted in s_j once s_i is explored. In other terms, suppose a property P such that for every optimal solution with $\mathcal{P}$ there exists an optimal solution with $\neg\mathcal{P}$. The solutions where $\neg\mathcal{P}$ holds dominate the solutions where $\mathcal{P}$ holds. In this case, posting the constraint $\neg\mathcal{P}$ preserves optimality. We refer to [Prestwich and Beck, 2004] for a formal definition of dominance.

3.1 Detecting Dominance Relations

In the context of the BACP, the dominance rules take the form of "less or equal" constraints between the periods of two courses. The conditions to post such a constraint are presented in the following theorem.

Theorem 3.1 *Posting a constraint $P_i \leq P_j$ preserves at least one optimally balanced solution to BACP if the following conditions hold :*

- $w_i = w_j$
- $pred_i \subseteq pred_j$
- $succ_i \supseteq succ_j$

where $pred_i = \{k|(k,i) \in prerequisites\}$ and $succ_i = \{k|(i,k) \in prerequisites\}$ denote respectively the set of prerequisite courses (predecessors) of i and the set of courses for which i is a prerequisite (successors of i).

Proof The first condition ensures that if there exists an assignment for which $P_i = a$ and $P_j = b$, then swapping the periods of i and j gives an equally good assignment. Indeed, whatever the objective, it is based on the workload of the periods that is not changed by swapping two courses with the same credit.

Let us denote P_{p_i} the period of the latest predecessor of i and P_{s_i} the period of the earliest successor of i. If $pred_i \subseteq pred_j$, the latest predecessor of j (whose assigned period is denoted P_{p_j}) cannot be earlier than the latest predecessor of i (whose period is denoted P_{p_i}). Either it is the same course or it is another one that is thus necessarily later. Conversely, $succ_i \supseteq succ_j$ means that the earliest successor of i (P_{s_i}) cannot be later than the earliest successor of j (P_{s_i}). The following inequalities hold in any solution :

- $P_{p_i} < P_i < P_{s_i}$
- $P_{p_j} < P_j < P_{s_j}$
- $P_{p_i} \leq P_{p_j}$
- $P_{s_i} \leq P_{s_j}$

In an optimal solution, either $P_i \leq P_j$ or $P_i > P_j$. To preserve at least one optimal solution when adding the constraint $P_i \leq P_j$, for any solution where $P_i > P_j$ holds, there must exist another optimal solution where $P_i \leq P_j$ holds.

Let us suppose an optimal solution with $P_i = a$, $P_j = b$ and $b < a$. Swapping the values of P_i and P_j (leading to $P_i = b$ and $P_j = a$) keeps the assignment optimal and enforces $P_i \leq P_j$. It remains to show that this assignment is still a solution and verifies the prerequisite constraints. Let us show that a is a valid value for P_j, that is that $P_{p_j} < a < P_{s_j}$. The relations $P_{p_i} < a < P_{s_i}$, $P_{p_j} < b < P_{s_j}$, $b < a$ and $P_{s_i} \leq P_{s_j}$ hold. Using the transitivity of $<$ and $\leq$, it results in $P_{p_j} < b < a < P_{s_i} \leq P_{s_j} \Rightarrow P_{p_j} < a < P_{s_j}$. The same reasoning can be done to show that b is a valid value for P_i. Thus enforcing $P_i \leq P_j$ preserves at least one optimally balanced solution. ∎

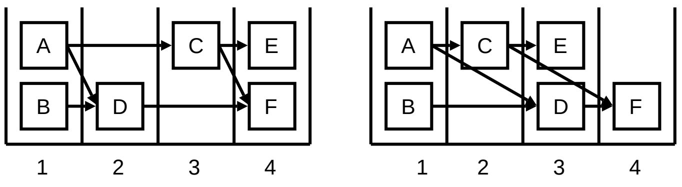

Figure 1: Example of dominance application.

