Hybrid Local Search Techniques for the Generalized Balanced Academic Curriculum Problem

Luca Di Gaspero and Andrea Schaerf

DIEGM, University of Udine
via delle Scienze 208, I-33100, Udine, Italy
`l.digaspero@uniud.it`, `schaerf@uniud.it`

Abstract. The Balanced Academic Curriculum Problem (BACP) consists in assigning courses to teaching periods satisfying prerequisites and balancing students' load. BACP is included in CSPLib along with three benchmark instances. However, the BACP formulation in CSPLib is actually simpler than the real problem that, in general, universities have to solve in practice.

In this paper, we propose a generalized formulation of the problem and we study a set of hybrid solution techniques based on high-level control strategies that drive a collection of basic local search components. The result of the study allows us to build a complex combination of simulated annealing, dynamic tabu search and large-neighborhood search. In addition, we present six new instances obtained from our university, which are much larger and more challenging than the CSPLib ones (the latter are always solved to optimality in less than 0.1 seconds by our techniques).

For the sake of possible future comparisons, we make available through the web all the input data, our scores and results, and a solution validator.

1   Introduction

The Balanced Academic Curriculum Problem (BACP) is an assignment problem that arises in universities, and consists in assigning courses to teaching periods satisfying prerequisites and balancing students' load.

A formulation of BACP has been proposed by Castro and Manzano [3], and it has been included in CSPLib [7, prob. 30] along with three benchmark instances.

The BAC problem, in the CSPLib formulation, has been also tackled by Hnich *et al* [9] and Castro *et al* [2], using CP and IP techniques, and by Lambert *et al* [11], using a hybrid techniques composed by genetic algorithms and constraint propagation. In all works, the authors report the finding of a proven-optimal solution for all three instances, although with quite different running times that range from less than one second to hundreds of seconds.

The BACP formulation is actually simpler than the real problem that universities have to solve in practice, at least for the cases we are aware of. In this

paper, we try to overcome this limitation, and we define a more complex formulation, which applies, among others, to our institution (School of Engineering of University of Udine).

We study the application of general high-level local search strategies and we develop a hybrid solution technique based on a complex combination of simulated annealing, dynamic tabu search and large-neighborhood search. We propose six new instances obtained from our university. Our instances are much larger and (for our techniques) more challenging than the CSPLib ones, and their optimal value is not known. Furthermore, they show different structures, as they represent cases very different from each other.

We report our experimental analysis for these new instances. In addition, given that the formulation proposed here is actually a generalization of the BACP one, we have been able to test our solver also on the CSPLib instances. The outcome has been that all three CSPLib instances are solved to optimality in less than 0.1 seconds in almost all runs of our solvers. This proves that, on those simpler cases, the solvers' performances are comparable or better than state of the art solutions.

For the general problem and for the larger instances, obviously we have no "competitors" to compare to, besides the variants implemented by ourselves. Nevertheless, for the sake of possible future comparisons, we make available through the web (`http://www.diegm.uniud.it/satt/projects/bacp/`) all the input data, our results, and also the source code of the solution *validator*.

The validator is a simple program that takes two command-line parameters, an instance and a solution, and returns both a detailed list of the violations and a summary of the costs. As we discussed in detail in [13], we believe that the publication of the validator is indeed necessary when proposing a new problem, so as to prevent possible misunderstandings on the details of the formulation, and thus to provide against the publication of incorrect results.

2   Problem Formulations

We first present here the basic BACP formulation as proposed in [3]. Later we discuss the extensions that we have added for dealing with our real-world problem.

2.1   BACP Formulation

The basic formulation consists of the following entities and constraints:

Courses: Each course has an integer number of *credits*, and it has to be taught during the planning horizon of the university degree.

Periods: The planning horizon is composed by a given number of *teaching periods* that have to be assigned to courses. Periods are divided in years, and each year is divided in a fixed number of terms. For example, a 3-year degree organized in four trimesters per year has 12 periods.

