Iterated Local Search Using an Add and Delete Hyper-heuristic for University Course Timetabling

Jorge A. Soria-Alcaraz[a,*], Ender Özcan[b], Jerry Swan[c], Graham Kendall[b,d], Martin Carpio[e]

[a] *Departamento de Estudios Organizacionales, División de Ciencias Economico Administrativas, Universidad de Guanajuato, México.*
[b] *University of Nottingham, School of Computer Science Jubilee Campus, Wollaton Road, Nottingham, NG8 1BB, UK*
[c] *York Centre for Complex Systems Analysis, University of York, UK*
[d] *University of Nottingham Malaysia Campus, Jalan Broga, 43500 Semenyih, Selangor Darul Ehsan, Malaysia*
[e] *Tecnológico Nacional de México, Instituto Tecnológico de León. México*

Abstract

Hyper-heuristics are (meta-)heuristics that operate at a higher level to choose or generate a set of low-level (meta-)heuristics in an attempt of solve difficult optimization problems. Iterated Local Search (ILS) is a well-known approach for discrete optimization, combining perturbation and hill-climbing within an iterative framework. In this study, we introduce an ILS approach, strengthened by a hyper-heuristic which generates heuristics based on a fixed number of add and delete operations. The performance of the proposed hyper-heuristic is tested across two different problem domains using real world benchmark of course timetabling instances from the second International Timetabling Competition Tracks 2 and 3. The results show that mixing add and delete operations within an ILS framework yields an effective hyper-heuristic approach.

Keywords: Hyper-heuristic, Iterated local search, Add-Delete List, Methodology of design, Educational timetabling

*corresponding author
Email addresses: jorge.soria@ugto.mx (Jorge A. Soria-Alcaraz), ender.ozcan@nottingham.ac.uk (Ender Özcan), jerry.swan@york.ac.uk (Jerry Swan), graham.kendall@nottingham.edu.my (Graham Kendall), jmcarpio61@hotmail.com (Martin Carpio)

1. Introduction

Hyper-heuristics are (meta-)heuristics that choose or generate a set of low level (meta-)heuristics in an attempt to solve difficult search and optimization problems [1, 2]. Heuristics can be used to search the solution space directly or construct a solution based on a sequence of moves. Hyper-heuristics aim to replace bespoke approaches by more general methodologies with the goal of reducing the expertise required to construct individual heuristics [3]. 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].

Educational timetabling problems are common and recurring real-world constraint optimization problems which are known to be NP-hard [5, 6, 7]. An educational timetabling problem requires scheduling of a set of events using limited resources subject to a set of constraints. There are a range of educational timetabling problems, such as examination timetabling and high school timetabling. This study focusses on the *university course timetabling problem*, which can be further categorized as either *post-enrollment problems,* in which the student enrollment is available before the timetabling process, and *curriculum-based problems* in which the curricula of the students are known, but not the student enrollment [8]. There are two main types of constraints in a timetabling problem: *hard* and $soft$ constraints. The hard constraints have to be satisfied in order to obtain a *feasible* solution, while violations of soft constraints are allowed, since they represent preferences. It is still the case at some universities that timetables are constructed by hand. Considering the inherent difficulty of generating high-quality feasible timetables which violate few soft constraints, it is usually desirable to automate timetable construction to improve upon solutions obtained by human experts [9]. However, automation of timetabling is not an easy task, since designing an automated method frequently requires a deep knowledge of the problem itself as well as the particular characteristics of the instance to be solved. This knowledge, in most cases, is not readily available to the typical researcher/end-user.

In this study, we describe an iterated local search (ILS) algorithm hybridized with a hyper-heuristic that generates heuristics based on add-delete operations to solve examination and university course timetabling problems. Re-usability, modularity and flexibility are some of the key features of the pro-

posed approach. To evaluate the generality of the generation hyper-heuristic, it is tested on a range of problem instances across two different domains; namely, post-enrollment university course timetabling and curriculum-based university course timetabling, without modification of the underlying solution framework.

