# Iterated VND Versus Hyper-heuristics: Effective and General Approaches to Course Timetabling

**Jorge A. Soria-Alcaraz, Gabriela Ochoa, Marco A. Sotelo-Figueroa, Martín Carpio and Hector Puga**

**Abstract** The course timetabling problem is one of the most difficult combinatorial problems, it requires the assignment of a fixed number of subjects into a number of time slots minimizing the number of student conflicts. This article presents a comparison between state-of-the-art hyper-heuristics and a newly proposed iterated variable neighborhood descent algorithm when solving the course timetabling problem. Our formulation can be seen as an adaptive iterated local search algorithm that combines several move operators in the improvement stage. Our improvement stage not only uses several neighborhoods, but it also incorporates state-of-the-art reinforcement learning mechanisms to adaptively select them on the fly. Our approach substitutes the adaptive improvement stage by a variable neighborhood descent (VND) algorithm. VND is an ingredient of the more general variable neighborhood search (VNS), a powerful metaheuristic that systematically exploits the idea of neighborhood change. This leads to a more effective search process according course timetabling benchmark results.

**Keywords** Course timetabling · Iterated local search · Variable neighborhood descend · Hyper-heuristics

J.A. Soria-Alcaraz (✉) · M.A. Sotelo-Figueroa
Division de Ciencias Economico-Administrativas, Departamento de Estudios Organizacionales, Universidad de Guanajuato, Leon, Mexico
e-mail: jorge.soria@ugto.mx

M.A. Sotelo-Figueroa
e-mail: masotelof@ugto.mx

G. Ochoa
Department of Computer Science and Mathematics,
University of Stirling, Striling, UK
e-mail: gabriela.ochoa@cs.stir.ac.uk

M. Carpio · H. Puga
División de Estudios de Posgrado e Investigacion,
Instituto Tecnológico de León, León Guanajuato, León, México
e-mail: jmcarpio61@hotmail.com

H. Puga
e-mail: pugahector@yahoo.com

## 1 Introduction

The timetabling problem is a common and recurring problem in many organizations. This paper focuses on a variant of the timetabling problem, named Course Timetabling Problem (CTTP). The CTTP, commonly seen at every university, works with the assignment of a fixed number of events into a number of timeslots. The main objective of this problem is to obtain a timetable that minimizes the number of conflicts for a student [9]. Many possible conflict types exist in CTTP, but the principal conflicts are usually time-related i.e. one student with two or more subjects assigned to the same time period.

Like most timetabling problems, the CTTP is known to be NP-Complete [10, 26]. Due to this complexity and the fact that course timetables are still often constructed by hand, it is necessary to automate timetable construction in order to improve upon the solutions reached by the human expert [25]. Unfortunately, the automation of a timetable construction is not an easy task, it requires a deep knowledge of the problem. This is the principal reason for the high popularity of hyper/metaheuristic solvers for the CTTP. Hyper-heuristics represents a novel direction on the heuristic optimization field, these algorithms aim to provide generic frameworks to solve differents problems. The advantage of this approach is that once we had a hyper-heuristic algorithm with reasonably good performance, that algorithm could be applied to similar problems without more changes that the update of a pool of low level (and problem dependent) operators. Hyper-heuristics can be defined as (meta) heuristics that choose or generate a set of low level heuristics to solve difficult search and optimization problems [4]. Hyper-heuristics aim to replace bespoke approaches by more general methodologies with the goal of reducing the expertise required to solve a problem [22]. In most of the previous studies on hyper-heuristics, low-level heuristics are uniform, i.e. they are either constructive or perturbative (improvement) heuristics [4, 6]. We use perturbative heuristics only.

Hyper-heuristics can be designed to select or construct heuristics when solving a problem. In this work we use hyper-heuristics that select the most promising heuristics to guide the search process. In order to create this kind of hyper-heuristics, a high level heuristic chooser is needed. In this work we use as high level chooser a hybrid ILS-VND algorithm. Iterated local search (ILS) is a simple but successful algorithm [13]. It operates by iteratively alternating between applying an operator to the incumbent solution and restarting local search from a perturbed solution. Variable neighborhood descend (VND) is a powerful component of variable neighborhood search (VNS), this simple strategy applies a set of hierarchically ordered heuristics to an incumbent solution, expecting that less perturbative heuristics are used more frequently than complex operators. The main contribution of this work is a newly proposed iterated variable neighborhood descent algorithm as a high level heuristic chooser when solving the course timetabling problem.