Figure 1 presents two situations that induce dominance rules. Squares represent courses with unit credit. Arrows represent the prerequisites between courses. Vertical bins are the periods. The two situations are parts of some solutions to a BACP instance. In the first situation, course C is after course D but they can be swapped without changing the value of the solution. In the second situation, course C is before course D but they cannot be swapped because it would not be a solution anymore (C would not be taught before course E). So the constraint $P_C \leq P_D$ can be posted.

As noted in [Prestwich and Beck, 2004], one has to carefully check that the use of several interacting dominance rules does not suppress all of the optimal solutions. Problems may arise only when two or more courses of equal credit have exactly the same prerequisites and are prerequisite of exactly the same courses. In such a case, there are real symmetric states and the "less or equal" constraint can be posted in either way but only one can be posted. When more than two courses have the same characteristics, the constraints must be posted without creating cycles. For instance, a lexicographic ordering can be used and the constraint $P_i \leq P_j$ is posted only if the additional condition $i < j$ holds.

The detection of dominance relations is performed once per instance at the root of the search tree with a temporal complexity in $\mathcal{O}(n^2p)$, where p is the maximum number of predecessors or successors of a course ($p \ll n$). Indeed, every pair of courses is considered and for each, if they have the same workload, it is necessary to compare the sets of predecessors and successors of the courses, which is linear in the size of these sets.

The number of dominance relations found may be rather important. For the three original instances in CSPLIB, there are respectively 134, 65 and 183 relations for the instance with respectively 8, 10 and 12 periods.

3.2 Applying Dominance Rules

Dominance rules can be used to improve the search. They can simply be posted at the root node of the search tree. However, this method has the drawback (observed also in symmetry breaking) that the first solution found might become inconsistent with the additional constraints. This would possibly force the search process to explore a large part of the search tree not explored otherwise.

Several techniques have been developed to avoid this undesirable phenomenon when dealing with symmetry breaking (see [Gent *et al.*, 2006] for an overview). We choose to reuse the idea of SBDS (Symmetry Breaking During Search) [Gent and Smith, 2003], applying the technique to dominance rules. Basically, SBDS posts constraints after backtracking to avoid to explore search states symmetrical to ones already explored. SBDS has been chosen instead of other dynamic techniques because it seemed the most adapted to our purpose and was easy to implement.

The SBDS technique cannot be applied as such however for dominance rules that are unidirectional. In the context of the dominance rules considered here, a dominance constraint is posted in the right branch of the binary search tree if this constraint is verified in the left branch. Suppose there exists the possible dominance rule $P_i \leq P_j$ and the decision constraint $P_i = a$ is posted during the search. After backtracking, when the opposite constraint ($P_i \neq a$) is posted, the dominance rule $P_i \leq P_j$ is added to the current state if $a < \underline{P}_j$ holds. This dynamic technique removes states dominated by other states already explored but not by states that could be explored later during the search.

4 Search Strategies

4.1 Branch-and-Bound

This section presents the different search strategies that can be used to solve the BACP. The first and most known technique is Branch-and-Bound. It consists in solving the optimization problem to minimize C (where C is the variable representing the criterion to optimize) as a satisfaction problem where the constraint $C < v$ is added. The value v is initially set to some upper bound (*e.g.* $+\infty$) and is changed to the value of C whenever a new solution is found. In this way, successive solutions improve the value of the criterion and when no more solution can be found, the last solution is proved to be an optimal one.

4.2 Including Local Search in Branch-and-Bound

Local Search (LS) can find rapidly a tight upper bound on the value of the objective to start Branch-and-Bound. We refer to [Hoos and Stützle, 2004] for extensive explanations about LS. The LS algorithm provides a feasible solution that is not necessarily optimal but that should be as close as possible to the optimum. A constraint-based LS approach [Hentenryck and Michel, 2005] is used to model the problem. The LS model is the same as the CP model.

A tabu search heuristic is used. The satisfaction of the prerequisites and the optimization of the balance are combined in an objective function. In order to drive the search toward feasible solutions, the constraint satisfaction term has a much larger weight than the optimization term. A local move consists in choosing the variable P_i and the value allowing the best improvement of the objective function and reassigning the variable to this value. The search is stopped after a fixed number of iterations or when a feasible solution that is known to be optimal is found (perfect balance).

