The Balanced Academic Curriculum Problem Revisited

Marco Chiarandini[1], Luca Di Gaspero[2], Stefano Gualandi[3], and Andrea Schaerf[2]

[1]IMADA, University of Southern Denmark, Campusvej 55, DK-5000, Odense, Denmark, Email: marco@imada.sdu.dk
[2]DIEGM, University of Udine, via delle Scienze 208, I-33100, Udine, Italy , Email: {l.digaspero,schaerf}@uniud.it
[3]DEI, Politecnico di Milano, via Ponzio 34/5, I-20133, Milan, Italy , Email: stefano.gualandi@polimi.it

December 29, 2009

Abstract

The Balanced Academic Curriculum Problem (BACP) consists in assigning courses to teaching terms satisfying prerequisites and balancing the credit course load within each term. The BACP is part of the CSPLib with three benchmark instances, but its formulation is simpler than the problem solved in practice by the universities. In this article, we introduce a generalized version of the problem that takes different curricula and professor preferences into account, and we provide a set of real-life problem instances arisen at University of Udine. Since the existing formulation based on a min–max objective function does not balance effectively the credit load for the new instances, we also propose alternative objective functions. Whereas all the CSPLib instances are efficiently solved with Integer Linear Programming (ILP) state-of-the-art solvers, our new set of real-life instances turns out to be much more challenging and still intractable for ILP solvers. Therefore, we have designed, implemented, and analyzed heuristics based on local search. We have solved all the new instances with the proposed approach and assessed the quality of solutions with respect to lower bounds found by ILP on a relaxed and decomposed problem. Results show that a selected heuristic finds solutions of very good quality at 9%-60% distance from the lower bound. We make all data publicly available, in order to stimulate further research on this problem.

Keywords Combinatorial optimization Metaheuristic Methodologies Timetabling Local Search

1 Introduction

The Balanced Academic Curriculum Problem (BACP) consists in assigning courses to teaching terms such that prerequisites are satisfied and the students course load is balanced. The BACP, introduced by Castro and Manzano (2001), has practical application in university planning, and it is part of the CSPLib (Gent and Walsh, 1999, prob. 30) with three benchmark instances. In the original formulation, the balance of the course load is achieved minimizing the maximum of the course load over all terms.

The BACP is tackled using Constraint Programming (CP) and Integer Linear Programming (ILP) techniques by Hnich et al. (2002) and Castro et al. (2007). Hybrid techniques composed of genetic algorithms and constraint propagation are used by Lambert et al. (2006). In their works, the authors are able to compute the optimal solution for all three instances present in the CSPLib, but exhibiting quite different running times.

Monette et al. (2007) report an extensive study on BACP. They introduce a random instance generator and experiment with 720 instances of size up to 200 courses and varying characteristics. Most importantly, they devote attention to the balance criteria, extending the original maximum load over the periods, used in all previous works (Hnich et al., 2002; Castro et al., 2007; Lambert et al., 2006), to the maximum deviation from an average load and to the linear and quadratic sum of such deviations.

Unfortunately, BACP is actually simpler than the real problem that universities have to solve in practice. We try to overcome this limitation and define a new, more general, formulation that includes the previous one and models a more realistic situation. Specifically, the new formulation makes it possible to include more than one curriculum, with courses shared among curricula. Moreover, it includes professors' preferences and unavailabilities for teaching in some terms.

We introduce ten new instances obtained from real data from University of Udine. Our instances are much larger and exhibit different structure, since they come from very different cases. As we will see, the new formulation and the new instances turn out to be much harder to solve than the previous BACP. Their optimal solutions remain unknown.

We revisit on the new problem the different criteria for balancing the load distribution for all curricula, similarly to what done in Monette et al. (2007). We give evidence that the quadratic criterion is the most appealing and we use it as main problem formulation in our analysis.

We focus on both integer programming and hybrid local search methods. Once assessed the infeasibility of an exact integer programming approach, its goal is providing good-quality solutions via upper bounds and, above all, lower bounds to assess the quality of the local search in absolute terms. On the quadratic problem formulation we improve the lower bound of a direct integer programming model, by solving a surrogate problem formulation and a problem decomposition with constraint relaxation.

The local search solvers tested in this paper are designed following the concept of *Generalized Local Search Machine*, which is a formal framework for representing search control introduced by Hoos and Stützle (2005, Chapter 3). We experiment with a number of machines, equipped with Simulated Annealing and Dynamic Tabu Search and with an intensification phase based on the exploration of a large neighborhood.

We make public all data on the web at `http://www.diegm.uniud.it/satt/projects/bacp/` together with a program to validate solutions. This is considered good practice when proposing a new problem and helpful to foster further research (Schaerf and Di Gaspero, 2007).

We introduce the problem formulation together with a toy example, in Section 2. We describe the integer programming models in Section 3 and the local search methods in Section 4. Empirical configuration of the heuristics and computational results on both exact and heuristic methods are reported in Section 5. Conclusions are drawn in Section 6.

2 Problem Formulations

First, we present the BACP formulation given by Castro and Manzano (2001). Then, we extend this formulation for dealing with the more realistic situation.

2.1 BACP Formulation

The basic formulation consists of the following entities and constraints:

Courses: Let C be the set of courses to be taught during the planning horizon of a university degree. Each course $c \in C$ gives a number of *credits* $r_c \in \mathbb{Z}^+$.

Periods: The planning horizon in divided in *academic years*, and each academic year is divided into terms. Each term is a *teaching period* in which courses can take place. Let P be the set of teaching periods, uniquely identified by the corresponding year and term. For example, a three-year degree organized in four terms per year has 12 periods, i.e., $P = \{1, \dots, 12\}$, and the first terms of each year are $\{1, 5, 9\}$.

Load limits: A minimum and a maximum number of courses, denoted by m and M, respectively, that can be assigned to each term.

Prerequisites: Based on their content, some courses have prerequisites, i.e., a set of courses that the students must attend earlier. Prerequisites are formalized with a precedence graph, that is, a directed acyclic graph $D = (V, A)$. Each vertex $i \in V$ represents a course, and each arc $(i, j) \in A$ a precedence relation, stating that the course i is a prerequisite of course j. If course i is a prerequisite of course j, then it has to be assigned to a teaching period that precedes the one assigned to course j.

Equal distribution of load: The distribution of credits among the teaching periods must be balanced. Ideally, each term should have the same number of credits.

The problem consists in finding an assignment of courses to periods that satisfies all the load limits and prerequisites constraints. The objective function accounts for the balancing of credits in periods. In detail, the objective function (to be minimized) used by Hnich et al. (2002) is *the maximum number of total credits per period.* For example, the CSPLib instance bacp8 has 46 courses for a total of 133 credits and 8 periods. The average number of credits per period is $133/8 = 16.625$. Therefore, the lower bound of the maximum number of credits per period is 17. Solutions with value 17 are thus optimal.

The complexity of BACP is as follows.

Theorem 1. *For any objective function that is polynomially computable, the decision version of BACP is strongly NP-complete.*

Proof. The proof that BACP is in NP is a consequence of the fact that the objective function is polynomially computable. The proof of completeness is based on a reduction from the 3-partition problem (Garey and Johnson, 1979, Prob. SP15) to the decision version of BACP that asks whether an ideally balanced distribution of credits exists. Given an instance of the 3-partition problem consisting of the integers $a_1, \dots, a_{3t}, b$, such that $b/4 < a_j < b/2$ and $\sum_{j=1}^{3t} a_j = tb$, an instance of BACP can be constructed as follows. The number of courses is set equal to $3t$ and the number of periods to t. The credits of the courses are set to $r_{c_1} = a_1, \dots, r_{c_{3t}} = a_{3t}$. Then, a solution of the BACP with cost equal to b exists only if the 3-partition problem has a solution, that is, if the numbers $a_1, \dots, a_{3t}$ can be partitioned in t pairwise-disjoint three element subsets, whose elements sum is b. $\square$

As a consequence of this fact, the optimization version of BACP is therefore strongly NP-hard.

2.2 GBACP Formulation

We extend the BACP formulation adding two features that arise in practice, and we call the resulting problem *Generalized BACP* (GBACP).

Curricula: In BACP, it is implicitly assumed that a student takes all courses without personal choices, whereas in practice a student can select among a set of alternatives. A *curriculum* is a set of courses representing a possible complete selection of a student. Let Q denote a curriculum, and $\mathcal{Q}$ be the collection of curricula. Sharing of courses among curricula is possible. The constraints defined in the BACP for the course set C must be satisfied by every curriculum.

Preferences: The professors can express preferences for their teaching periods. Specifically, a professor can indicate some terms of the year as undesirable for teaching a specific course. Let $U \subseteq C \times P$ be the set of undesirable assignments. In the three-year degree example used above, if a professor declares undesirable to teach the course c in the first term of the years, then we have $(c, 1), (c, 5), (c, 9) \in U$. Any assignment of a course c to a period p with $(c, p) \in U$ determines a violation of a preference.

Figure 1 shows a small GBACP instance with 6 courses, 4 periods (2 years and 2 terms per year), 3 curricula, and 4 pairs of preferences U. The precedence graph is given, along with all the input data.

It is easy to see that GBACP with any polynomially computable objective function is also strongly NP-hard because BACP reduces to GBACP. Indeed, a procedure that

$$\begin{array}{rl} C := \{ & A \quad B \quad C \quad D \quad E \quad F \quad \} \\ r := \{ & 5 \quad 10 \quad 5 \quad 4 \quad 20 \quad 7 \quad \} \end{array}$$

$$P := \{ \quad 1 \quad 2 \quad 3 \quad 4 \quad \}$$

$$\mathcal{Q} := \{[A, B, C, D], [B, C, D, F], [A, B, E, F]\}$$

$$U := \{(B, 1), (B, 3), (A, 2), (A, 4)\}$$

$$\begin{tikzcd} & C \arrow[dr] & \\ D \arrow[ur] \arrow[dr] & & F \\ & E & \end{tikzcd}$$

Figure 1: An illustration of a GBACP instance. On the left, we are given the set of courses C, the corresponding vector of credits r, the set of periods P, the collection of curricula $\mathcal{Q}$, and a set of preferences U. On the right, we are given the precedence graph.

solves GBACP with one curriculum and no undesirability solves also the BACP. Thus GBACP must be at least as hard as BACP.