Although the problem domains we investigate are timetabling problems, each domain exhibits differing characteristics, particularly with respect to the complexity of the real-world constraints. This is the main reason why a recent competition has used two tracks. The International Timetabling Competition series was organized to create a common ground for the cross-fertilization of ideas, bridging the gap between theory and practice and creating a better understanding between researchers and practitioners in this field [8]. The second competition in the series (ITC2007) was on educational timetabling, containing an examination timetabling track and two separate tracks for post-enrollment and curriculum-based university course timetabling [8]. We have investigated the performance of the proposed approach on the last instances. The results show that our approach is promising.

This paper is organized as follows. Section 2 provides an overview of educational timetabling problems, particularly university course timetabling. This section also discusses solution methodologies. Section 3 discusses the specifics of the solution methodology including the relevant data structures and the add-delete representation. Section 4 summarizes the experimental results. Finally, Section 5 presents the conclusions and future work.

2. Background

2.1. Hyper-heuristics

The term "hyper-heuristic" is relatively new, having first appeared in a technical report by Denzinger et al. [62] as a strategy to combine artificial intelligence methods. The un-hyphenated version of the term initially appeared in Cowling et al. [3] describing hyper-heuristics as *heuristics to choose heuristics* in the context of combinatorial optimization. However, the idea of automating the design of heuristic methods is not new and can be traced back to the 1960's in works such as Fisher et al. [11] and Crowston et al. [12].

The main motivation behind hyper-heuristic research is to reduce the need for a human experts in designing effective algorithms, and consequently to raise the level of generality at which search methodologies are able to operate.

Hyper-heuristics share the quest for greater autonomy and generality with appraoches such as autonomous search by Hamadi et al. [13], reactive search by Battiti [14], adaptive operator selection by Maturana et al. [15], adaptive memetic algorithms [16], automated tuning [17] and parameter control by Lobo et al. [18]. In a recent book chapter by Burke et al. [4], the authors extended the definition of hyper-heuristics and provided a unified classification which captures more recent work that is being undertaken in this field. A hyper-heuristic is defined as a "*search method or learning mechanism for selecting or generating heuristics to solve computational search problems*". The classification of approaches considers two dimensions: (i) the nature of the heuristics' search space, and (ii) the different sources of feedback information from the search space. According to the nature of search space, we have;

- *Heuristic selection*: methodologies for choosing or selecting existing heuristics.
- *Heuristic generation*: methodologies for generating new heuristics from given components.

Orthogonal to the notion of selective versus generative is the distinction between constructive and perturbative mechanisms for searching the solution space, i.e. whether it operates via partial or complete solutions respectively.

This study describes an ILS which uses a generative hyper-heuristic for creating perturbation heuristics (move operators). The important feature of the proposed approach is the use of an add-delete list (i.e. a sequence of insertions or deletions of partial solution states) which acts like a ruin-recreate operator as proposed by Swan et al. [19]. This idea of removing and reinserting parts of the solution has produced encouraging results in previous work, for example: Schrimpf et al. for Vehicle Routing [20] and Misevicius et al. [21] [22] for Quadratic Assignment. It is important to note that not all add-delete lists are feasible. In Section 3.2.1, we describe a divide-and-conquer algorithm for building a feasible add-delete list.

Figure 1 illustrates the traditional framework for selective hyper-heuristics, with the *domain barrier* insulating the high-level search strategy from the underlying problem domain. The high-level strategy selects and applies a low-level heuristic (move operator) from the available set considering only (the history of) domain-independent information from the search process. It is worth mentioning, however, that low-level heuristics which encapsulate

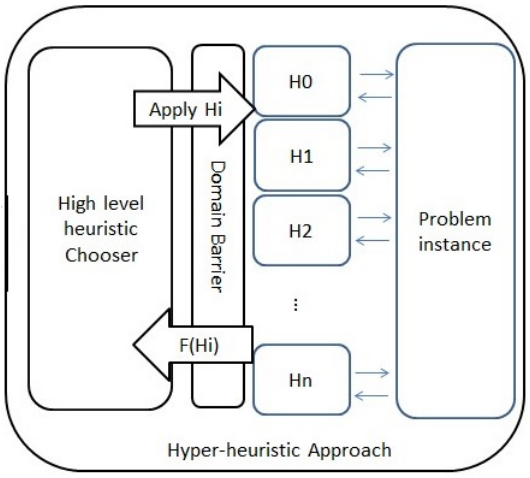


Figure 1: General framework of a selection hyper-heuristic based on Cowling et al. [3].

domain-specific information can be (and usually are) incorporated in the pool of available heuristics.