4.3 Iterative Satisfaction Problem

An alternative to Branch-and-Bound is to solve successive satisfaction problems with the constraint $C < v$, incrementing the value of v until a solution is found. The first obtained solution is optimal as the CSPs with smaller values are inconsistency. This technique has been used successfully in Scheduling [Baptiste *et al.*, 2001].

5 Instances Generator

Only three instances of BACP are available in CSPLIB. Moreover they are easy to solve. Therefore a parametrized instances generator has been created. Parameters are fixed to obtain additional instances with a similar structure to the three existing ones as detailed below.

5.1 The Generator

The generator creates instances parametrized by the number of courses, the number of periods, the minimum and maximum possible credits for a course and a probability to have a prerequisite between two courses. Instances are generated according to the following scheme:

1. For each course, a random credit is chosen between the minimum and maximum values.
2. Each course is assigned to a random period.
3. For each pair of courses that are in two successive periods, a prerequisite relation is created according to the given probability.

The instance generation ensures that a solution always exists. The choice of prerequisites only between courses in successive periods excludes some configuration of prerequisite graphs but still offers a wide range of possibilities. The generated solution is usually not optimal and in an optimal solution, prerequisites will not necessary be between courses of successive periods.

5.2 Generated Problem Sets

To obtain instances similar to the original ones in CSPLIB, the ratio between the number of courses and the number of periods is fixed to 5. This is the mean ratio among the three original instances (see Section 2).

Preliminary experiments on instances with 20 periods and 100 courses whose credits range from 3 to 5 enabled us to study how the probability of prerequisites affects the difficulty of the problem. We generated 20 instances for each probability from 0% to 50% by step of 5%. The number of solved instances in less than 30 seconds with $C^{\max}$ and Branch-and-Bound search was considered. These preliminary runs showed that instances are globally easier when they include more prerequisites. Especially, the problems stay hard from 0% to 25% of prerequisites (about 75% of unsolved cases). After this 25% limit, the difficulty of the problem drops rapidly. At 50% of prerequisites all instances are solved.

Two sets of test instances have been generated. The first set is composed of 100 small instances with 5 periods and 25 courses. The probability for prerequisites is set to 30% and the range of credits is $[1, 5]$. This simple set has been generated to compare the different balance criteria.

The second problem set includes instances with a number of courses varying from 30 to 200 with an increment of 10. The probability to have a prerequisite is fixed to 20%. Instances with different ranges of credit are generated to study the influence of this parameter. There are four classes:

- The range $[1, 5]$ corresponds to the instances in CSPLIB.
- A unit credit for every courses induces easier instances.
- The range $[1, 10]$ is an example of larger interval for which instances are harder.
- The range $[3, 5]$ forbids courses with small credits that can be used to easily fill holes in period not loaded enough.

Generating 10 instances for each configuration (4 classes and 18 sizes), the second set includes a total of 720 instances.

6 Experiments

The aim of this section is to answer the following questions:

- How well does each balance criterion approximate the others?
- How much does the proposed branching heuristic reduce the search space as compared to the one proposed in [Hnich *et al.*, 2002]?
- Which search strategy is more time efficient to solve the BACP?
- How much dominance rules improve search efficiency?

	$C^{\max}$	$C(1)$	$C(2)$	$C(\infty)$	Average
$C^{\max}$	0.00	10.62	16.53	0.06	9.07
$C(1)$	2.63	0.00	6.27	0.12	3.00
$C(2)$	0.28	0.00	0.00	0.00	0.09
$C(\infty)$	10.37	18.07	23.66	0.00	17.36
Average	4.43	9.56	15.48	0.06	

Table 1: Comparison of four balance criteria. Each row corresponds to an optimized criterion. Each column corresponds to an evaluated criterion.