The paper is organized as follows. Section 2 defines the problem and some important concepts. Section 3 presents the solution approach and its justification. Section 4 contains the experimental set-up, results, their analysis and discussion. Finally, Sect. 5 gives the conclusions and describes some future work.

## 2 Main Concepts

In this section, the generic representation (methodology of design) is briefly explained. We also give a brief description of hyper-heuristic algorithms and high level choosers. Our base algorithms: ILS and VND are also detailed here.

### *2.1 Problem Definition*

The CTTP, part of the Constraint Satisfaction Problems (CSP), considers its main variables as events to be scheduled according to a given curricula and its constraints as time-related restrictions (e.g., specific events to be assigned into specific timeslots). CTTP formulation consist of several sets [9]: a set of events (courses or subjects), $E = \{e1, e2, \ldots, en\}$ are the basic element of a CTTP. In addition, there are a set of time-periods, $T = \{t1, t2, \ldots, ts\}$, a set of places (classrooms) $P = \{p1, p2, \ldots, pm\}$, and a set of agents (students registered in the courses) $A = \{a1, a2, \ldots, ao\}$. Each member $e \in E$ is a unique event that requires the assignment of a period of time $t \in T$, a place $p \in P$ and a set of students $S \subseteq A$, so that an assignment is a quadruple $(e, t, p, S)$. A timetabling solution is a complete set of $n$ assignments, one for each event, which satisfies the set of hard constraints defined by each university or college. This problem is known to be at least NP-complete [10, 19].

The CTTP has been studied intensively, from early solution approaches based on logic programming [2, 5, 14] to metaheuristic schemes such as Tabu-list [16], Genetic Algorithms [28], Ant Colony [17], PSO [12], Variable Neighborhood Search [3] and Bee Algorithms [20]. In the same way, a great number of surveys of metaheuristics solution schemes that have been used to solve the CTTP problem are available [7, 8, 15, 18]. In addition, in recent years Hyper-heuristic frameworks has been applied with encouraging results [22, 23]. Almost all of this research used the ITC2007 [1] important benchmark in order to gather evidence about the efficiency of each proposed approach. This paper takes hyper-heuristics from the state of the art and applies them to ITC2007 instances.

### *2.2 Methodology of Design for the Course Timetabling Problem*

In the literature it can be seen that a problem with the diversity of course timetabling instances exists due different university policies. This situation directly impacts on the reproducibility and comparison between course timetabling algorithms [21]. The state of the art indicates some strategies to avoid this problem; for example, a more formal problem formulation [15], as well as the construction of benchmark instances [1]. These schemes are useful for a deeper understanding of the university timetabling complexity, but the portability and the reproducibility of a timetabling solver in another educational institution is still in discussion [21]. In this sense, we use a context-independent layer for the course timetabling solution representation. This layer, named 'Methodology of Design', has been used previously with encouraging results for CTTP [23]. In addition it has been defined formally on Soria-Alcaraz et al. [24].

When using this layer, a heuristic algorithm only needs to choose a pair (timeslot–classroom) for each event $e$ from a previously defined structure, as seen on Fig. 1. The construction of this structure of valid assignations exceeds the scope of this paper; for further information, please consult [22–24].

The solution representation used by any heuristic algorithm that uses this methodology can therefore be seen as an integer array with length equal to the number of variables (events/subjects). The objective is then to search in this space for a solution that minimizes student conflicts given by Eqs. (1) and (2). One example of this representation can be seen in Fig. 1.

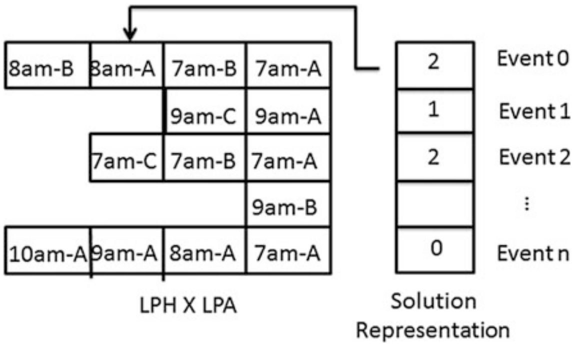
$$\mathrm{Min}(\mathrm{FA}) = \sum_{i=1}^{k} \mathrm{FA}_{V_i} \tag{1}$$

$$\mathrm{FA}_{V_j} = \sum_{s=1}^{(M_{V_j})-1} \sum_{l=1}^{M_{V_i}-s} \left(A_{j,s} \wedge A_{j,s+l}\right) \tag{2}$$

where

| | |
|---|---|
| FA | Student conflicts of current timetabling. |
| $V_i$ | Student conflicts from "Time slot" $i$ of the current Timetabling. |
| $A_{j,s} \wedge A_{j,s+l}$ | students that simultaneously demand subjects $s$ and $s+1$ inside the timeslot $j$. |
| $A$ | student that demands subject $s$ in timetabling $j$. |