The satisfaction of professors' preferences along with the other constraints is not always possible, and it generally goes against the balanced load distribution of credits for curricula. Therefore, we have decided to relax the preference constraints, and to consider them as soft, penalizing their violation in the objective function. This is reasonable, because in practice the violations of preferences are much more tolerable than violations of prerequisites and load limits. This constraint relaxation leads to a constrained bi-objective optimization problem. Although multi-objective problems can be treated in different ways (Ehrgott, 2005), we limit here ourselves to minimize a weighted sum of the two criteria.

Let s be a candidate solution, that is, an assignment of each course to a teaching period. The two criteria considered are: (i) the violations $L(s)$ of the ideal load balancing constraint, and (ii) the violations $R(s)$ of professors' preferences. Let w_1 and w_2 be two given weights that penalize the violations $L(s)$ and $R(s)$, respectively. The bi-objective function we consider for GBACP is:

$$F_\ell(s) = w_1 \, L_\ell(s) + w_2 \, R(s) \tag{1}$$

While the term $R(s)$ is simply the sum of the undesired course–period pairs appearing in a solution s, the term $L_\ell(s)$ deserves more attention.

2.3 Balancing the Course Load

The term $L(s)$ guides toward solutions with balanced distributions of credits over the periods. The criterion used by Hnich et al. (2002) is the so-called min–max: *Minimize the maximum number of total credits per period.* This choice is adequate for the CSPLib instances, because they all admit a solution with course load equally distributed. Unfortunately, when more general instances are considered this criterion fails to balance the course load.

For the instance given in Figure 1, the min–max objective function does not balance properly. In fact, there are many solutions of cost 20, and some are shown in Figure 2. This is clearly the optimal value, because of the presence of course E of 20 credits. However, the set of solutions of cost 20 includes both solutions that spread the other courses evenly (Figure 2.c), and others that concentrate them in some periods (Figure 2.a).

For this reason, we investigate the balance of the course load using different *norms*, as proposed by Monette et al. (2007). Our criterion consists of minimizing the *sum of the distances from an ideal distribution of credits.* Let us associate to each curriculum a value indicating the ideal distribution of credits throughout the periods. Given a solution s, let $\vec{z}(s)$ be a matrix of $|\mathcal{Q}| \times |P|$ elements indicating for each curriculum Q in $\mathcal{Q}$ the credit load in each period p of P.

For a curriculum Q, the ideal credit load is denoted by $\alpha(Q)$ and is equal to $\alpha(Q) = \sum_{c \in Q} r_c/|P|$; let $\boldsymbol{\alpha}$ be a matrix of size $|\mathcal{Q}| \times |P|$ composed of $|\mathcal{Q}|$ vectors with every elements equal to $\alpha(Q)$.

We define $L_\ell(s) = \|\vec{z}(s) - \alpha\|_\ell$ as the distance in the ℓ-norm of the load distribution of assignment s from the ideal load distribution $\vec{\alpha}$. Depending on the ℓ-norm, used, we get for $\ell = 1, 2, \infty$ three different objective functions:

$$\begin{aligned} L_1(s) &= \|\vec{z}(s) - \boldsymbol{\alpha}\|_1 = \sum_{Q \in \mathcal{Q}} \sum_{p \in P} |z_p(Q,s) - \alpha(Q)|, & (2) \\ L_2(s) &= \|\vec{z}(s) - \boldsymbol{\alpha}\|_2 = \sqrt{\sum_{Q \in \mathcal{Q}} \sum_{p \in P} \left(z_p(Q,s) - \alpha(Q)\right)^2}, & (3) \\ L_\infty(s) &= \|\vec{z}(s) - \boldsymbol{\alpha}\|_\infty = \max_{p \in P, Q \in \mathcal{Q}} |z_p(Q,s) - \alpha(Q)|. & (4) \end{aligned}$$

The balancing criterion used in Hnich et al. (2002) is related to L_∞, since with a single curriculum the objective function reduces to $\max_{p \in P} z_p(s)$.

Note that because of the norm relation $\|\cdot\|_\infty \leq \|\cdot\|_2 \leq \|\cdot\|_1$, the following statement holds: $L_\infty \leq L_2 \leq L_1$.

Using one of the definitions (2)-(4) in the objective function (1) it might be that, due to fractional values of $\alpha(Q)$, the optimal solution has value different from zero even in the absence of prerequisites and load constraints and perfect balance of working load. For example, suppose there are five periods, five courses and one single curriculum including all five courses. Two courses give four credits each and the other three give five credits each. Let a solution $\bar{s}$ place one of these courses per each of the five periods: this is clearly a *perfect* solution, and a better balanced one does not exist. Yet, we have $\alpha = \frac{2\times 4 + 3 \times 5}{5} = 4.6$ and the norms, computed as above, are all greater than zero: $L_\infty(\bar{s}) = 1.16$, $L_2(\bar{s}) = 2.03$, and $L_1(\bar{s}) = 3.83$.

In order to make perfectly balanced solutions having cost zero, we modify the difference of terms in the norm functions (2)-(4) using the ceil and floor operators as follows:

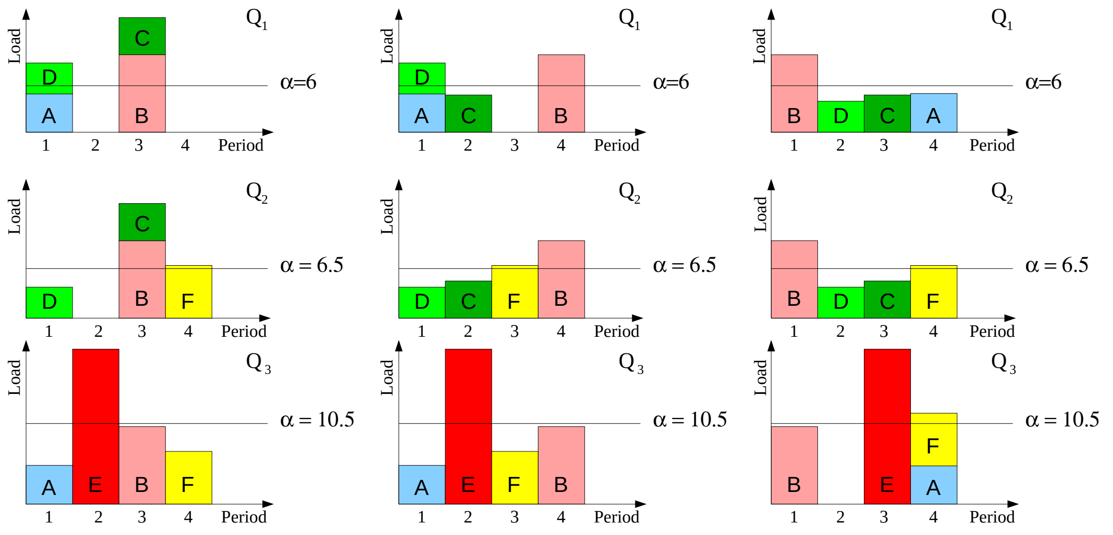
$$z_p(Q,s) - \alpha(Q) = \begin{cases} z_p(Q,s) - \lceil \alpha(Q) \rceil, & \text{if } z_p(Q,s) > \lceil \alpha(Q) \rceil, \\ \lfloor \alpha(Q) \rfloor - z_p(Q,s), & \text{if } z_p(Q,s) < \lfloor \alpha(Q) \rfloor, \\ 0, & \text{otherwise.} \end{cases} \quad (5)$$

Using relation (5) to compute the (modified) norm functions of the solution $\bar{s}$ of the previous example yields costs equal to zero. Note however that prerequisite and preference constraints can make a perfectly balanced solution infeasible. Hence, even using (5) we might have optimal solutions with non-null objective value. The CSPLib instances admit a perfectly balanced solution, that is, the optimal solution s^* has $L'_\infty(s^*) = L'_2(s^*) = L'_1(s^*) = 0$, where the prime apex is used to indicate that the norms are modified using equation (5).

Note that the application of (5) has the other favorable effect of rendering integer all terms in (2), (3) (before square root) and (4). In local search algorithms this is a convenient property. An alternative way of forcing integrality would be multiplying by $|P|$ all terms. This, however, would not always associate zero cost to perfectly balanced work load.

Figures 2.a, 2.b, and 2.c show an optimal schedule for each objective functions L_∞, L_2 and L_1, respectively, and with term $R(s)$ omitted. Both the objective functions L_1 and L_2 redistribute clearly better than L_∞. Observe, for example, that the curricula Q_1 and Q_2 are clearly unbalanced in Figure 2.a. On the contrary, it is less easy to establish which solution is better among those of Figures 2.b and 2.c. However, from the practical point of view, several small deviations are less harmful than a few large discrepancy, and this is best accounted for by the L_2-norm. Therefore, in the following, the problem formulation on which we focus our attention is the one with the L_2-norm in the objective function. The other two norms, L_1 and L_∞, remain however relevant to determine upper and lower bounds, respectively.

Our preference for L_2 is supported also by the study of Monette et al. (2007). Their extensive empirical analysis on several randomly generated instances of BACP, shows that the quadratic norm approximates best the others, that is, a solution that is optimal for that criterion is also often optimal with respect to the other criteria, while the converse is hardly true.



$\mathbf{L_\infty = 9.5}$, $L_2 = 20.40$, $L_1 = 61$ $L_\infty = 9.5$, $\mathbf{L_2 = 14.70}$, $L_1 = 41$ $L_\infty = 10.5$, $L_2 = 15.68$, $\mathbf{L_1 = 38}$
$\mathbf{L'_\infty = 9}$, $L'_2 = 19.52$, $L'_1 = 57$ $L_\infty = 9$, $\mathbf{L'_2 = 13.82}$, $L'_1 = 37$ $L'_\infty = 10$, $L'_2 = 14.76$, $\mathbf{L'_1 = 34}$

(a) Optimizing with L_∞ (b) Optimizing with L_2 (c) Optimizing with L_1

Figure 2: Three different solutions to the GBACP instance of Figure 1. The load profile is depicted for each curriculum Q_1, Q_2, and Q_3. In the charts, the horizontal axis represents the teaching periods and the vertical axis represents the credit loads. The solutions are measured with the three norms L_1, L_2, and L_∞. The norms modified using equation (5), are given with the prime apex, i.e., they are denoted by L'_1, L'_2, and L'_∞. The numbers in bold indicate the norm actually used as objective function.