Load limits: For each period there is a minimum and a maximum number of courses that can be assigned to it. Further, there are minimum and maximum limits also on the number of total credits per period.

Prerequisites: Based on their content, some courses have to be taught before other courses. This means that we are given a set of pairs of courses, such that the period assigned to the first course has to be strictly less than the period assigned to the second. Obviously, prerequisite relation is transitive and cannot contain cycles.

The problem consists in finding an assignment of courses to periods that satisfies all above (hard) constraints: load limits and prerequisites. The objective function accounts for the balancing of credits in periods. In detail, the objective function (to be minimized) used in the cited papers 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.

2.2   GBACP Formulation

Given the above basic formulation as starting point, we extend it in the following directions.

Curricula: First of all, in the formulation it is implicitly assumed that a student takes all courses without personal choices, whereas in practice a student can select among alternatives. A *curriculum* is a set of courses representing a possible complete selection of a student. For each single curriculum, courses have to be balanced and limited in number. We extend the problem by considering many curricula, which may share some of the courses.

Preferences: Professors can express preferences about their teaching periods. Specifically, a teacher expresses his/her preferences for a specific term of the year but not for the year. A preference of a course for a given term results in a penalty for any assignment of the course to a period which is not in that term. Preferences are not strict, and therefore they contribute to the objective function (soft constraints).

The objective function is thus a composition of preference violations and unbalanced load. In order to adhere more precisely to the real situation, the load component that we consider is not based only on the maximum load, but it sums up all the deviations (positive and negative) from the average number of credits per period for each curriculum.

More precisely, for each curriculum we compute α as the total number of credits of a curriculum divided by the number of periods (not necessarily integer-valued). A number of credits per period equal to $\lfloor \alpha \rfloor$ or $\lceil \alpha \rceil$ has penalty 0. All values below $\lfloor \alpha \rfloor$ or above $\lceil \alpha \rceil$ are penalized. In order to avoid large discrepancies

from α, the penalty is quadratic. That is, a deviation of 1, 2, or 3 counts as 1, 4, and 9 points of penalty respectively (and so on for larger values).

Load limits are evaluated for each single curriculum and summed up. Conversely, prerequisites remain expressed at the global level. In order to have a single objective function, we assign to each violation of a preference 5 points of penalty.

As mentioned above, the original formulation includes limits per period in terms of both number of courses and total credits. However, the quadratic penalty of credit balancing makes the presence of hard limits in the number of credits meaningless in our formulation, because large discrepancies are already prevented. Therefore, we remove this constraint. Conversely, the limits on the number of courses instead are maintained because they helps in avoiding extreme situations in terms of the number of final exams (independently of credits), which is desirable in our institution.

We call the corresponding problem GBACP (G for Generalized). Notice that a 0 cost solution in our GBACP formulation, costs $\lceil \alpha \rceil$ in the original BACP and it is thus optimal.

The opposite is not necessarily true. For example, for the CSPLib instance `bacp8`, which has (for its single curriculum) $\alpha = 16.625$, a solution with 15, 16 and 17 credits per period has optimal value (17) for the basic formulation, but the periods with 15 credits are penalized 1 in our formulation.

3   Local Search for GBACP

Local search is a family of general-purpose techniques for search and optimization problems, which are based on an iterative process of navigating a search space stepping from one solution to a neighboring one. The process is guided by a cost function, which drives the search toward good solutions.

In order to apply local search to GBACP we have to define several features. We first illustrate the search space and the procedure for computing the initial state. Then, we define the neighborhood structure, and finally we describe the set of search components employed and the high-level strategies for combining them.

3.1   Search Space, Initial Solution, and Neighborhood Relation

Based on prerequisites, we can infer some infeasible assignment by constraint propagation. For example, if course a must be before course b, then course a cannot be assigned to the last period, and course b cannot assigned to the first.

We therefore compute the transitive closure of the prerequisites, and based on it we determine a minimum and a maximum starting period for each course as its assignable range. The search space is then defined by all possible assignments of a period in the above range to each course.