**Fig. 1** Representation used



## 2.3 Hyper-heuristics

The term "hyper-heuristics" is relatively new, having first appeared as a strategy to combine artificial intelligence methods. The un-hyphenated version of the term initially appeared in Burke et al. [4] describing hyper-heuristics as "*heuristics to choose heuristics*" in the context of combinatorial optimization. Hyper-heuristics can be designed in a wide variety of ways, but in general they are constructed to achieve two objectives [4]: *heuristic selection* or *heuristic generation*. This work focuses on heuristic selection. Selection hyper-heuristics generally use a meta-heuristic to choose, from a pool of low-level heuristics, the best possible heuristic/operator to be applied to a problem instance to guide the search effectively. The basic scheme for the selection of hyper-heuristics can be seen in Fig. 2.

The high level heuristic chooser must decide what heuristic is the most promising one to continue the search at a given point. Obviously these algorithms need to have some desired characteristics including a fast and consistent response to changes in the current state of the problem, a mechanism to select heuristics previously not common selected and a mechanism to identify whenever the current operator has more potential to be discover by further applications. The domain barrier prevents the high level chooser from being problem dependent, i.e. having a low level knowledge of the problem. In practice, this means that the high level chooser only knows the last fitness improvement product of the last heuristic applied. This domain barrier ensures the generality of the whole model. Low level heuristics are another important part of the hyper-heuristic framework, this pool of operators receive a signal from the high level chooser whenever they must in order to produce a change in the current state of the problem instance and they return the fitness improvement of its application (if any). Finally, the problem instance state changes dynamically thought the successive application of low level heuristics to an optimal state.

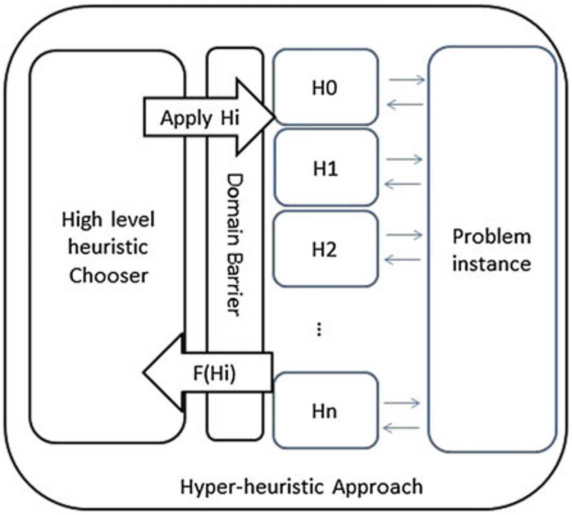


**Fig. 2** Selection of hyper-heuristic

### 2.4 Iterated Local Search High Level Chooser

Iterated local search is a relatively simple yet powerful strategy. It operates by iteratively alternating between applying a move operator to the incumbent solution (perturbation stage) and restarting the local search from the perturbed solution (improvement stage). The term *iterated local search* (ILS) was proposed in [13].

A number of adaptive variants of multi-neighborhood iterated local search have been recently proposed with encouraging results in other problem domains [27]. If several options are available for conducting perturbation and improvement, a mechanism needs to be provided to choose between them. The idea is to use online learning to adaptively select the operators either at the perturbation stage or the improvement stage, or both. These approaches inspired the algorithm implemented in this article, which can be seen in Algorithm 1 and 2. In this implementation, the perturbation stage (step **4** in Algorithm 1) applies a single fixed move operator to the incumbent solution. This move operator (Simple Random Perturbation) simply selects uniformly at random a single variable and substitutes it for another variable in the range selected uniformly at random. Learning is then applied to the improvement stage (Algorithm 2), in which a perturbation operator is applied to the incumbent solution (steps 4 and 5 in Algorithm 2).