3 Integer Programming Model of GBACP

Let x_{cp} be a binary variable equal to one if the course c is assigned to the teaching period p. The integer (quadratic) model of the GBACP problem is as follows:

$$\min \quad w_1 L'_\ell + w_2 \sum_{(c,p)\in U} x_{cp} \tag{6}$$

$$\text{subject to} \quad \sum_{p \in P} x_{cp} = 1, \qquad \forall c \in C, \tag{7}$$

$$m \le \sum_{c \in Q} x_{cp} \le M, \qquad \forall Q \in \mathcal{Q}, p \in P, \tag{8}$$

$$\sum_{s=1}^{p-1} x_{c_1 s} \ge x_{c_2 p}, \qquad \forall (c_1, c_2) \in A, p \in P, \tag{9}$$

$$\sum_{p \in P} p\, x_{c_2 p} - \sum_{p \in P} p\, x_{c_1 p} \ge 1, \qquad \forall (c_1, c_2) \in A, \tag{10}$$

$$x_{cp} \in \{0, 1\}, \qquad \forall c \in C, p \in P. \tag{11}$$

Constraints (7) assign each course c to a single teaching period p. Box constraints (8) enforce that the number of courses assigned in a period stay within its limits. Constraints (9) and (10) impose the precedence relations among the prerequisites. Note that we have modified the precedence graph by computing the transitive closure and hence adding all the arcs implied by the precedence relations.

The term L'_ℓ in the objective function (6) depends on the norm used to balance the credit load. We focus first on L'_1 and L'_2, i.e., L_1 and L_2 modified as in (5). Both these two norms yield non-linear objective functions that can be linearized with standard techniques, adding $3 \times |\mathcal{Q}| \times |P|$ auxiliary variables. Let u_{Qp}, y^+_{Qp} and y^-_{Qp} be the variables used to linearize L_ℓ as follows:

$$
\begin{aligned}
& y^+_{Qp} - y^-_{Qp} = \sum_{c \in C} r_c\, x_{cp} - \lfloor \alpha(Q) \rfloor + (\lfloor \alpha(Q) \rfloor - \lceil \alpha(Q) \rceil)\, u_{Qp}, && \forall Q \in \mathcal{Q}, p \in P, && (12)\\
& y^+_{Qp} \le M\, u_{Qp}, && \forall Q \in \mathcal{Q}, p \in P, && (13)\\
& y^-_{Qp} \le M\,(1 - u_{Qp}), && \forall Q \in \mathcal{Q}, p \in P, && (14)\\
& y^+_{Qp} \ge 0, y^-_{Qp} \ge 0, && \forall Q \in \mathcal{Q}, p \in P, && (15)\\
& u_{Qp} \in \{0,1\}, && \forall Q \in \mathcal{Q}, p \in P. && (16)
\end{aligned}
$$

where M is a large enough constant. At this point the two norms are computed as:

$$
L'_1 = \sum_{Q \in \mathcal{Q}, p \in P} (y^+_{Qp} + y^-_{Qp}), \qquad \text{and} \qquad L'_2 = \sum_{Q \in \mathcal{Q}, p \in P} (y^+_{Qp} + y^-_{Qp})^2. \qquad (17)
$$

Using these terms in the objective function (6) and adding the constraints (12)-(16) to the problem (6)-(11) yields an Integer Linear Programming formulation for L'_1, and an Integer Quadratic Programming formulation for L'_2, respectively. Both formulations are used in our computational evaluation. Note that in the case of L'_2 we can omitted the square root, since the objective function is a monotone function. However, once computed the optimal solution, in order to compare the L'_2 with the bound given by L'_1 we have to compute the square root.

In order to compute L'_∞, we need one additional variable, denoted by $\hat{y}$, that linearizes the min–max objective function (4). The $\hat{y}$ variable is used as follows:

$$
\begin{aligned}
& L'_\infty = \hat{y}, && && (18)\\
& \hat{y} \ge y^+_{Qp} + y^-_{Qp}, && \forall Q \in \mathcal{Q}, p \in P. && (19)
\end{aligned}
$$

The integer (quadratic) formulation is appealing because it permits to compute both lower and upper bounds to the optimal solution of the norm 2 component in the objective function. Upper bounds can be computed by using L_1, and lower bounds by using L_∞. Unfortunately, as we will see in Section 5.3, the L_1 lower bounds, as well as those obtained by the linear relaxation, turn out to be rather weak. Therefore, we propose below a sharper lower bound based on a decomposition.

3.1 A Decomposition Procedure to Compute Lower Bounds

A sharper lower bound can be computed by decomposing the problem into a number of smaller subproblems, in which some constraints are relaxed. This is possible because problem (6)-(11), apart from constraints (7), depends only on the specific curriculum. Therefore, by solving $|\mathcal{Q}|$ smaller subproblems, one for each curriculum, and summing up their optimal values (or their lower bounds), we obtain a valid lower bound to the original problem. Formally, this corresponds to restrict the constraints (7) and (9) to a single curriculum Q as follows:

$$\sum_{p \in P} x_{cp} = 1, \qquad \forall c \in Q, \tag{20}$$

$$\sum_{s=1}^{p-1} x_{c_1 s} \geq x_{c_2 p}, \qquad \forall (c_1, c_2) \in A_Q, p \in P. \tag{21}$$

where A_Q is the subset of precedence relations that involves only the courses in the curriculum Q. Similarly, the constraints (12)–(16) and (19) that linearize the objective function (6) are simply defined over a single curriculum Q. Note that also the variables are restricted to the set Q instead of the whole set of courses C.

The lower bounds obtained by this procedure using L_2 in the objective function is denoted herein by $LB(\mathcal{Q})$. It will be used to evaluate the quality of the solution obtained by our local search heuristics, described in the next section.

4 Local Search for GBACP

In order to apply local search to GBACP we need to define a few components. We first define the search space and the procedure for generating an initial solution. Then, we define the neighborhood structure. Finally, maintaining a component-wise approach we describe the search strategies and the high-level templates to combine them.

4.1 Search Space and Initial Solution

We represent a solution to GBACP by an assignment of courses to periods, that is a mapping $\sigma : C \rightarrow P$, in which each c is assigned to a period. In the implementation, an assignment is simply a vector of length $|C|$.

Based on prerequisites, we slightly restrict the assignable range of periods for each course. Assuming $P = \{1, \ldots, n\}$, and courses c_1 and c_2 with $(c_1, c_2) \in A$, we never assign c_1 to period n, and in turn c_2 to period 1. (We recall that we use the the transitive closure of the prerequisites).

All other assignments are taken as candidate solutions and included in the search space, even if they violate load limits and prerequisites. These two latter constraints are considered as hard constraints in the usual local search terminology and their violations is taken into account by a *distance to feasibility* measure that is added in the evaluation function with higher penalties w.r.t. the objective function (1). More formally,

$$\begin{aligned} f(\sigma) = P_1 \cdot \sum_{\substack{Q \in \mathcal{Q} \\ p \in P}} & \max\{\texttt{count}(\sigma, Q, p) - M, m - \texttt{count}(\sigma, Q, p), 0\} \\ & + P_2 \cdot \#\{(c_1, c_2) \mid (c_1, c_2) \in A \text{ and } \sigma(v_1) \geq \sigma(c_2)\} \\ & + P_3 \cdot w_1 \cdot L'_\ell(\sigma) \\ & + P_4 \cdot w_2 \cdot R(\sigma) \end{aligned} \tag{22}$$

where $\texttt{count}(\sigma, Q, p)$ is the number of courses from Q scheduled in p in solution σ, $P_3 = P_4 = 1$ and $P_1 = P_2 \gg 1$.

The initial state is generated in a totally random way: each course is assigned a random period uniformly chosen from its assignable range.

4.2 Neighborhood Relations

We consider two different neighborhood relations:

MoveCourse (MC): Move one course from its period to another one within its assignable range.

SwapCourses (SC): Take two courses in different periods and swap their periods. The two courses to be eligible for a move must have at least one curriculum and their current periods must be compatible in terms of assignable range.

The MC neighborhood has been investigated in the preliminary work by Di Gaspero and Schaerf (2008). Here, we focus on the neighborhood obtained by the union of MC and SC, which, as we will see in Section 5, is more effective than MC alone. We denote this neighborhood as MC$\oplus$SC.

4.3 Generalized Local Search Machines

We describe our local search algorithms in terms of high-level search control strategies that assemble basic local search components. The formal framework to separate search control from search components is provided by the *Generalized Local Search Machines* (GLSM), introduced by Hoos and Stützle (2005). In this framework, the basic search components are represented as states (i.e., nodes) of a Finite State Machine, whereas the transitions (i.e., arcs) correspond to conditions for modeling the search control. A transition takes place when the node has finished its execution, i.e., it has met its stopping criterion. The transition is carried out on the arcs. Each arc is labeled with a pair c/a, where c is the condition that needs to be fulfilled to select that arc, and a is the action performed on GLSM variables during the transition. The condition c can be omitted when there is only one possible transition from a state (unconditional transition), whereas the action a is omitted when there is no action to perform.

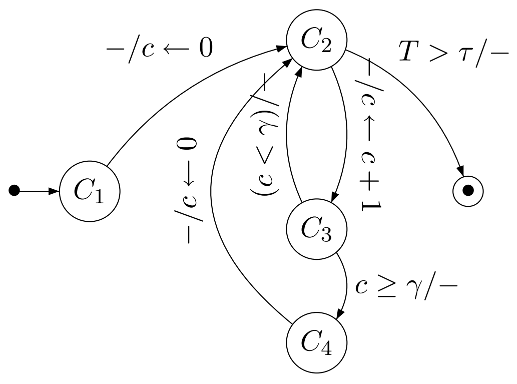


Figure 3: An example of GLSM with three search components.

In Figure 3 we show an example of a GLSM with four search components. The search starts from C_1 and when this component has finished it unconditionally passes its solution to component C_2, resetting the value of a counter c. Component C_2 continues the search, followed by C_3. Also the transition between C_2 and C_3 takes place unconditionally and the value of counter c is incremented by 1. Differently from the previous cases, from C_3 two transitions are allowed, back to C_2 or to C_4, depending on the condition $c \geq \gamma$ or $\neg(c \geq \gamma) = c < \gamma$, respectively. After C_4 the process is unconditionally started again from C_2, resetting the value of counter c. The whole procedure terminates in C_2 when an overall timeout τ has expired.