Assignments that violates hard constraints (load limits and prerequisites) are also considered, and the violations (*distance to feasibility*) are included in the cost function of the search with a higher weight w.r.t. the objective function.
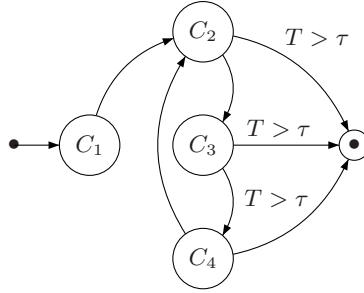
Fig. 1. An example of Generalized Local Search Machine with three search components

The initial solution is generated in a totally random way: each course is assigned a uniform random period in its assignable range.

The neighborhood relation consists in moving one course from its period to another one (in its range). A move is therefore identified by two attributes, namely the course and the new period.

3.2   Search Techniques

In this work we study a set of high-level search control strategies that hybridize several basic local search components. This idea is an instantiation of Hoos and Stützle's *Generalized Local Search Machines* (GLSM) [10, Chapter 3], which is a formal framework for describing search control by clearly separating it from the search components. In this framework, the basic search components are represented as states (i.e., nodes) of a Finite State Machine, whereas the transitions (i.e., edges) correspond to conditions for modeling the search control. Within the GLSM framework it is possible to specify also complex strategies such as VNS [8] and ILS [12].

In our terminology we refer to the states of the machine as *runners*, that are basic local search metaheuristics, or *kickers*, that are perturbation components represented by a single move in a large neighborhood used either for intensification or diversification purposes.

In Figure 1 we show an example of a GLSM with three search components. The search starts from C_1 and when this component has finished it unconditionally passes its solution to component C_2. Component C_2 continues the search, followed by C_3 and afterward C_4. Then the process is started again from C_2. The whole strategy is stopped when an overall timeout τ has expired.

3.3   Search Components for GBACP

The search components employed in this study range from very trivial strategies to more complex meta-heuristics. In detail, we consider the following ones (see [10] for a detailed description):

Steepest Descent (SD): at each step of the search the whole neighborhood is explored and the *best* improving move is performed. The search is stopped when a local minimum is reached.

First Descent (FD): differently from SD, this technique interrupts the exploration the neighborhood as soon as the *first* improving neighbor has been found. Like SD, The search is stopped when a local minimum is reached.

Randomized Hill Climbing (RHC): at each step of the search a random neighbor is selected. The move is performed only if the cost of the neighbor is less *or equal* than the current solution cost.

Simulated Annealing (SA): similarly to RHC a random neighbor is selected at each step. The move is performed either if it is an improving one or according to an exponential time-decreasing probability. In detail, if the cost of the move is $\Delta f > 0$, the move is accepted with probability $e^{-\Delta f/T}$, where T is a time-decreasing parameter called *temperature*. At each temperature level a number σ_N of neighbors of the current solution is sampled and the new solution is accepted according the above mentioned probability distribution. Afterward the value of T is modified using a *geometric* schedule, i.e., $T' = \beta \cdot T$, in which the parameter $\beta < 1$ is called the *cooling rate*.

Tabu Search (TS): at each step a subset of the neighborhood is explored and the neighbor that gives the minimum 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 [10]), so that a move is kept in the tabu list for a random number of iterations in the range $[k_{min}..k_{max}]$.

Dynamic Tabu Search (DTS): the search strategy is the same as TS, however this 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 (the weight is reduced by a factor $\gamma > 1$, i.e., $w' = w/\gamma$) in order to allow the exploration of solutions where those constraints do not hold. Conversely, if some constraint is not satisfied, it is tighten ($w' = w \cdot \gamma$) with the aim of driving the search toward its satisfaction.

Kickers (K): as explained in [6], kickers are special-purpose components that perform just one single step in a composite neighborhood (i.e., a sequence of moves of arbitrary length). 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. Random kicks can be used for diversification purposes (thus giving raise to the Iterated Local Search strategy [12]), while first and best kicks are employed for intensifying the search.