**Algorithm 1** Iterated Local Search (ILS)

```
1: S0= GenerateIntialSolution()
2: S*= ImprovementStage(S0)
3: while !StopCriteria() do
4:     S' = SimpleRandomPerturbation(S*)
5:     S*' = ImprovementStage(S')
6:if f(S*')  better than f(S*) then
7:          S* = S*'
8:        end if
9:end while
11: return S*
```

**Algorithm 2** Improvement Stage

```
1: ls = IncumbentSolution()
2: S*= ImprovementStage(S0)
3: while !LocalStopCriteria() do
4:     hi = Select Heuristic()
5:     ls* = apply(hi,ls)
6:     if f(ls*) better than f(ls) then
7:          ls = ls*
8:        end if
9:  end while
11: return ls
```

## *2.5 Variable Neighborhood Descent*

Variable neighborhood search (VNS), developed by Hansen and Mladenovic [11] in 1996, is a metaheuristic for solving combinatorial and global optimization problems. Unlike many metaheuristics where only a single operator is employed, VNS systematically changes different operators within a local search. The idea is that a local optimum defined by one neighborhood structure is not necessarily the local optimum of another neighborhood structure, thus the search can systematically traverse different search spaces, which are defined by different neighborhood structures. This potentially leads to better solutions, which are difficult to obtain by using single neighborhood based local search algorithms [11]. The basic principles of VNS are simple in execution, since parameters are kept to a minimum. Our high level chooser is based on variable neighborhood descent search (VND), a variant of the VNS algorithm. VND algorithm differs from VNS in that VND needs a hierarchically ordered list of operators; this ordering is commonly made by putting simpler and less perturbative heuristics at the beginning of the ordering, so more complex and perturbative heuristics appear latter in the ordering. The main idea is to choose in higher proportion heuristics that act as local search and when no further improvement can be achieved from this heuristics, to choose more complex operators in order to scape from the local optima. Algorithm 3 details our implementation of this algorithm. Lines 5–9 from Algorithm 3 detail the behavior of our VND-based heuristic chooser.

**Algorithm 3** Variable Neighborhood Search (VND)

```
1: ls = IncumbentSolution()
2: $H_k$ = getHierarchicallyOrderedOperators()
3: i=0
4: while !LocalStopCriteria() do
5:     ls* = apply($H_i$,ls)
6:     if f(ls*) better than f(ls) then
7:         ls = ls*
8:         i=0
8:     else
9:         i= i+1
10:  end if
11:  end while
12:  return ls
```

### *2.6 Low Level Heuristic Pool*

An important design decision in any hyper-heuristic algorithm is the selection of a pool of neighborhood structures or heuristics. We used as a base the successful operators proposed in [23] extending some of them by multiple applications in order to have larger-scale neighborhoods. The base operators are briefly described below.

- **Simple Random Perturbation** (SRP): uniformly at random chooses a variable and changes its value for another one inside its feasible domain.
- **Swap** (SWP): selects two variables uniformly at random and interchanges their values if possible. Otherwise leaves the solution unchanged.
- **Statistical Dynamic Perturbation** (SDP): chooses a variable following a probability based on the frequency of variable selection in the last k iterations. Variables with lower frequency will have a higher probability of being selected.
- **Double Dynamic Perturbation** (DDP): similar to heuristic SDP, it selects a new variable with a probability inversely proportional to its frequency of selection in the last k iterations. It differs from SDP in that it internally maintains an additional solution (which is a copy of the first initialized solution) and makes random changes to it following the same distribution.
- **Best Single Perturbation** (BSP): chooses a variable following a sequential order and changes its value to that producing the minimum conflict.

Four additional neighborhoods were created by applying SRP two times, SWP three times, SDP five times, and DDP five times. Therefore, a total of nine neighborhood structures were considered in that order.

## 3 Solution Approach

### *3.1 Combining Methodology of Design with Hyper-heuristics*

As seen in Sect. 2, each high level chooser detailed so far utilizes a pool of low level heuristics, this pool is defined in Sect. 2.6. In this section, we define the codification and parameters used, as well as several details of each hyper-heuristic configuration.