In order to make the notation lighter, in case of multiple transitions originating from a given state we omit the annotation of the complementary conditions. For example, in the figure the condition $c < \gamma$ on the transition between $C_3 and C_2$ can be omitted since it is implied by the condition $c \geq \gamma$ of the C_3–C_4 transition.

Within the GLSM framework it is possible to specify complex strategies such as VNS (Hansen and Mladenović, 1999), ILS (Lourenço et al., 2001), and other metaheuristics.

In what follows, we use states of the machine to represent either *runners*, that are basic local search algorithms using the neighborhoods previously defined, or *kickers*, that are perturbation procedures performing a single complex move in a composite neighborhood.

4.4 Search Components for GBACP

We restrict our attention to the search components that gave the most promising results in Di Gaspero and Schaerf (2008) and we study them more deeply. In detail, we consider the following three components for our GLSMs, two runners and one kicker.

Simulated Annealing (SA) At each step of the search a random neighbor is selected. The neighbor is accepted as new current solution if it is an improving solution or with a probability that depends exponentially on the degradation of solution quality. In detail, if the cost of moving to the neighbor is Δf, the move is accepted with probability $e^{-\Delta f/T}$, where T is a time-decreasing parameter called *temperature*.

Among the different decreasing strategies for the temperature (*cooling schedules*), we adopt the geometric scheme, that is, after a number of explored moves the temperature is multiplied by a factor $\gamma < 1$ (i.e., $T \leftarrow \gamma T$). The search is stopped when the temperature goes under a value T_{min}.

Dynamic Tabu Search (DTS) At each step a subset of the neighborhood is explored and the neighbor with the best cost value becomes the new solution independently of the fact that its cost value is better or worse than the current one. The subset is induced by the *tabu list*, i.e., a list of the moves recently performed, which are currently forbidden and thus excluded from the exploration. Our tabu search implementation employs a dynamic short-term tabu list (called Robust Tabu Search in Hoos and Stützle 2005), so that a move is kept in the tabu list for a random number of iterations in the range $[k_{min}, k_{max}]$.

The tabu status of a move can be overruled by the so-called *aspiration criterion* which makes a move acceptable even if it is tabu. In this work, we use a basic aspiration criterion which states that a move is accepted if it improves on the current best solution.

The *dynamic* variant of the algorithm is equipped with a mechanism that adaptively changes the shape of the cost function. In detail, the constraints which are satisfied for a given number of iterations will be relaxed (their penalties are reduced by a factor ξ, i.e., $P_i \leftarrow P_i/\xi$) in order to allow the exploration of regions of the search space where those constraints do not hold. Conversely, if some constraint is not satisfied, it is tighten ($P_i \leftarrow P_i \cdot \xi$) with the aim of driving the search toward its satisfaction.

Kickers (K) Kickers are special-purpose components that perform one single complex step obtained by sequencing moves up to an arbitrary number (Di Gaspero and Schaerf, 2006). Each move of the sequence is selected from a certain neighborhood. We use the symbol $\otimes$ to indicate the neighborhood used. Thus, for example, the kicker $\mathsf{MC}\otimes(\mathsf{MC}\oplus\mathsf{SC})$, denotes a composition of moves obtained by a move in MC and a move in $\mathsf{MC}\oplus\mathsf{SC}$. Kickers support three strategies for selecting the moves: (i) *random kick* (K_r), that is a sequence of random moves, (ii) *first kick* (K_f), the first improving sequence in the exploration of the composite neighborhood, (iii) *best kick* (K_b), the best sequence in the exhaustive exploration of the composite neighborhood. Note that we evaluate the entire composite move in order to establish if it is improving. The single moves of the sequence can also be worsening.

In addition, in order to reduce the computation time needed to explore the composite neighborhood, only sequences of moves that are related to each other are evaluated, in a fashion similar to ejection chains (Ahuja et al., 2002). Specifically for our problem, two moves are considered as related if they involve the same period and courses that have at least one curriculum in common.

Random kicks can be used for diversification purposes (thus giving raise to the Iterated Local Search strategy (Lourenço et al., 2001)), while first and best kicks are employed for intensifying the search. In this work we focus on intensification kicks, thus leaving out the random ones.

4.5 GLSM Templates

The three search components described are combined by means of GLSM templates. We consider various combinations and compare them experimentally. In order to perform such a comparison on a fair basis, we decide to grant to all of them the same total computation time. In the design of GLSMs, this entails that the only condition to go to the final state is the expiration of timeout τ.

We introduce the GLSM templates gradually by first reviewing the four basic templates in Figure 4. They all include one single generic runner R. The simplest one, shown in Figure 4(a), is composed by the generation of a starting solution, through a random assignment G_r, followed by the execution of the runner R.

If the runner terminates before the total computation time allowed, a simple variant that allows to make use of the whole time is the *multi-start* template, Figure 4(b), that consists in the repetition of solution generation and runner until the timeout is expired. The best solution encountered during the whole search is returned. The *multi-run* strategy, Figure 4(c) differs from the previous in that the runner is repeated from the best solution found in the previous run. These two latter strategies are simple mechanisms whose rationale is respectively to enhance diversification and intensification of a single runner. Finally, the *multi-start-multi-run* template shown in Figure 4(d) generalizes the previous ones at the price of introducing one additional parameter. This GLSM repeats only the execution of the runner for a maximum number ρ of times without an improvement (*idle rounds*, denoted by r), and then gets back to G_r to produce a new initial solution.

The GLSM templates that we test are extensions of the basic *multi-start-multi-run* strategy in which we replace the single runner by a sequence of two or three components. This yields the composite search machines that are shown in Figure 5(b-f). In these machines we also use kickers that implement an intensification of the search. We experimented with two *kicking* strategies: the first one, denoted simply by K, and represented in Fig. 5(b), consists in performing a single complex step in a composite neighborhood; the second one, denoted by K^+, and represented in Fig. 5(c), performs an iterative improvement local search made by complex moves until no further improvement is available, a mechanism similar to a Large Neighborhood Search (Ahuja et al., 2002).

The instantiation of the components R_1, R_2 and K will be unveiled in Section 5. There, we will identify the machines with the notation in the captions of Figure 5.

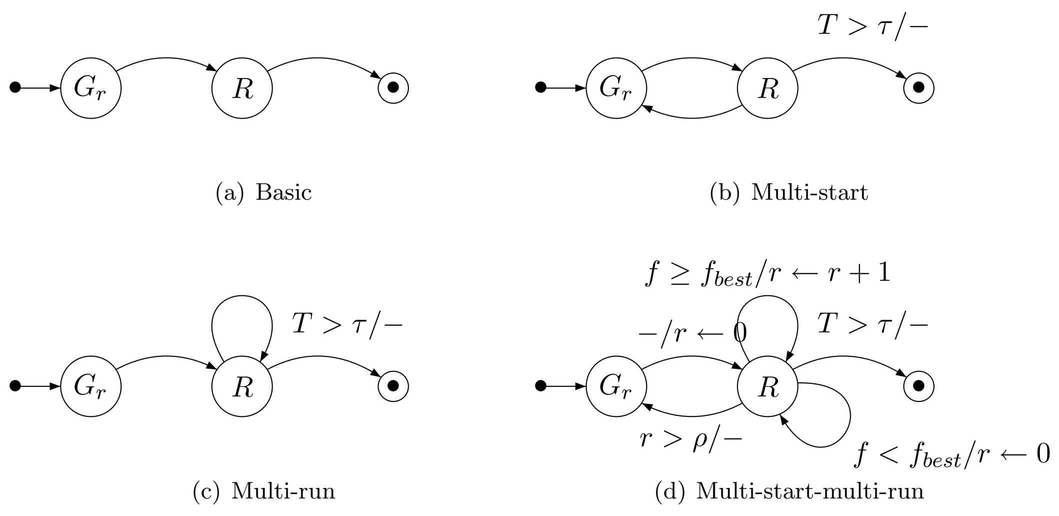


Figure 4: The basic GLSM templates.

5 Computational Results

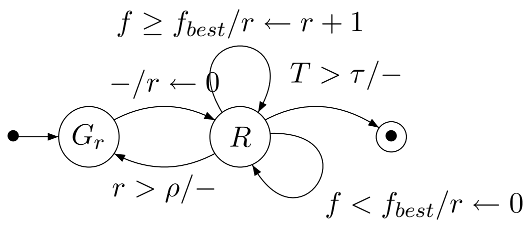
5.1 Benchmark Instances

Ten instances for the GBACP, called UD1–UD10, have been extracted from the database containing historical data at the School of Engineering of the University of Udine. As we faced three national regulations changes in ten years, we could build this dataset by selecting a number of structurally different instances. Table 1 summarizes the main features of the instances. For reference, we also include the three CSPLib instances.

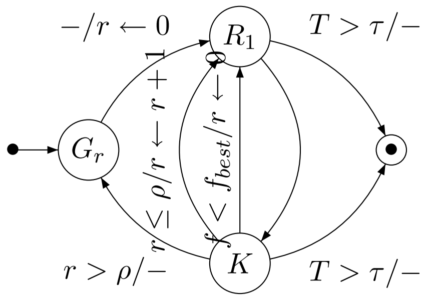
All instances are available from the web at `http://www.diegm.uniud.it/satt/projects/bacp/`, together with a format description, our best solutions and the C++ source code of the validator that *certifies* their scores.

All results are presented with weights $w_1 = 1$ and $w_2 = 5$. In the quadratic formulation, this choice implies that a preference violation will count more than a discrepancy of two credits but less than a discrepancy of three credits. We found this to be a good compromise in practice. In the experiments with local search we further fix $P_1 = P_2 = 1000$ and unless, diversely stated, the adjustment (5) is always present in the objective function.