3.4   GLSM Templates

The search components studied in this work are combined by means of the GLSM templates shown in Figure 2. These machines represent a set of fundamental strategies for the cooperation of search components.

The *multi-start* search (Fig. 2a) consists in the repetition of the generation of a starting solution, e.g., through a random assignment G_r, followed by the run of a basic meta-heuristic. The *multi-run* strategy (Fig. 2b), instead, repeats only the run of the meta-heuristic from the best solution found that far. These two strategies are simple strategies whose rationale is respectively to enhance diversification and intensification of a single runner at a coarse grain of granularity. The strategies are denoted in the following by $\mathrm{MS}(R)$ and $\mathrm{MR}(R)$, respectively.

The other two strategies, instead, include intensification/diversification during the high-level search process, according to the types of search components employed. The token-ring strategy (Fig. 2c) executes in sequence a set of basic meta-heuristic $R_1, R_2, \ldots$, followed by another search component that can either be a *kicker*, giving raise to what we call *run and kick* strategy (which is similar to Iterated Local Search [12]), or another *runner* (see [6] for more details). At the end of the process, the sequence is started again from R_1. The last template (Fig. 2d) is similar to the previous one, the only difference is that more emphasis is put on intensification, since the last component is a kicker that it can be iterated while it keeps improving the solution, thus performing a Large Neighborhood Search (LNS) [1]. Even though the difference between these two templates might seem minimal, both deserve a study since the computational cost related to the LNS might favor the simpler run and kick strategy when the running time granted to the solver is limited. We denote with the symbol ▷ the token-ring sequence and with a superscript plus ($^+$) sign the LNS.
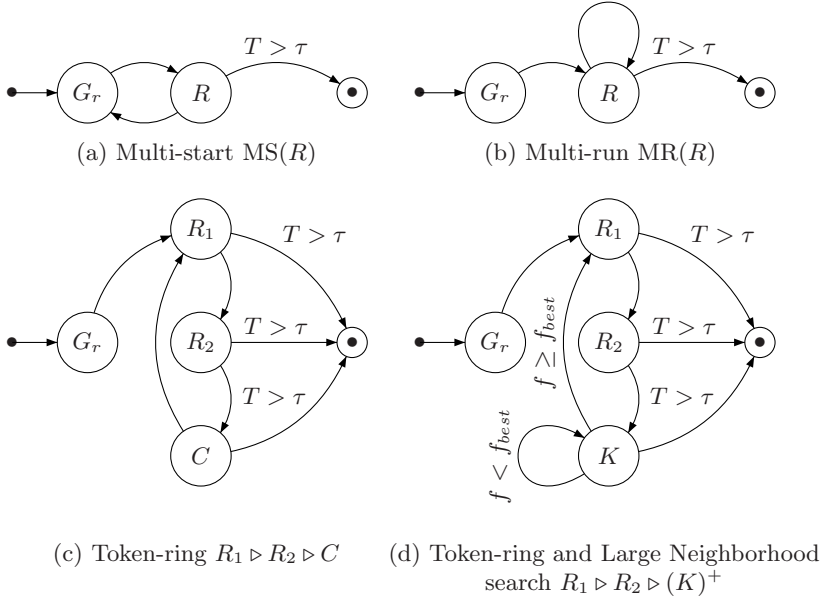
4   Experimental Analysis

In this section, we first introduce the benchmark instance and the general settings of our analysis, and then we move to the experimental results.

4.1   Benchmark Instances

In order to accommodate the additional information that we need, we have to modify the input file format of BACP. Rather than extending it, we have decide to define a completely new format for GBACP, along the lines of the `ctt` format defined in [4] for the timetabling problem defined for the International Timetabling Competition (ITC-2007) track 3. This new format, explained in the website, is in our opinion more easily parsable using any programming language than the one used in CSPLib.