When a hyper-heuristic uses some feedback from the search process, it can be considered as a learning algorithm (Figure 1). According to the source of the feedback during learning, Burke et al. [4] distinguishes between *online* and *offline* learning hyper-heuristics, i.e. online learning takes place whilst a given algorithm is solving a problem instance.

In offline learning, the idea is to gather knowledge (e.g. in the form of rules or programs), from a set of training instances, in expectation of generalizing to unseen instances. Genetic Programming is one of the most commonly used methods for heuristic generation. Examples of off-line heuristic generation include [23], [24] with [25] introducing a policy-matrix representation to inform the generation of heuristics. The add-delete hyper-heuristic proposed in this study is a novel online heuristic-generation method.

2.2. Educational Timetabling

Although it has been extensively studied, educational timetabling problems are still of interest to many researchers and practitioners. There are many types of educational timetabling problems and this section focuses on a specific type of educational timetabling problem, that is, university course timetabling, in which the main objective is to assign each subject a timeslot such that that they attend all lectures to which they are enrolled. Formally,

Table 1: A summary of previous studies on education timetabling problems are provided in chronological order (Exam: examination timetabling, Post: post-enrollment course timetabling, Curr: curricula-based course timetabling, Univ: university timetabling, HSchool: high school timetabling .

Year	Problem	Approach	Source
1985	Exam and Curr	Graph heuristics	Werra [26]
1986	Exam	Linear programing	Carter [27]
1995	Curr	Complexity analysis	Cooper et al. [6]
1996	Univ	Logic programing	Lajos [28]
1996	Exam	Logic programing	Boizumault et al. [29]
1997	HSchool	Genetic algorithms	Colorni et al. [31]
1998	Exam	Simulated annealing	Thompson et al. [36]
2002	HSchool	Complexity analysis	Willemen [7]
2002	Univ	Genetic algorithms	Yu et al. [32]
2002	Curr	Ant Colony System	Socha et al. [34]
2007	Exam and Post	Hyper-heuristics	Burke et al. [39]
2007	Post	Ant Colony System	Mayer et al. [33]
2009	Exam	Hyper-heuristics	Qu et al. [41]
2010	Post	Hyper-heuristics	Soria-Alcaraz et al. [40]
2010	Post	Adaptive Tabu Search	Z. Lü et al. [30]
2010	Exam	Variable Neighborhood Search	E.K. Burke et al. [35]
2011	Curr and Post	Constraint Programing	H. Rudová et al. [37]
2012	Post	Consraint Programing	H. Cambazard et al. [38]

the university course timetabling problem can be considered as a Constraint Satisfaction Problem (CSP) where the variables are events and the most common constraints are time-related. A more detailed explanation of each timetabling variant used in this paper can be found in Section 2.2.1. This problem is reported as extremely challenging by Cooper et al. [6] and Willmen et al. [7].

Many approaches have been proposed for solving variants of educational timetabling problems, ranging from early approaches based on graph heuristics [26], linear programming [27] and logic programming [28, 29] to metaheuristics including tabu search [30], genetic algorithms [31, 32], ant colony optimization [33, 34], variable neighborhood search [35], simulated annealing [36], among others. Various CSP solvers have also been proposed to solve timetabling problems [37, 38]. In recent years, hyper-heuristics have been applied to timetabling with encouraging results [39, 40, 41]. A chronological order of the state of the art in Educational Timetabling can be seen in Table 1

Most of the studies in the last decade use benchmarks created for the International Timetabling Competition ITC2007 (University Course timetabling appearing in Tracks 2 and 3). The competition entries and datasets serve as a benchmark for performance comparison for any newly-proposed solution methodologies. We have therefore tested the performance of our approach on the real-world instances from this competition, as described in the following Section.