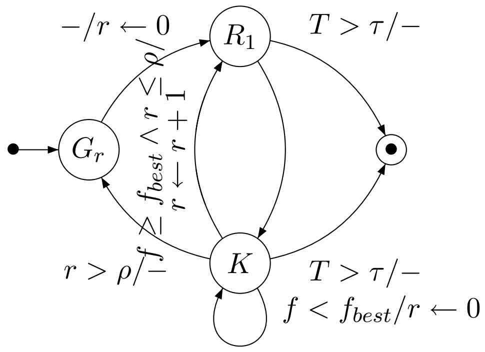
In the following, we report only results on instances UD1–UD10 because the CSPLib instances turned out to be extremely easy. For these instances, all ILP models and our simplest local search solver reach always a solution of cost 0 (obviously optimal) within one second.


$f \geq f_{best}/r \leftarrow r+1$
$-/r \leftarrow 0$
$T > \tau/-$
G_r
R
$r > \rho/-$
$f < f_{best}/r \leftarrow 0$

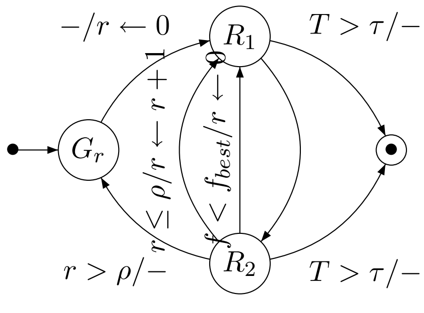
(a) R_1



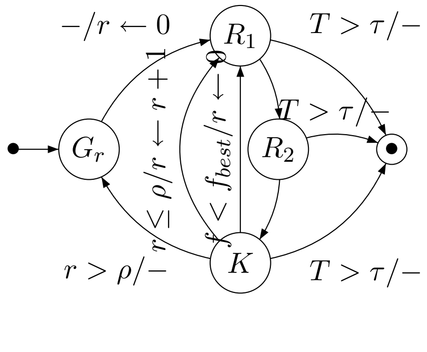
(b) $R_1 \triangleright K$



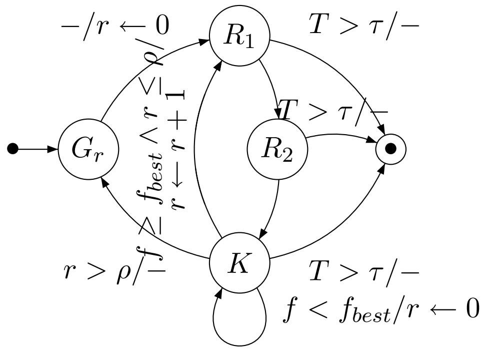
(c) $R_1 \triangleright K^+$



(d) $R_1 \triangleright R_2$



(e) $R_1 \triangleright R_2 \triangleright K$



(f) $R_1 \triangleright R_2 \triangleright K^+$

Figure 5: The composite GLSM templates (b)-(f) obtained from the *multi-start-multi-run* machine (a).

Instance	Periods (Years × Terms)	Courses	Curricula	Courses per curriculum	Courses per period	Prerequ.	Pref.
csplib8	8 (4 × 2)	46	1	46	5.75	33	0
csplib10	10 (5 × 2)	42	1	42	4.2	33	0
csplib12	12 (6 × 2)	66	1	66	5.5	65	0
UD1	9 (3 × 3)	307	37	34.62	3.847	1383	270
UD2	6 (2 × 3)	268	20	27.8	4.633	174	158
UD3	9 (3 × 3)	236	31	29.81	3.312	1092	198
UD4	6 (2 × 3)	139	16	25.69	4.281	188	80
UD5	6 (3 × 2)	282	31	34.32	5.72	397	162
UD6	4 (2 × 2)	264	20	27.15	6.787	70	110
UD7	9 (3 × 3)	302	37	33.89	3.766	1550	249
UD8	6 (2 × 3)	208	19	22.58	3.763	149	120
UD9	9 (3 × 3)	303	37	34.08	3.787	1541	255
UD10	6 (2 × 3)	188	15	25.07	4.178	214	110

Table 1: Statistics on the GBACP instances.

5.2 Implementations and General Settings

The integer programming models described in Section 3 have been implemented in AMPL and solved with CPLEX version 11.0. We make available the AMPL models from the web site. All the local search algorithms have been implemented in C++ language, exploiting the EasyLocal++ framework (Di Gaspero and Schaerf, 2003). EasyLocal++ is a tool for local search that provides the full control structures of the algorithms. The current version of EasyLocal++ also supports the implementation of the GLSMs described in this paper.

All experiments have been performed on a server Power Edge 1900 double quad-core at 2.66Ghz and with 4 GB RAM, running Ubuntu Server 64 bits (although, CPLEX and AMPL 11 were used in 32 bits). The local search software has been compiled using the GNU C++ compiler (v. 4.1.2), and it did not take advantage of multi-threading capabilities of the testing platform (i.e., the implementation is single-threaded).

As mentioned, we compare all local search algorithms on a fixed timeout basis set to 320 seconds on the computational architecture described above.

5.3 Integer Programming

We performed two sets of experiments with integer programming. In the first set, our goal was assessing the computational feasibility of the three IP models. In the second, we used the decomposition approach described in Section 3.1 to produce tight lower bounds for the assessment of the local search heuristics.

Table 2 shows the computational results of the first set of experiments. We solved the models (6)-(11) with (12)-(16) and (19) defined for $\ell = 1, 2, \infty$. In the table, we indicate the value of the objective function by F'_ℓ as in (1), with the prime apex to recall that we use the adjustment (5) in the computation of the norms. One single run on each instance is considered with timeout set to 3600 seconds. For each instance, the table gives the upper and lower bounds of the solution attained at the timeout, denoted by LB and UB, respectively. These values include preference violations and, in the case of L'_2, the square root is not calculated. The solutions with F'_1 and F'_∞ are also evaluated in terms of the 2-norm in order to assess the difference in quality implied by the three norms. The contribution of preference violation $w_1 R$ to the objective function is reported in a separated row.

We observe the following.

- The model with ∞-norm instances are often closed in less than one hour. When they are not closed the optimality gap is very small (note that the lower bound could be rounded up when fractional).
- The model with 2-norm is the hardest to solve providing also loose lower bounds and failing even to find a feasible solution on the instance UD7. The only instance where it performs relatively well is instance UD4. The solutions at the time limit are often much worse than those found by the model with 1-norm evaluated in terms of 2-norm, indicating that this latter model could be used as a surrogate model to optimize also the quadratic model.
- The model with 1-norm can prove the optimality on UD4 and reach good optimality gaps on at least the instances UD2, UD5 and UD10.

Table 3 focuses on the second set of experiments with the goal of computing good lower bounds for the quadratic formulation in which we are mainly interested. In the table three lower bounds are compared: $LB(F'_\infty)$ and $LB(F'_2)$ (now with square root included) reported from Table 2 and $LB(\mathcal{Q})$, computed with the decomposition procedure described in Section 3.1. For each lower bound, we report the value attained and the respective computation times in seconds. The main observation is that apart from the instance UD5, where the $LB(F'_\infty)$ is the best lower bound, the decomposition-based lower bound procedure outperforms the other two, both in terms of quality of the bounds and in terms of computation times.

A closer look at the detailed results of the decomposition procedure (not shown in the table) unveils that most of the balancing cost, above all for the instances UD1, UD7, UD9 is taken by few curricula. Moreover, one single subproblem is responsible for exceeding the time limit on the instance UD5. Finally, common to all instances, there are no preference violations in the solutions found to the subproblems.

Note that all the results presented in this section have been obtained using CPLEX with the default setting. We have tried to use the automatic tuning tool, introduced with CPLEX 11.0, but it was not possible to find some parameters suitable for all the instances with the three norm functions. For computing the decomposition lower bound $LB(\mathcal{Q})$, we just set the CPLEX parameters `mipemphasis` $= 3$, that tells CPLEX to stress the search for the optimal solution. With parameters tuned for each model

		UD1	UD2	UD3	UD4	UD5	UD6	UD7	UD8	UD9	UD10
	$LB_{F'_\infty}$		3.5	1.6	5	11.5			1.7		
	$UB_{F'_\infty}$	8	5	3	6	12	3	8	2	8	2
F'_∞	$UB_{F'_2}$	6120	693	838	540	4742	191	5948	155	4583	166
	w_2R	0	0	0	0	0	0	0	0	0	0
	time	1225	3600	3600	3600	3600	10	2114	3600	3429	267
	$LB_{F'_2}$	30.62	0	0	71.14	56.82	29.71	x	1	13.03	0
F'_2	$UB_{F'_2}$	10109	580	1425	396	8861	55	x	112	8266	149
	w_2R	10	15	40	0	140	0	x	0	20	10
	time	3600	3600	3600	3600	3600	3600	x	3600	3600	3600
	$LB_{F'_1}$	39.39	50	70.99		44.4	20.04	9	18	11.04	26.71
	$UB_{F'_1}$	178	78	134	114	134	30	186	32	318	30
F'_1	$UB_{F'_2}$	366	164	214	396	492	52	634	64	862	48
	w_2R	0	0	5	0	10	0	0	0	10	0
	time	3600	3600	3600	42	3600	3600	3600	3600	3600	3600

Table 2: Results from the IP models with the three norms (2)–(4) and modification (5) solved by CPLEX with a time limit of 3600 seconds. Computation times are in seconds. Note that all results including the 2-norm do not include square root computation.

results improve but the general conclusions remain substantially the same as discussed in this section.

5.4 Local Search

In the case of the local search heuristics, we first report about the experiments carried out for configuring and tuning the algorithms and then, once a final candidate has been selected, we discuss the numerical results it achieves on the benchmark instances.

5.4.1 Configuration and Tuning

In Fig 5 we introduced six GLSMs defined by parameters. Each of them can be instantiated with two basic local search strategies (SA and DTS) that come with their own parameters as well. Thus, the number of possible configurations to consider is large, possibly infinite due to the continuous nature of some of these parameters. In order to simplify the task of selecting the best instantiated composite machine, we divide it into two stages: first we tune the basic machines with a single local search component and then we plug the best configurations attained for these components in the composite GLSMs and we tune these latter. This organization is based on the heuristic assumption that the behavior of the basic local search components does not change when integrated within more complex search strategies in the GLSM. Evidence in a limited preliminary experiment supported this assumption. As a further simplifying element, we limit ourselves to test a few values from the domains of the continuous parameters.