All experiments are performed on an Intel® Celeron® 2.8 GHz with 1GB of memory. Our implementation uses the Gecode (http://www.gecode.org) constraints library and the Local Search is written in the Comet (http://www.comet-online.org) programming language.

6.1 Balance Criteria

The goal of this experiment is to compare the four considered criteria ($C^{\max}$, $C(1)$, $C(2)$ and $C(\infty)$). The focus is on the quality of the solution in terms of balancing and thus time is not considered.

The 100 instances of the first problem set are successively solved with the four criteria and the best solutions found are evaluated with respect to the four criteria. The search is performed using Branch-and-Bound with the new branching heuristic.

There exists a global constraint for $C(1)$ running in $\mathcal{O}(m)$ introduced in [Schaus *et al.*, 2007] and a less efficient one ($\mathcal{O}(m^2)$) for $C(2)$ presented in [Pesant and Régin, 2005; Schaus *et al.*, 2006]. $C^{\max}$ and $C(\infty)$ use classical constraints whose temporal complexity is $\mathcal{O}(m)$. The total runs took less than 2 seconds for each criterion except for $C(2)$ for which it took 14 seconds.

The result is presented in Table 1 covering four possible optimizations and evaluations. Each row represents an optimized criterion and each column an evaluated criterion. Each table entry reports the average relative difference in percent between the evaluated criterion and the optimum value for that criterion over the 100 instances. Naturally, the values on the diagonal are zero as the instances are solved to the optimum. For example, the value 2.63 at the intersection of row $C(1)$ and column $C^{\max}$ means that the best solution found while optimizing $C(1)$ presents on average a $C^{\max}$ value 2.63% larger than the optimal value for $C^{\max}$.

On these instances, Table 1 shows that $C(2)$ is the criterion that approximates best the other criteria, as a solution found when $C(2)$ is optimized is also often optimal with respect to the other criteria. The criterion $C(\infty)$ appears to be the worse criterion to approximate the three others. Conversely, $C(2)$ is hardly approximated by the others criteria while C^{∞} is quite well approximated by others. Criteria $C(1)$ and $C^{\max}$ lie between these two extremes with $C(1)$ being a better approximation.

Instance	Nb Leaf Nodes		Nb Int. Sol.	
	first-fail	min-load	first-fail	min-load
8	1172	98	69	3
10	3191	722	58	13
12	30147	2975	87	12

Table 2: First-fail and proposed ("min-load") heuristics on the original instances with $C(1)$

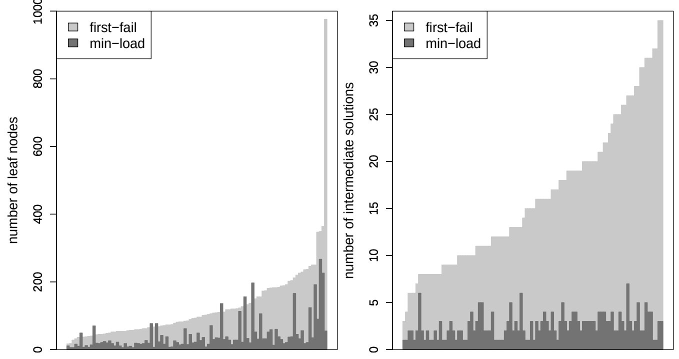

Figure 2: Comparison of heuristics on 100 instances. Left part : Number of leaves. Right part : Number of intermediate solutions.

6.2 Efficiency of the Proposed Branching Heuristic

To analyze the effect of the new value ordering heuristic, it is compared with the *first-fail* heuristic from [Hnich *et al.*, 2002] using the Branch-and-Bound search and with the minimization of $C(1)$ (the minimization with the other criteria gives similar results and are not reported). First the evaluation is achieved on the original instances used in [Hnich *et al.*, 2002] and on the 100 instances of the first generated problem set. The results are presented in Table 2 and Figure 2. The comparison is based on the number of leaf nodes and the number of intermediate solutions found during the Branch-and-Bound search. The left part of Figure 2 presents the number of leaf nodes with both heuristics, sorted by the values obtained with the basic heuristic. The right part presents the results in a similar way for the number of intermediate solutions.