We use the methodology of design approach shown in Sect. 2.2 in order to generalize the implementation of our metaheuristic over different CTTP instances. For each instance, the valid moves structure is built (Fig. 1). The detailed construction process for each of these lists is beyond the scope and purpose of this article, but interested readers are referred to [22, 24]. In the same manner as discussed above, these lists ensure by design that every variable in our solution represents a feasible selection in terms of time-space constraints. The main optimization exercise is to minimize the student conflict by means of the permutation of the events into time slots (i.e. integer values in our representation in Fig. 1). The function to be minimized has been described in Sect. 2.2, Eqs. (1) and (2). Initially our algorithm starts from a random solution and then we use Algorithm 1 to change the initial solution to a more promising one.

## 3.2 *High Level Iterated Variable Neighborhod Decsent*

In a hyper-heuristic framework with only ILS as the high level chooser the poll of low lever heuristics are presented without a particular ordering [27]. In our proposed approach, the improvement stage not only uses several neighborhoods, but it also incorporates a hierarchically ordering as seen in Sect. 2.6. We propose an hybridization of high level chooser, this hybridization takes the framework seen in Algorithm 1 but also utilizes a list of ordering heuristics as Algorithm 3. Algorithm 4 details our approach.

**Algorithm 4** Iterated Variable Neighborhood Decent (IVND)

```
1: S0 = GenerateIntialSolution()
2: S*= VND(S0)  {Improvement Stage}
3: while !StopCriteria() do
4:     S' = PerturbationStage(S*) {Using SRP Neighborhoods}
5:     S*' = VND(S') {Improvement Stage}
6:     if f(S*')  better than f(S*) then
7:         S* = S*'
8:     end if
9: end while
11: return S*
```

The approach proposed here (IVND, Algorithm 4), takes as a base the ILS hyper-heuristic framework, but substitutes the adaptive improvement stage by a variable neighborhood descent (VND) algorithm. if an improvement to $S$ is not possible using the first neighborhood $H_1$ then it is changed to $H_2$ *and* so forth with subsequent neighborhoods. As soon as an improvement is found with the current neighborhood, the sequence is restarted and $H_1$ is used again. This is a desired behavior since less perturbative heuristics are positioned at the beginning of the operator ordering.

## 4 Experiments and Results

In this section, several experiments are performed in order to find evidence about the performance of our approach against the CTTP state of art. We describe each experiment together with the characteristics of the benchmarks adopted.

### 4.1 Test Instances

The methodology of design allows the solution of diverse problem formulations it is merely necessary that each instance be expressed in terms of the generic structures (MMA, LPH and LPA). A well-known CTTP benchmark from the second international timetabling competition, PATAT ITC2007 Track 2 [1], is used for comparison between sequential and parallel approaches. This benchmark has 24 instances with main characteristics as follows:

**Patat 2007**

- A set of $n$ events that are to be scheduled into 45 timeslots.
- A set of $r$ rooms, each which has a specific seating capacity.
- A set *features* that are satisfied by rooms and required by events.
- A set of $S$ students who attend various different combination of events.

The hard constraints are:

- No student should be required to attend more that one event at the same time
- In each case the room should be big enough for all the attending students
- Only one event is put into each room in any time slot.
- Events should only be assigned to time slots that are pre-defined as available *
- Where specified, events should be scheduled to occur in the correct order. *

The soft constraints are:

- Students should not attend an event in the last time slot of a day.
- Students should not have to attend three or more events in successive timeslots.
- Student should not be required to attend only one event in particular day.

### 4.2 Experimental Design

Experimental conditions used throughout this section resemble those of the international timetabling competition (ITC) 2007 track 2 (post-enrollment course timetabling). A total of 24 instances are available [1]. The objective function minimizes the sum of hard and soft constraint violations. For the post enrolment track, the number of hard constraint violations is termed the '*distance to feasibility*' metric, and it is defined as the number of students that are affected by unplaced events. In general, the cost of a solution for each timetabling problem is denoted using a value, *sv*, where *sv* and *sv* are the sum of soft constraint violations, In this paper, we only record results with a value of 0 Hard violations. Following the *ITC2007* rules, 10 independent runs per instance were conducted, and results are reported as the average of *sv*. The stopping condition for each run corresponds to a *time limit* of about 10 min, according to the benchmark algorithm provided in the competition website. Finally, these instances range from 400 to 600 events. Experiments were conducted on a CPU with Intel i7, 8 GB Ram using the Java language and the 64 bits JVM.