	UD1	UD2	UD3	UD4	UD5	UD6	UD7	UD8	UD9	UD10
$LB(F'_\infty)$	8.00	3.50	1.60	5.00	11.50	3.00	8.00	2.00	8.00	2.00
time	1225	3600	3600	3600	3600	19	2114	3600	3429	267
$LB(F'_2)$	5.53	0.00	0.00	8.43	7.54	5.45	x	1.00	3.69	0.00
time	3600	3600	3600	3600	3600	3600	3600	3600	3600	3600
$LB(\mathcal{Q})$	10.54	11.40	10.30	18.00	1.73	5.92	10.95	5.20	11.66	6.16
time	3.5	459	19	12	3600	1516	594	695	3055	1.6
Best	10.54	11.40	10.30	18.00	11.50	5.92	10.95	5.20	11.66	6.16

Table 3: Comparing the quality of three lower bounds: (i) the lower bounds obtained with the ∞-norm, denoted with $LB(F'_\infty)$, (ii) the lower bounds produced by the quadratic program that optimizes F'_2, denoted with $LB(F'_\infty)$, (iii) the lower bound $LB(\mathcal{Q})$ obtained by our decomposition procedure presented in Section 3.1. A time limit of 3600 seconds was imposed overall. All results include square root computations.

Methodology The parameters and their selected values are used to produce a full-factorial design of configurations to test by means of F-Race (Birattari et al., 2002). F-race is a sequential testing procedure that uses the Friedman two-way analysis of variance by ranks to decide upon the elimination of inferior candidates. At each stage a new instance is selected, all left configurations run on it and weaker configurations discarded if enough statistical evidence has arisen against them. We use a canonical 0.05 as significance level in the tests. Since ten instances are too few to determine a clear winner in the F-race, we resample them until a maximum of 200 results per configuration have been collected. The transformation of results in ranks within the instances guarantees that in the statistical test procedure the aggregation of results over the ten GBACP instances is not negatively influenced by the inherent differences in the cost functions of the instances.

In the following, the results are presented in form of box-and-whiskers plots showing the distribution of the ranks obtained by each configuration. Moreover, boxes are filled with a gray level which is proportional to the stage of the F-Race procedure in which the corresponding configuration has been discarded (the darker, the sooner). In this way, the configurations that were found as equally good at the end of the procedure are denoted by white boxes. The variance of results around the median diminishes as configurations remain longer in the race. This is a well known effect and indicates that the assessment of the promising configurations becomes more precise.

Simulated Annealing Runner The parameters of SA are the starting temperature T_0, the cooling rate γ and the minimum temperature T_{min}. In addition, we include the maximum number of idle rounds ρ. The candidate values for these parameters are reported in Table 4. The results of the race are visualized in Figure 6.

From the figure it is clear that the best results are obtained by the following parameter setting: $T_0 = 100$, $\gamma = 0.999$, $T_{min} = 0.001$. Moreover, the setting is insensitive to the value of the number of restarts. This result can be explained by the fact that the

Parameter	Values
T_0	10, 100
γ	0.999, 0.9999
T_{min}	0.01, 0.001, 0.0001
ρ	1, 2, 4

Table 4: Experimental design for a *multi-start-multi-run* GLSM with SA. A full-factorial design yields 36 configurations.

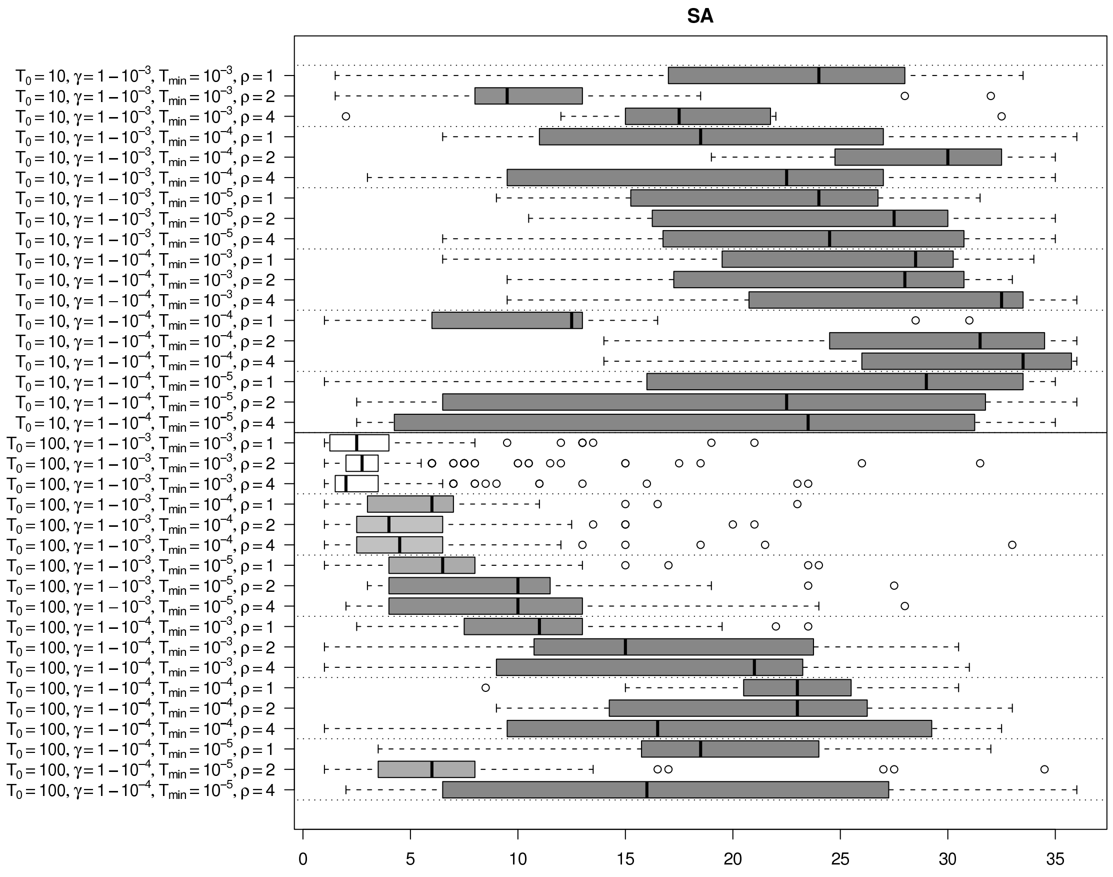


Figure 6: Results of the F-Race selection procedure on the SA machines.

disruption of the solution performed by the SA local search at early stages of the search when the temperature is high is similar to a restart.

Dynamic Tabu Search Runner The parameters of DTS are the shifting ratio ξ and the minimum length of the tabu list tl_m. In addition, we include the idle rounds ρ. In the implementation of DTS, the actual tabu list length is set in the range $[tl_m, tl_m + 5]$. Moreover, we link tl_m to the size of the instance, imposing it to be equal to a fraction

Parameter	Values
ξ	1, 1.1, 1.2, 1.4, 1.8
tl_m	$\lvert C\rvert/3$, $\lvert C\rvert/4$, $\lvert C\rvert/5$
ρ	1, 2, 4

Table 5: Experimental design for a *multi-start-multi-run* GLSM with DTS. A full-factorial design yields 45 configurations.

Parameter	Values
ϵ	first improvement, best improvement
μ	$MC \otimes MS$, $MS \otimes MC$
ρ	1, 2, 4

Table 6: Experimental design for Kicker-equipped GLSM. A full-factorial design yields 18 configurations.

of the number of courses $|C|$. The selected values for these parameters are reported in Table 5. Results are visualized in Figure 7.

In this case there is no clear winner, but a number of equivalent configurations that seem characterized by the use of restarts. Finally, it is worth noticing, that the basic non-dynamic tabu search (i.e., the one without dynamic weights, or $\xi = 1$) performs poorly on these instances.

Kickers In order to keep running times within the decided timeout, we restrict ourselves to kickers whose complex step is make of a sequence of exactly two moves in two different neighborhoods. Kickers have been tuned in conjunction with both SA and DTS *multi-run multi-start* machines. We report only those based on SA, but similar results have been obtained also with DTS.

The design parameters for Kickers are the strategy ϵ for exploring the composite neighborhood (either look for the *first* improving move or for the *best* one), the sequence of basic neighborhoods used μ, and again the overall number of allowed restarts ρ. SA is now a black box used with the best parameter setting found in the previous experiment. Among different sequences of neighborhoods considered, the combinations $\mathsf{MC}\otimes\mathsf{SC}$ and $\mathsf{SC}\otimes\mathsf{MC}$ turned out to be more effective than $\mathsf{SC}\otimes\mathsf{SC}$ and $(\mathsf{MC}\oplus\mathsf{SC})\otimes(\mathsf{MC}\oplus\mathsf{SC})$ and we concentrated only on those two. The selected values for these parameters are reported in Table 6 and results are depicted in Figure 8.

We observe that the combination $MC \otimes MS$ outperforms $MS \otimes MC$, and that the restarting mechanism (with $\rho = 4$) has a positive effect on the results. Moreover, the *best improvement* strategy is superior to the *first improvement* one.

Composite GLSMs The final comparison for the configuration of GLSMs comprises all the machines of Figure 5 equipped with the kicker and runners previously tuned. In addition, we include again the parameter ρ for which we consider the values $\{1, 2, 4\}$.