Six new instances, called UD1–UD6, have been extracted from the database of our university. Actually, given that the database contains historical data, many more could be created, but they would have been quite similar to these ones, therefore we decided to keep only the structurally different ones. We have also

(a) Multi-start MS(R)   (b) Multi-run MR(R)

(c) Token-ring $R_1 \triangleright R_2 \triangleright C$ (d) Token-ring and Large Neighborhood
                                              search $R_1 \triangleright R_2 \triangleright (K)^+$

Fig. 2. The GLSM templates studied

Table 1. Features of the instances

Instance	Courses	Periods (Years × Terms)	Curricula	Prerequisites	Preferences
bcap8	46	8 (4 × 2)	1	33	0
bacp10	42	10 (5 × 2)	1	33	0
bacp12	66	12 (6 × 2)	1	65	0
UD1	307	9 (3 × 3)	37	1383	90
UD2	268	6 (2 × 3)	20	174	79
UD3	236	9 (3 × 3)	31	1092	66
UD4	139	6 (2 × 3)	16	188	40
UD5	282	6 (3 × 2)	31	397	54
UD6	264	4 (2 × 2)	20	70	55

translated the three CSPLib instances into our GBACP format, summing up to
9 cases to experiment on.

Table 1 summarizes the main features of the instances. All instances are avail-
able from the web, along with the format description, our best solutions, and
the C++ source code of the validator that *certifies* their scores.

The format of the solution file is simply the sequence of the periods assigned
to the courses listed in the same order of the input file.

4.2   General Settings and Implementation

All the algorithms have been implemented in the C++ language, exploiting
the EASYLOCAL++ framework [5]. EASYLOCAL++ is a tool for local search

that provides the full control structures of the algorithms, which, in its current version, also supports a limited implementation of GLSM.

The experiments were performed on an Intel QuadCore PC (64 bit) running Debian Linux 4.0, the software is compiled using the GNU C++ compiler (v. 4.1.2).

In order to compare different techniques in a fair way, we decided to grant to all of them the same total time. That is, the only condition to exit the GLSM is the timeout expired. The timeout is set to 60 seconds of CPU time on the above described PC.

The parameters of the basic components have been tuned according to the results of preliminary experiments on the single search components. The values are reported in Table 2, where c denotes the number of courses in the input instance and $N(s_0)$ is the neighborhood of solution s_0.

Table 2. Parameter settings of the algorithms

Algorithm	Parameter	Value
Dynamic Tabu Search	k_{min}	$c/4$
	k_{max}	$c/4 + 15$
	γ	1.06
Simulated Annealing	T_{start}	$\max_{s \in N(s_0)} \Delta f$
	β	0.99
	σ_N	2000

4.3   Experimental Results

We first report the results of our basic strategies on the CSPLib instances. For these "easy" instances we report only the outcome of single runs. In detail, Table 3 reports (for 1000 runs) the percentage of times an optimal (0 cost) solution has been reached and the average running time (in seconds) of all runs. The running times reported are the total one, i.e., including reading/writing files and preprocessing.

The table shows that non-trivial techniques (SA, TS, and, DTS) solve all three instances to optimality very quickly on almost all runs, whereas the others also solve them quickly but with much less success rate. A more detailed analysis of the differences of the techniques is performed on the more challenging instances.

Table 3. Results for CSPLib instances: values are success rate and average number of seconds to reach a feasible solution

Instance	SD %succ	SD time	FD %succ	FD time	RHC %succ	RHC time	SA %succ	SA time	TS %succ	TS time	DTS %succ	DTS time
bacp8	28.3	.0006	19.6	.0008	49.5	.1450	100.0	.0042	100.0	.0023	100.0	.0026
bacp10	6.2	.0009	2.2	.0012	33.3	.1906	100.0	.0429	100.0	.0046	100.0	.0060
bacp12	1.3	.0033	0.2	.0040	9.0	.2775	97.9	.1764	98.9	.0459	96.7	.08426

Regarding previous work, the best results in the literature are those in [9] which report a running time of 0.29, 0.59, and 1.09 secs to find to optimal solution for `bacp8`, `bacp10`, and `bacp12`, respectively. Our running times are somewhat better, however the absolute values are both small, and we cannot reach any conclusion on the comparison. But at least, these results confirm that local search techniques are indeed suitable to solve the BACP problem.