It is clear that the new ("min-load") branching heuristic improves the search except for 7 out of 100 instances (left part of Figure 2). As expected, the search is rapidly guided toward balanced solutions. Indeed, it can be seen on the right part of Figure 2 that the number of intermediate solutions is always smaller with the new branching heuristic. The same conclusions hold on the original instances in Table 2.

$C^{\max}$	-	D	DDS
BB	76.4	45.8	83.3
HY	96.5	94.0	95.7
IT	85.8	56.5	85.9
$C(1)$	-	D	DDS
BB	38.9	18.9	45.0
HY	76.5	70.4	76.9
IT	79.7	53.6	80.7

Table 3: Percentage of solved instances (on 720).

6.3 Comparing Search and Dominance Breaking Strategies

This section analyzes both the search strategies and the dominance breaking strategies. This study is performed on the second problem set using the new branching heuristic and with two different criteria, $C^{\max}$ and $C(1)$. We restrict our attention to those two because the first is the most commonly used and the second is a good and fast approximation to the others according to the results of Section 6.1.

There are 3 search strategies (Section 4) referred to as "BB" for Branch-and-Bound, "HY" for the hybrid approach and "IT" for the iterative approach. There are 3 dominance rules strategies (Section 3.2) that are respectively not to apply them ("-"), to post them at the beginning ("D" standing for Dominance), and to use the adapted SBDS ("DDS" for Dominance During Search). This leads to 9 possible combinations.

The percentage of solved instances (on the 720 instances of the second test set) with each setting within the time limit of 30 seconds is presented in Table 3.

The hybrid search outperforms the two other search strategies with any dominance policy when minimizing $C^{\max}$. More than 96% of the instances can be solved with this technique. Among the two other techniques, the iterative one is overall better than Branch-and-Bound. On the other hand, when optimizing $C(1)$, the largest number of solved instances is obtained with the iterative strategy, followed by the hybrid search.

Concerning the different uses of dominance rules, posting them at the beginning of the search is a bad idea. It is seen with both criteria that column "D" is worse than the two others. On the contrary, the dynamic use of the dominance rules improves the search.

It is interesting to characterize the instances that are harder to solve. On the whole test set, only five instances are not solved within 30 seconds by any combination of the search and dominance strategies for the optimization of $C^{\max}$. They are all in the $[1, 10]$ class. 47 instances stay unsolved when using the criterion $C(1)$. Most of them (40) have credits ranging in the interval $[3, 5]$. As expected problems with credits in $[3, 5]$ and $[1, 10]$ are harder to solve.

Regarding the search strategies, there are respectively 93, 8 and 58 unsolved instances respectively for "BB", "HY" and "IT" with any dominance strategy and with

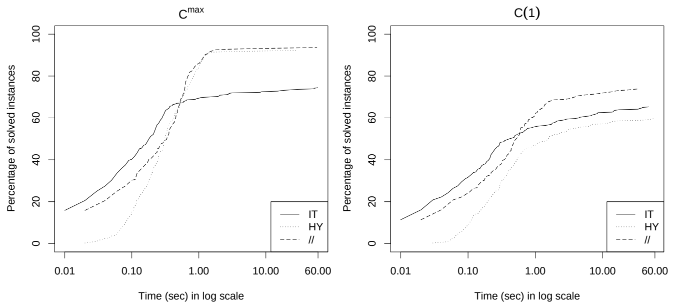

Figure 3: Percentage of solved instances in function of the allocated time.

$C^{\max}$. These values becomes 385, 104 and 84 when optimizing $C(1)$. These values show that some instances are only solved by one method. The effect of dominance rules is quite unpredictable. Adding dominance constraints increases the number of solved instances within the time constraint but it prevents the resolution of some instances. We have no clear understanding about which instances are easier to solve with and without the use of dominance rules.