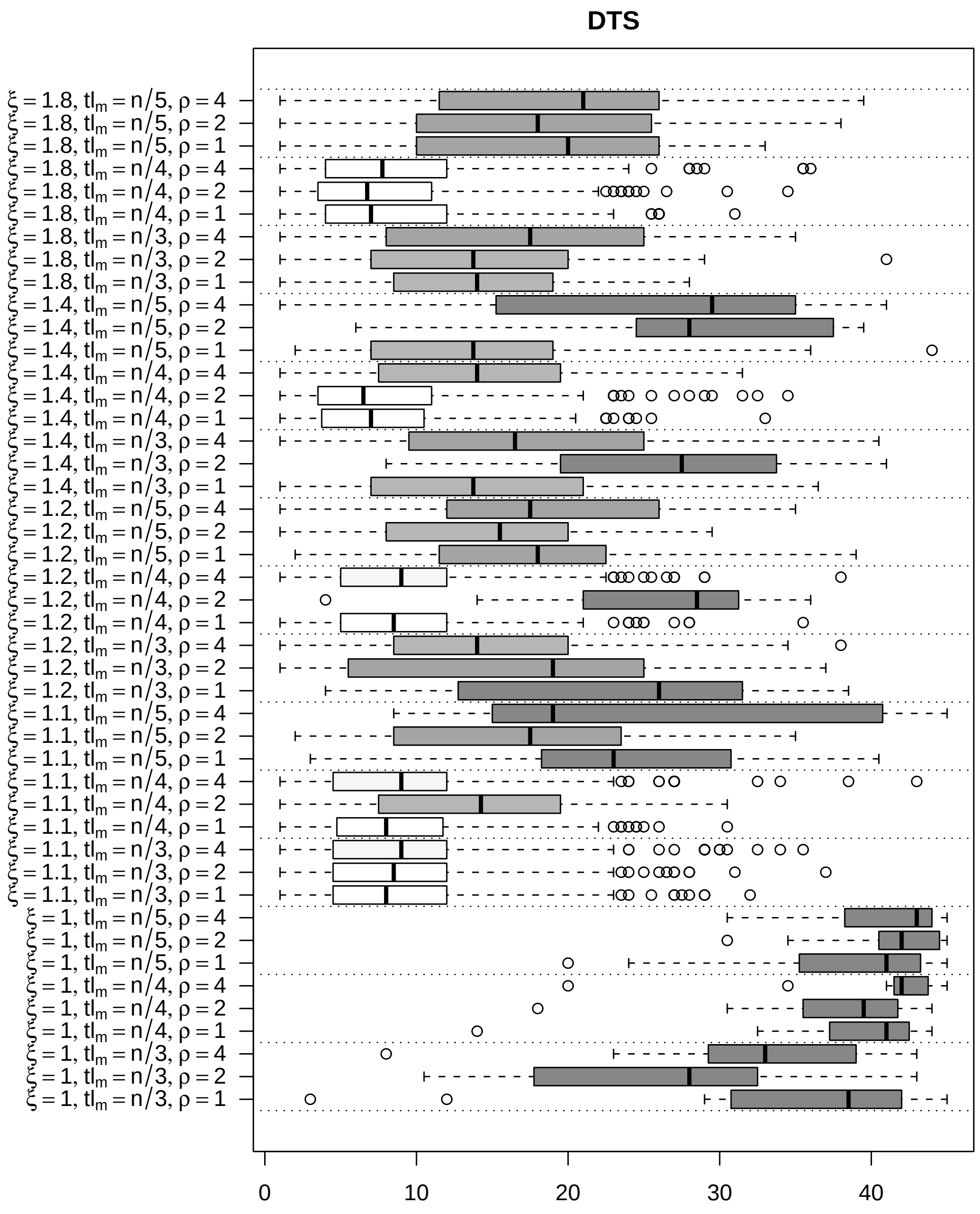


Figure 7: Results of the F-Race selection procedure on the DTS machines.

All together, we assembled 27 configurations.

The results of the experiment are reported in Figure 9. We observe that the best machines are $SA \rhd K(MC \otimes MS)_b$ and $SA \rhd K^{+}(MC \otimes MS)_b$. On the contrary, all machines with DTS perform poorly. Trying to explain the better performances of SA over DTS, we may conjecture the high computational cost of the full neighborhood

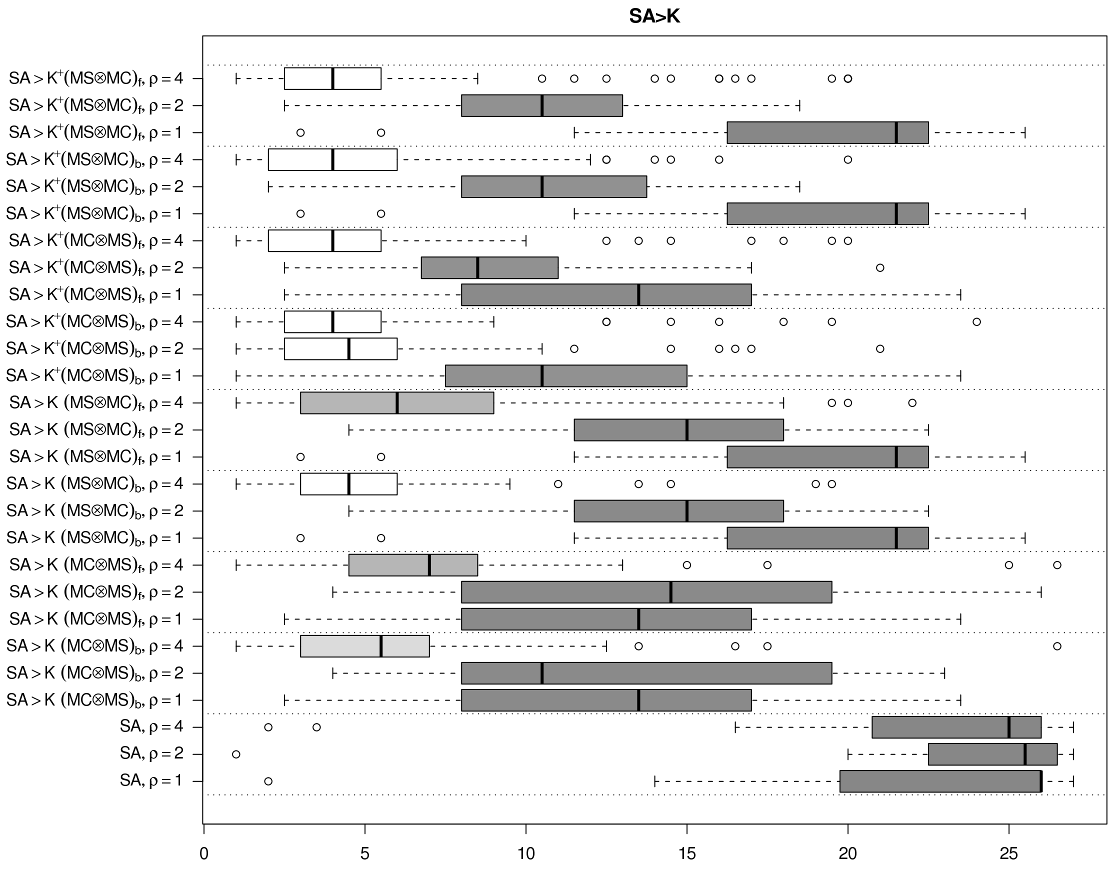


Figure 8: Results of the F-Race selection procedure on the Kickers in conjunction with the best SA runner.

exploration required at each iteration of DTS. Conversely, the random move selection of SA contributes to a faster move to a new neighbor. In addition, the kickers are more appropriately combined with SA, because they provide a form of intensification that blends well with the random diversifying strategy of SA.

5.4.2 Final Results

In Table 7, we report the results on the GBACP instances for $SA \triangleright K^+(MC \otimes MS)_b$ that is the best configuration of local search heuristic arisen from the previous analysis. More precisely, we run the heuristic 20 times on the ten instances with a timeout of 320 seconds and we report the median results as the most representative of the 20 trials. For the record of best solutions we also report the best results. These results and the corresponding solutions will be maintained at the web site `http://www.diegm.uniud.it/satt/projects/bacp/` for future benchmarking. In order to avoid rounding problems we give these results omitting the square root computation.

In the same table we also report a summary of the results presented in Tables 2 and

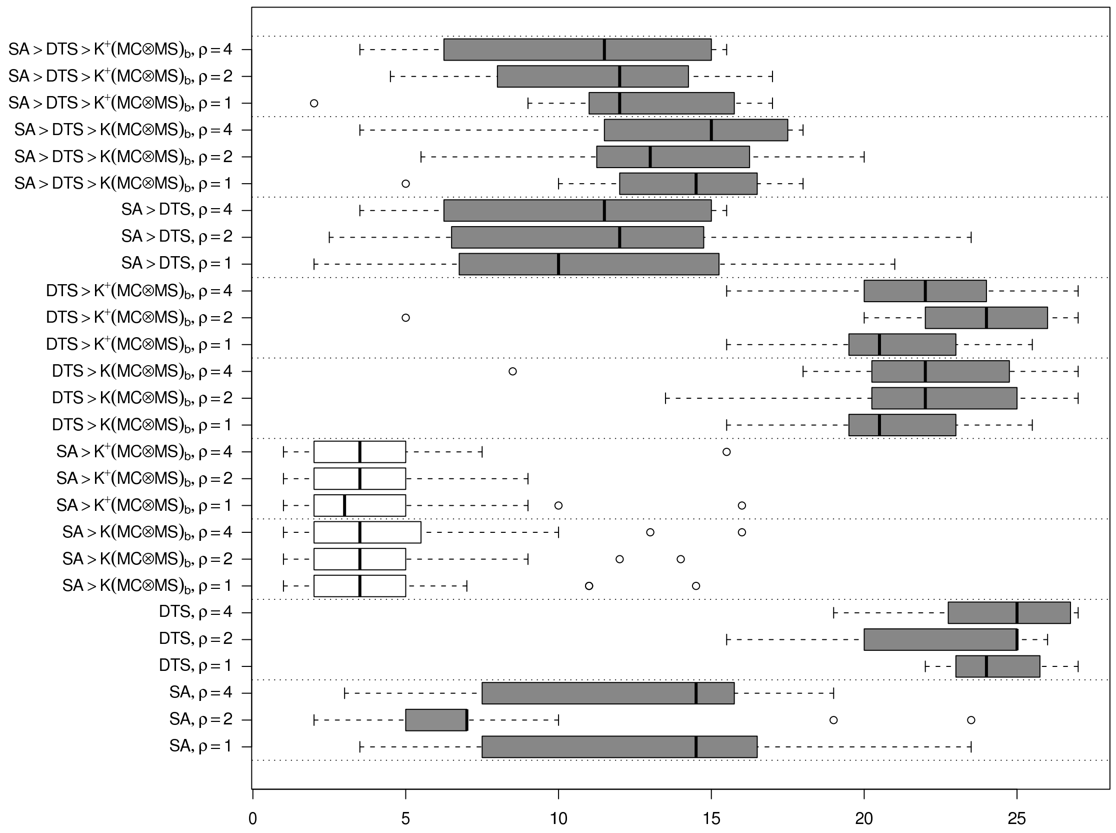


Figure 9: Results of the F-Race selection procedure on the best algorithmic configurations identified in the previous phases.