### 4.2.1 Results

Table 1 shows the best results (out of 10 runs, as this was the experimental setting used in the competition). The last column correspond to the proposed IVND, the results from the competition winner (Cambazard) are taken from the *ITC-2007*

**Table 1** Results experiments ITC2007

| Instance | Cambazard | Ceschia | Lewis | Jat and Yang | ILSHH | itVND |
|---|---|---|---|---|---|---|
| 1 | 571 | 59 | 1166 | 501 | 650 | 677 |
| 2 | 993 | 0 | 1665 | 342 | 470 | 450 |
| 3 | 164 | 148 | 251 | 3770 | 290 | 288 |
| 4 | 310 | 25 | 424 | 234 | 600 | 570 |
| 5 | 5 | 0 | 47 | 0 | 35 | 30 |
| 6 | 0 | 0 | 412 | 0 | 20 | 10 |
| 7 | 6 | 0 | 6 | 0 | 30 | 15 |
| 8 | 0 | 0 | 65 | 0 | 0 | **0** |
| 9 | 1560 | 0 | 1819 | 989 | 630 | 620 |
| 10 | 2163 | 3 | 2091 | 499 | 2349 | 1764 |
| 11 | 178 | 142 | 288 | 246 | 350 | 250 |
| 12 | 146 | 267 | 474 | 172 | 480 | 450 |
| 13 | 0 | 1 | 298 | 0 | 46 | 30 |
| 14 | 1 | 0 | 127 | 0 | 80 | 68 |
| 15 | 0 | 0 | 108 | 0 | 0 | 30 |
| 16 | 2 | 0 | 138 | 0 | 0 | 20 |
| 17 | 0 | 0 | 0 | 0 | 0 | **0** |
| 18 | 0 | 0 | 25 | 0 | 20 | 10 |
| 19 | 1824 | 0 | 2146 | 84 | 360 | 299 |
| 20 | 445 | 543 | 625 | 297 | 150 | 150 |
| 21 | 0 | 5 | 308 | 0 | 0 | **0** |
| 22 | 29 | 5 | x | 1142 | 33 | 15 |
| 23 | 238 | 1292 | 3101 | 963 | 1007 | 892 |
| 24 | 21 | 0 | 841 | 274 | 0 | **0** |

website and the other columns from recent articles as indicated. The comparison was conducted using the 2007 competition rules and corresponding running time. Since all solutions are feasible, the values in the table correspond to the soft constraint violations, i.e. have zero hard constraint violation (except those marked with an *x*).

## 5 Conclusions and Future Work

This paper has presented a comparison between state-of-the-art hyper-heuristics and a newly proposed IVND algorithm when solving the course timetabling problem. Our approach substitutes the adaptive improvement stage by a VND algorithm with encouraging results over ITC 2007 track 2 instances.

Results indicate that the conceptually simpler IVND outperforms the recently proposed adaptive hyper-heuristic, and it shows competitive results against more complex state-of-the-art approaches.

Future work will study the extent to which the simpler and deterministic VND mechanism compares against more sophisticated reinforcement learning counterparts within and outside the competition setting. We will also explore whether applying multiple neighborhoods and adaptation to the perturbation stage in addition to (or instead of) the improvement stage provides improved results.

## References

1. URL http://www.cs.qub.ac.uk/itc2007/
2. Boizumault, P., Delon, Y., Peridy, L.: Logic programming for examination timetabling. Logic Program **26**, 217–233 (1996)
3. Burke, E., Eckersley, A., McCollum, B., Petrovic, S., Qu, R.: Hybrid variable neighborhood approaches to university exam timetabling. European Journal of Operational Research **206**(1), 46 – 53 (2010)
4. Burke, E.K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Ozcan, E., Qu, R.: Hyper-heuristics: a survey of the state of the art. Journal of the Operational Research Society (JORS) 64(12), 1695–1724 (2013)
5. Cambazard, H., Hebrard, E., OSullivan, B., Papadopoulos, A.: Local search and constraint programming for the post enrolment-based course timetabling problem. Annals of Operations Research **194**, 111–135 (2012)
6. Carter, M.: A survey of practical applications of examination timetabling algorithms. Operations Research **34**, 193–202 (1986)
7. Causmaecker, P.D., Demeester, P., Berghe, G.V.: A decomposed metaheuristic approachfor a real-world university timetabling problem. European Journal of Operational Research **195**(1), 307 – 318 (2009)
8. Colorni, A., Dorigo, M., Maniezzo, V.: Metaheuristics for high-school timetabling. Computational Optimization and Applications **9**, 277–298 (1997)