Moving to the analysis of the GLSM strategies described in Section 3.4, we perform a two step analysis. First we run the simple strategies equipped with all the search components to identify the most promising ones. Then we try to improve the performances of the selected search components by equipping the complex strategies with them.

Notice that not all combinations of simple search strategies and search components are meaningful. For example the multi run applied to the SD and FD algorithm is completely useless, since this algorithms stop as soon they got stuck in a local minimum.

The results are shown in Figure 3. In the plots it has been reported the normalized cost found by each algorithm on all the instances UD1–UD6, the data refers to 80 repetitions of each algorithm with different random seeds. The results of the runs are normalized with respect to the value of the best known cost of each instance, i.e., $f_{norm} = f/f_{best,i}$. Moreover we employ a semi-logarithmic scale on the y-axis, to enhance the differences between the different methods.

From the picture in Figure 3a we can see that, in general, the multi start strategy performs slightly better than the multi run ones. This can be explained by a early stagnation of the search, which prompts for a strong diversification of the search. Regardless of the search strategy employed, the best results for the simple strategies are obtained by the SA and DTS search components.

In the second step of the analysis we equip the complex strategies with the best algorithms found in the previous step, i.e., the SA and DTS search components. We test those algorithms followed by a kicker of length 2.

(a) Simple strategies   (b) Complex strategies
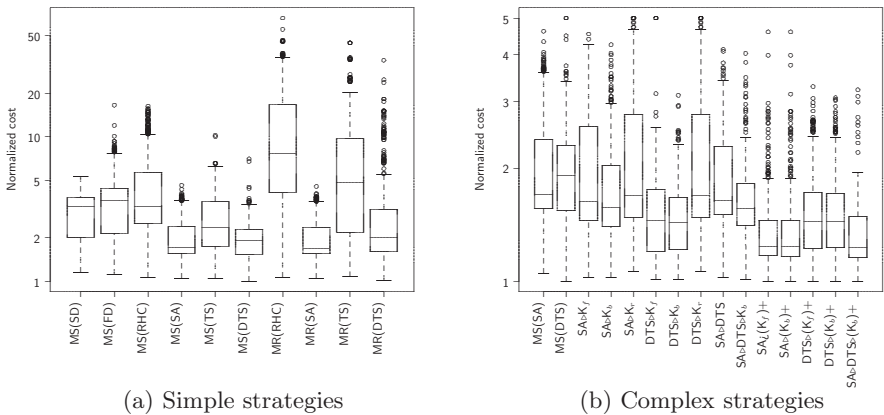
Fig. 3. Results of simple and complex strategies on instances UD1–UD6

Concerning the results for the complex strategies it is worth to notice that, in general, their results are not worse than those achieved by the simple ones (the two leftmost boxes in Fig. 3b). However, it seems that the complex strategies fits better with an intensification mechanism, e.g., DTS with both strategies or SA with the token ring followed by LNS. Moreover, the combination that gives the best performances is SA$\triangleright$DTS$\triangleright(K_b)^+$, which combines intensification and diversification steps in a quite complex way.

5   Conclusions and Future Work

This paper reports on an ongoing work for the solution of the GBAC problem by means of hybrid local search techniques. The preliminary results show that, although local search techniques are in general effective to solve this kind of problems, there is need to devise complex combinations of techniques to obtain better results.

In the future we plan to explore more systematically the possible GLSM combinations, in order to obtain a clearer picture of the importance of the hybridization.