3 of Section 5.3. These data correspond to the best lower bounds and the best upper bounds found there. The upper bounds are all obtained by 1-norm formulation used as surrogate to solve the 2-norm. In this case we report also the partial result taken at the same time when the local search heuristic is halted, that is, after 320 seconds. It is evident from these results that the heuristics find by large the best upper bounds. Only on instances UD4 and UD6 the solution of the heuristic is not better than the one of the IP solver. Possibly that solution is the optimal solution for that instance.

In addition, we can assess the absolute importance of the results attained with the heuristics by comparing them with the lower bounds and computing the optimality gaps. It seems more appropriate, in this case, to include the square roots in the computations. The results are indicated between round brackets in the table. We believe that these results are good and it will be interesting to monitor their evolution in future research.

	CPLEX					$SA \triangleright K^+(MC \otimes MS)_b$			
Inst.	LB	LB^2	at 320	at 3600	(gap)	best	(gap)	median	(gap)
UD1	10.54	111	952	366	(81.5)	276	(57.6)	284.5	(60.1)
UD2	11.40	130	177	164	(12.3)	148	(6.7)	156.0	(9.5)
UD3	10.30	106	651	214	(42.0)	163	(24.0)	171.5	(27.2)
UD4	18.00	324	396	396	(10.5)	396	(10.5)	396.0	(10.5)
UD5	11.50	3	771	492	(92.8)	219	(28.6)	230.0	(31.8)
UD6	5.92	35	52	52	(21.8)	55	(25.3)	62.5	(33.6)
UD7	10.95	120	x	634	(129.8)	214	(33.5)	240.0	(41.4)
UD8	5.20	27	64	64	(53.9)	46	(30.5)	53.0	(40.1)
UD9	11.66	136	1895	862	(151.7)	223	(28.0)	234.0	(31.1)
UD10	6.16	38	63	48	(12.3)	48	(12.3)	54.50	(19.7)

Table 7: Results of the best local search heuristic. The last four columns report the best and median result of $SA \triangleright K^+(MC \otimes MS)_b$ on 20 trials per instance. The second and third columns report for comparisons the lower and upper bounds of Table 3. The column labelled "at 320" reports the value of the best feasible solution that CPLEX found after 320 seconds, which is the same time limit given to the heuristic. Values in parenthesis are the optimality gaps computed as $100 \cdot (x - LB)/LB$, where x is the square root of the result reported in the column that precedes.

6 Conclusions and Future Work

This article revisited the balanced academic curriculum problem and proposed a new generalized formulation that gets closer to meet all the requirements of a real life situation. It then presented ongoing work for the solution of the generalized version of the problem that turns out to be much harder than the previous formulation. Both integer programming and local search techniques have been studied. The experimental analysis showed that IP is suitable to obtain good LBs above all when the basic formulation is combined with a problem decomposition and relaxation. IP can also reach the best results in a few cases. In the other cases, a suitably configured and tuned local search procedure outperforms IP and is able to provide solutions that range between 60% to 9% from the lower bounds.

There are several directions of future research that span from this work. Firstly, the problem formulation GBACP still contains some limitations with respect to other real life situations. In detail, here are two issues on which we are currently working:

- It might be the case that the same course has to be attended in different years depending on the curricula, while the course, begin taught by the same person, has to be scheduled only once. This possibility would require a more complicated model in which courses are assigned to terms only, and for each actual pair course/curriculum we assign a different year. A way to cope with this issue within the GBACP formulation would be to introduce two different courses one for each curriculum that contains it and then introduce a new type of constraint that force pairs of courses to be scheduled in the same term of the year, while still allowing different periods.

- In the current GBACP instances, some curricula represent different choices among elective courses of the same degree. Nevertheless, some extra lower level alternatives are not included and some of the curricula contain actually more courses than needed by the student to graduate; thus students can drop a few of the courses in the curriculum after these have been scheduled. Including all alternatives in the model by splitting the current curricula would create an explosion in the number of curricula that would simply make the problem intractable. For example, if a student can drop 4 out of 12 courses in a curriculum, this creates $\binom{12}{8} = 495$ different curricula starting from one. Therefore, an alternative way to cope with this issue has to be devised.

However, before introducing further complications, we look forward to a second thread of research departing from here, that is, the run to close the optimality gap on the GBACP instances.

On the side of local search techniques, further, possibly better, GLSM combinations could be studied. In addition, the possibility to obtain fast and good lower bounds by means of the problem decomposition presented opens interesting avenues for specializations of branch and bound algorithms. The local search heuristic could also be embedded in a branch and bound framework and used as primal heuristic.

The problem should be appealing also to the Constraint Programming community from where it originally comes. In particular, it would be interesting to see the work developed by Pesant and Régin (2005); Schaus et al. (2007); Monette et al. (2007) extended to the GBACP. Preliminary experiments conducted in Gecode emphasized a huge increase in the hardness of the GBACP formulation with respect to the BACP. But this attempt did not use any specialized constraint propagator.

Finally, as shown in a recent work (Schaus et al., 2009), the balancing criterion is interesting not only for university timetabling but also for other scheduling applications, such as the allocation of workload to employees or, in that specific example, to nurses. We hope that the models and the techniques discussed in this work may contribute to foster further research also in this context.

References

Ahuja, R., Ergun, Ö., Orlin, J., and Punnen, A. (2002). A survey of very-large-scale neighborhood search techniques. *Discrete Applied Mathematics*, 123:75–102.

Birattari, M., Stützle, T., Paquete, L., and Varrentrapp, K. (2002). A racing algorithm for configuring metaheuristics. In et al., W. B. L., editor, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2002)*, pages 11–18, New York. Morgan Kaufmann Publishers.

Castro, C., Crawford, B., and Monfroy, E. (2007). A quantitative approach for the design of academic curricula. In *HCI*, volume 4558 of *Lecture Notes in Computer Science*, pages 279–288. Springer.

Castro, C. and Manzano, S. (2001). Variable and value ordering when solving balanced academic curriculum problems. In *6th Workshop of the ERCIM WG on Constraints.*

Di Gaspero, L. and Schaerf, A. (2003). EasyLocal++: An object-oriented framework for flexible design of local search algorithms. *Software—Practice and Experience*, 33(8):733–765.

Di Gaspero, L. and Schaerf, A. (2006). Neighborhood portfolio approach for local search applied to timetabling problems. *Journal of Mathematical Modeling and Algorithms.* To appear.

Di Gaspero, L. and Schaerf, A. (2008). Hybrid local search techniques for the generalized balanced academic curriculum problem. In Blesa, M. J., Blum, C., Cotta, C., Fernández, A. J., Gallardo, J. E., Roli, A., and Sampels, M., editors, *Hybrid Metaheuristics*, volume 5296 of *Lecture Notes in Computer Science*, pages 146–157. Springer.

Ehrgott, M. (2005). *Multicriteria optimization.* Lecture Notes in Economics and Mathematical Systems. Springer Berlin Heidelberg, second edition edition.

Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of $\mathcal{NP}$-Completeness.* Freeman, San Francisco, CA, USA.

Gent, I. P. and Walsh, T. (1999). CSPLib: a benchmark library for constraints. Technical report, Technical report APES-09-1999. Available from http://csplib.cs.strath.ac.uk/. A shorter version appears in the Proceedings of the 5th International Conference on Principles and Practices of Constraint Programming (CP-99), pp. 480-481, Springer-Verlag, LNCS 1713, 1999.

Hansen, P. and Mladenović, N. (1999). An introduction to variable neighbourhood search. In Voß, S., Martello, S., Osman, I., and Roucairol, C., editors, *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, pages 433–458. Kluwer Academic Publishers.

Hnich, B., Kızıltan, Z., and Walsh, T. (2002). Modelling a balanced academic curriculum problem. In *CP-AI-OR 2002*, pages 121–131.

Hoos, H. H. and Stützle, T. (2005). *Stochastic Local Search – Foundations and Applications.* Morgan Kaufmann Publishers, San Francisco, CA (USA).

Lambert, T., Castro, C., Monfroy, E., and Saubion, F. (2006). Solving the balanced academic curriculum problem with an hybridization of genetic algorithm and constraint propagation. In *Artificial Intelligence and Soft Computing - ICAISC 2006*, volume 4029 of *Lecture Notes in Computer Science*, pages 410–419. Springer.

Lourenço, H. R., Martin, O., and Stützle, T. (2001). Applying iterated local search to the permutation flow shop problem. In Glover, F. and Kochenberger, G., editors, *Handbook of Metaheuristics*. Kluwer.

Monette, J., Schaus, P., Zampelli, S., Deville, Y., and Dupont, P. (2007). A cp approach to the balanced academic curriculum problem. In Benhamou, B., Choueiry, B., and Hnich, B., editors, *Symcon'07, The Seventh International Workshop on Symmetry and Constraint Satisfaction Problems.*

Pesant, G. and Régin, J.-C. (2005). Spread: A balancing constraint based on statistics. In van Beek, P., editor, *Principles and Practice of Constraint Programming - CP 2005, 11th International Conference, CP 2005, Sitges, Spain, October 1-5, 2005, Proceedings*, volume 3709 of *Lecture Notes in Computer Science*, pages 460–474. Springer.

Schaerf, A. and Di Gaspero, L. (2007). Measurability and reproducibility in timetabling research: Discussion and proposals. In Burke, E. and Rudová, H., editors, *Proc. of the 6th Int. Conf. on the Practice and Theory of Automated Timetabling (PATAT-2006), selected papers*, volume 3867 of *Lecture Notes in Computer Science*, pages 40–49, Berlin-Heidelberg. Springer-Verlag.

Schaus, P., Deville, Y., Dupont, P., and Régin, J.-C. (2007). The deviation constraint. In Hentenryck, P. V. and Wolsey, L. A., editors, *CPAIOR*, volume 4510 of *Lecture Notes in Computer Science*, pages 260–274. Springer.

Schaus, P., Hentenryck, P. V., and Régin, J.-C. (2009). Scalable load balancing in nurse to patient assignment problems. In van Hoeve, W. J. and Hooker, J. N., editors, *CPAIOR*, volume 5547 of *Lecture Notes in Computer Science*, pages 248–262. Springer.