9. Conant-Pablos, S.E., Magaa-Lozano, D.J., Terashima-Marin, H.: Pipelining memetic algorithms, constraint satisfaction, and local search for course timetabling. MICAI Mexican international conference on artificial intelligence **1**, 408–419 (2009)
10. Cooper, T.B., Kingston, J.H.: The complexity of timetable construction problems. Ph.D. thesis, The University of Sydney (1995)
11. Hansen, P., Mladenovic, N.: Variable neighborhood search. In: Burke, E., Kendall, G. (eds.) Search Methodologies, pp. 211–238. Springer US (2005)
12. Jarboui, B., Damak, N., Siarry, P., Rebai, A.: A combinatorial particle swarm optimization for solving multi-mode resource-constrained project scheduling problems. Applied Mathematics and Computation **195**(1), 299 – 308 (2008)
13. Lourenço, H., Martin, O., Stützle, T.: Iterated local search. In: Glover, F., Kochenberger, G., Hillier, F.S. (eds.) Handbook of Metaheuristics, International Series in Operations Research & Management Science, vol. 57, pp. 320–353. Springer New York (2003)
14. Lajos, G.: Complete university modular timetabling using constraint logic programming. In E Burke and P Ross editors. Practice and Theory of Automated Timetabling (PATAT) I **1153**, 146–161 (1996)
15. Lewis, R.: Metaheuristics for university course timetabling. Ph.D. thesis, University of Nottingham. (August 2006)
16. L, Z., Hao, J.K.: Adaptive tabu search for course timetabling. European Journal of Operational Research **200**(1), 235 – 244 (2010)
17. Mayer, A., Nothegger, C., Chwatal, A., Raidl, G.: Solving the post enrolment course timetabling problem by ant colony optimization. International Timetabling Competition 2007 (2008)
18. Qu, R., Burke, E.K., McCollum, B.: Adaptive automated construction of hybrid heuristics for exam timetabling and graph coloring problems. European Journal of Operational Research **198**(2), 392 – 404 (2009)
19. Rudova, H., Muller, T., Murray, K.: Complex university course timetabling. Journal of Scheduling **14**, 187–207 (2011).
20. Sabar, N.R., Ayob, M., Kendall, G., Qu, R.: A honey-bee mating optimization algorithm for educational timetabling problems. European Journal of Operational Research **216**(3), 533 – 543 (2012)
21. Schaerf, A. & Gaspero, L.Burke, E. K. & Rudová, H. *(Eds.)*Practice and Theory of Automated Timetabling VI: 6th International Conference, PATAT 2006 Brno, Czech Republic, August 30–September 1, 2006 Revised Selected Papers Measurability and Reproducibility in University Timetabling Research: Discussion and Proposals *Springer Berlin Heidelberg,* **2007**, 40-49
22. Soria-Alcaraz, J.A., Terashima-Marin, H., Carpio, M.: Academic timetabling design using hyper-heuristics. Advances in Soft Computing, ITT Springer-Verlag **1**, 158–164 (2010)
23. Soria-Alcaraz, J.A., Ochoa, G., Swan, J., Carpio, M., Puga, H., Burke, E.K.: Effective learning hyper-heuristics for the course timetabling problem. European Journal of Operational Research 238(1), 77 – 86 (2014).
24. Soria-Alcaraz Jorge, A., Carpio, M., Puga, H., Sotelo-Figueroa, M.: Methodology of design: A novel generic approach applied to the course timetabling problem. In: P. Melin, O. Castillo (eds.) Soft Computing Applications in Optimization, Control, and Recognition, *Studies in Fuzziness and Soft Computing*, vol. 294, pp. 287–319. Springer Berlin Heidelberg (2013)
25. de Werra, D.: An introduction to timetabling. European Journal of Operational Research**19**(2), 151 – 162 (1985)

26. Willemen, R.J.: School timetable construction: Algorithms and complexity. Ph.D. thesis, Institute for Programming research and Algorithms (2002)
27. Ochoa, G., Walker, J., Hyde, M., Curtois, T.: Adaptive evolutionary algorithms and extensions to the hyflex hyper-heuristic framework. In: Parallel Problem Solving from Nature - PPSN 2012, Lecture Notes in Computer Science, vol. 7492, pp. 418–427. Springer, Berlin (2012).
28. Yu, E., Sung, K.S.: A genetic algorithm for a university weekly courses timetabling problem. Transactions in Operational Research **9**, 703–717 (2002).