Using the hybrid technique, the upper-bound found by LS is optimal in 93% of the solved instances with the criterion $C^{\max}$ and in 77% of the solved instances for the minimization of $C(1)$. LS performs well on this problem with $C^{\max}$ because there are many solutions in the search space and Local Search can easily reach one of them. The results are not as good with the criterion $C(1)$ because optimal solutions are sparser. The time spent in LS never exceeded one second and was less than 500 milliseconds most of the time. For easy instances, most of the work is done by LS to find an upper-bound while for harder instances, only a fraction of the time is spent in LS.

Figure 3 shows the percentage of solved instances among the harder classes ($[3,5]$ and $[1,10]$) in function of the allocated time. It compares three strategies for minimizing $C^{\max}$ (left part) and $C(1)$ (right part). The three strategies use dynamic dominance rules and are respectively "IT", "HY" and "//". This last strategy supposes that "IT" and "HY" are run in parallel (on a single processor) and that they are both stopped when one of them finds the solution. Thus the time allocated for each search is half the normal time.

When optimizing $C^{\max}$, the iterative technique solves small problems faster than the hybrid one but the hybrid technique outperforms the iterative search for more complex problems. The method "HY" solves most problems in less than 1.5 seconds and is unable to solve problems after this limit. On the contrary, "IT" still increases the number of solved instances when allocated time increases. The parallel search technique slightly improves the results in comparison to the hybrid technique. This improvement is more important when minimizing $C(1)$. In this case, the parallelism permits to solve 10% more instances when the allocated time is more than 1 second. Optimizing $C(1)$, "IT" is the technique that solves the most instances in less than 1 second. When the allocated time is more than 1 second, the parallel search becomes better because some hard instances with "IT" are solved with "HY". The three techniques continue to improve their result when the allocated time increases.

7 Conclusion

This article presents a CP approach to the Balanced Academic Curriculum Problem [Hnich *et al.*, 2002]. It explores several directions to improve the resolution of this problem and it introduces an instance generator and two benchmarks of instances that allows us to validate the proposed improvements.

The first proposition is to consider other balance criteria than the minimization of the maximum load ($C^{\max}$) used up to now. We propose to minimize the mean deviation ($C(1)$), the mean square deviation ($C(2)$) or the maximum deviation ($C(\infty)$) from the mean workload. The first experiment shows that minimizing the mean square deviation is a good approximation to the other criteria. But existing propagators for $C(2)$ have a higher complexity than for the others criteria. For this reason, we propose to use the criterion $C(1)$ that is the second best approximation to the other criteria. $C^{\max}$ can also be used because its minimization is far more efficient than the one of $C(1)$. A possible direction of future research is to combine these criteria. For instance, one could use $C^{\max}$ to find a first well balanced solution then improve the balance with another criterion like $C(1)$ or $C(2)$.

Our second contribution consists in using an *ad-hoc* branching heuristic that guides the search towards balanced solutions. Experiments show that this new heuristic outperforms the classical *first-fail* heuristic to reach optimal solutions and to prove their optimality.

Next we propose the use of dominance relations to reduce the search. Dominance relations can be thought as one-way symmetries. Dominance relations for BACP are presented. They can be found in $\mathcal{O}(n^2.p)$ (where n is the number of courses and p the maximum number of prerequisites). These relations can be posted at the start of the search or can be dynamically added during the search, in a way similar to SBDS for symmetry breaking. Experiments show that it is counterproductive to post the dominance rules at the start. For the dynamic use of the dominance rules, the effect is less clear. While for some instances it reduces the search size, for other it increases it. The understanding of this phenomenon is a wide subject of research as it is the case in symmetry breaking.

The last contribution presents two alternatives to the simple Branch-and-Bound scheme to find optimal solutions. The first one relies on Local Search to find a good initial upper bound to start the Branch-and-Bound. We illustrate experimentally that this hybrid technique is really efficient for the BACP because the Tabu search often finds optimal values. The second alternative is to start from below and iteratively increase the upper bound on the optimized criterion until finding a solution. This method performs well because it requires often few iterations before finding a solution. Experiments point out that the hybrid search is by far the best strategy when minimizing $C^{\max}$, while the iterative technique is a little better for the minimization of $C(1)$.