Regarding the problem formulation, the GBACP proposed here seems to be adequate to solve our practical problem. Nevertheless, it still contains some limitations, that we plan to further investigate in future work. In detail, there are two main issues to address:

- Although not advisable, it is possible that a course is taken in different years in different curricula. 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.
- Different curricula in some cases represent free alternatives of the same degree, nevertheless some extra lower level alternatives are not modeled in the formulation. In fact, some of the curricula contain actually more courses than needed by the student to graduate, and they can drop a few of them among a larger list.

Notice that the second one could be removed simply including all alternatives in the model by splitting the current curricula. This solution however would create an explosion in the number of curricula. 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, we need to devise some techniques to manage a large number of similar curricula.

Acknowledgment

References

1. Ahuja, R., Ergun, Ö., Orlin, J., Punnen, A.: A survey of very-large-scale neighbor-hood search techniques. Discrete Applied Mathematics 123, 75–102 (2002)
2. Castro, C., Crawford, B., Monfroy, E.: A quantitative approach for the design of academic curricula. In: Smith, M.J., Salvendy, G. (eds.) HCII 2007. LNCS, vol. 4558, pp. 279–288. Springer, Heidelberg (2007)
3. Castro, C., Manzano, S.: Variable and value ordering when solving balanced academic curriculum problems. In: 6th Workshop of the ERCIM WG on Constraints (2001)
4. Di Gaspero, L., McCollum, B., Schaerf, A.: The second international timetabling competition (ITC-,: Curriculum-based course timetabling (track 3). Technical Report QUB/IEEE/Tech/ITC2007/CurriculumCTT/v1.0/1, School of Electronics, Electrical Engineering and Computer Science, Queens University, Belfast (UK). ITC-2007 (August 2007), `http://www.cs.qub.ac.uk/itc2007/`
5. Di Gaspero, L., Schaerf, A.: EasyLocal++: An object-oriented framework for flex-ible design of local search algorithms. Software—Practice and Experience 33(8), 733–765 (2003)
6. Di Gaspero, L., Schaerf, A.: Neighborhood portfolio approach for local search applied to timetabling problems. Journal of Mathematical Modeling and Algo-rithms 5(1), 65–89 (2006)
7. Gent, I.P., Walsh, T.: CSPLib: a benchmark library for constraints. Technical re-port, Technical report APES-09-1999 (1999), `http://csplib.cs.strath.ac.uk/`; A shorter version appears. In: The Proceedings of the 5th International Confer-ence on Principles and Practices of Constraint Programming (CP 1999). LNCS, vol. 1713, pp. 480–481. Springer, Heidelberg (1999)
8. Hansen, P., Mladenović, N.: An introduction to variable neighbourhood search. In: Voß, S., Martello, S., Osman, I.H., Roucairol, C. (eds.) Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization, pp. 433–458. Kluwer Academic Publishers, Dordrecht (1999)
9. Hnich, B., Kızıltan, Z., Walsh, T.: Modelling a balanced academic curriculum prob-lem. In: CP-AI-OR 2002, pp. 121–131 (2002)
10. Hoos, H.H., Stützle, T.: Stochastic Local Search – Foundations and Applications. Morgan Kaufmann Publishers, San Francisco (2005)
11. Lambert, T., Castro, C., Monfroy, E., Saubion, F.: Solving the balanced academic curriculum problem with an hybridization of genetic algorithm and constraint prop-agation. In: Rutkowski, L., Tadeusiewicz, R., Zadeh, L.A., Żurada, J.M. (eds.) ICAISC 2006. LNCS (LNAI), vol. 4029, pp. 410–419. Springer, Heidelberg (2006)
12. Lourenço, H.R., Martin, O., Stützle, T.: Applying iterated local search to the permutation flow shop problem. In: Glover, F., Kochenberger, G. (eds.) Handbook of Metaheuristics. Kluwer, Dordrecht (2001)
13. Schaerf, A., Di Gaspero, L.: Measurability and reproducibility in timetabling re-search: Discussion and proposals. In: Burke, E., Rudová, H. (eds.) PATAT 2007. LNCS, vol. 3867, pp. 40–49. Springer, Heidelberg (2007)