Although no technique is the panacea, there are only few instances that are not solved by at least one of the studied strategies. Consequently, we think that a portfolio of different search strategies could be the right choice to efficiently solve the BACP. We propose to use the hybrid and iterative search techniques with and without the dynamical use of dominance rules. The analysis of portfolio search techniques [Huberman *et al.*, 1997] for BACP is an interesting direction for future research.

Acknowledgment

The authors wish to thank the anonymous reviewers for their constructive comments. This research is supported by the Walloon Region, project TransMaze (516207).

References

[Baptiste *et al.*, 2001] Philippe Baptiste, Claude Le Pape, and Wim Nuijten. *Constraint-based Scheduling: Applying Constraint Programming to Scheduling Problems.* Kluwer Academic Publisher, 2001.

[Castro and Manzano, 2001] C. Castro and S. Manzano. Variable and value ordering when solving balanced academic curriculum problem. *Proc. of the ERCIM Working Group on Constraints*, 2001.

[Gent and Smith, 2003] Ian P. Gent and Barbara M. Smith. Symmetry breaking in constraint programming. *Proceedings of the Third International Workshop on symmetry in Constraint Satisfaction Problems*, pages 55–65, 2003.

[Gent *et al.*, 2006] Ian P. Gent, Karen E. Petrie, and Jean-François Puget. Symmetry in consraint programming. In Francesca Rossi, Peter Van Beek, and Toby Walsh, editors, *Handbook of constraint programming.* Elsevier, 2006.

[Hentenryck and Michel, 2005] Pascal Van Hentenryck and Laurent Michel. *Constraint-Based Local Search.* The MIT Press, 2005.

[Hnich *et al.*, 2002] Brahim Hnich, Zeynep Kiziltan, and Toby Walsh. Modelling a balanced academic curriculum problem. *Proceedings of CP-AI-OR-2002*, pages 121–131, 2002.

[Hnich *et al.*, 2004] Brahim Hnich, Zeynep Kiziltan, Ian Miguel, and Toby Walsh. Hybrid modelling for robust solving. *Annals of Operations Research 130*, pages 19–39, 2004.

[Hoos and Stützle, 2004] Holger H. Hoos and Thomas Stützle. *Stochastic Local Search: Foundations and Applications.* Morgan Kaufmann, 2004.

[Huberman *et al.*, 1997] Bernardo A. Huberman, Rajan M. Lukose, and Tad Hogg. An economics approach to hard computational problems. *Science*, 275:51–54, January 3 1997.

[Lambert *et al.*, 2006] Tony Lambert, Carlos Castro, Eric Monfroy, and Frédéric Saubion. Solving the balanced academic curriculum problem with an hybridization of genetic algorithm and constraint propagation. In *Proceedings of The International Conference on Artificial Intelligence and Soft Computing (ICAISC'06)*, volume 4029 of *Lecture Notes in Artificial Intelligence*, pages 410–419, Zakopane, Poland, 2006. Springer Verlag.

[Pesant and Régin, 2005] G. Pesant and J.C. Régin. Spread: A balancing constraint based on statistics. *Lecture Notes in Computer Science*, 3709:460–474, 2005.

[Prestwich and Beck, 2004] Steven Prestwich and J. Christopher Beck. Exploiting dominance in three symmetric problems. *Fourth International Workshop on Symmetry and Constraint Satisfaction Problems*, 2004.

[Schaus *et al.*, 2006] P. Schaus, Y. Deville, P. Dupont, and J.C. Régin. Simplification and extension of the spread constraint. *Third International Workshop on Constraint Propagation And Implementation*, pages 77–91, 2006.

[Schaus *et al.*, 2007] P. Schaus, Y. Deville, P. Dupont, and J.C. Régin. The deviation constraint. *4th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR'07)*, pages 260–274, 2007.