2.2.1. The Second International Timetabling Competition (ITC2007)

Three international timetabling competitions have been organized to date. The most recent competition (ITC2011) was on high school timetabling, but the first (ITC2002) and second (ITC2007) competitions were concerned with university timetabling. In this study, we apply our approach to the post-enrolment and curriculum-based course timetabling problem instances provided in the ITC2007 competition (Tracks 2 and 3, respectively). A solution to an ITC2007 instance is evaluated as the sum of hard and soft constraint violations. For the post enrollment track, there are no hard constraints. Consequently, in this particular case the number of hard constraints 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 pair of values, (hv, sv), where hv and sv are the sum of hard and soft constraint violations, respectively. In order to compare two or more solutions, the pairs (hv, sv) are ranked in a lexicographically ascending order. More details on ITC2007 can be found in [8].

The main characteristics of the post-enrollment course timetabling problem instances (Track 2) are as follows:

- A set of n events that are scheduled into 45 timeslots.
- A set of r rooms, each which has a specific seating capacity.
- A set of *room-features* that are satisfied by rooms and required by events.
- A set of s students who attend various different combinations 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 satisfy the class requirements (be big enough for all the attending students and/or required class features).

- Only one event is put into each room in any timeslot.
- Events should only be assigned to timeslots 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 be scheduled to attend an event in the last timeslot 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 a particular day.

The main characteristics of the curriculum-based course timetabling problem (Track 3) instances are as follows:

- A set of n teaching days (typically 5 or 6). Each day is split into a fixed number of t timeslots which are the same for all days. A period p is a pair (n_i, t_j) composed by a day and a timeslot.
- A set of l lectures to be scheduled in distinct periods, a lecture is attended by a given *number of students*.
- A set of r rooms with fixed capacity.
- A *curriculum*, i.e. a group of courses such that any pair of courses in the group have students in common.

The hard constraints are:

- All lectures must be scheduled, and they must be assigned to distinct periods.
- Two lectures cannot take place in the same room in the same period.
- Lectures of courses from the same curriculum must be all scheduled in different periods.
- Lectures must meet the requirements and availability of teachers.

The soft constraints are:

- For each lecture, the number of students that attend the course must not exceed the room's capacity.
- Lectures of each course must be spread into the given *minimum number of days*.
- Lectures in a curriculum should be adjacent to each other (consecutive periods).

- All lectures of a course should be given in the same room.

Some important studies in the scientific literature on solving ITC2007 Track 2 and 3 instances are as follows.

Track 2 - Post-Enrollment Course Timetabling

The winning algorithm of the 2007 competition Track 2 was presented in [38]. This approach is a multi-stage local search algorithm considering several neighborhoods, and involving aspects of tabu search and simulated annealing at different stages.

The second ranking algorithm in ITC2007 Track 2 was [42], which formulates the timetabling instances as constraint satisfaction problems, and then uses a general purpose CSP solver to find solutions. In particular, they used the solver proposed in [43], which uses a hybrid metaheuristic combining tabu search and iterated local search, and handles weighted constraints. This algorithm was designed specifically for the competition.

In the later work of Cheschia et al. [44] a single-step metaheuristic approach based on simulated annealing is proposed, with a neighborhood composed of moves that reschedule one event or swap two events. The solver is able to deal with all the variants of the course timetabling problem (CTTP) proposed in the literature, and provides new best-known solutions for many instances.

In the work of Lewis et al. [45] a 3-stage local search algorithm is presented, in which a constructive phase is followed by two separate simulated annealing phases. The algorithm's behavior depends on the allotted running time, as several parameters controlling the intensity of search are derived from this. The algorithm provides good results on ITC2007 Track 1 instances, but it is not superior to the top entries and fails to produce new best-known solutions. The algorithm is also competition-specific.

In Jat et al. [46] a two phase approach is used. In the first phase, a guided genetic algorithm is applied which integrates local search. The guided search strategy uses a data structure that stores features of previous good individuals to guide the generation of offspring. A local search is then used to improve the quality of the individuals. In the second phase, a tabu search heuristic is used to further improve the solution, if possible. This approach presents encouraging results and improves upon the best-known solutions for ITC track 2 benchmark.

Track 3 - Curricula-Based Course Timetabling

The winning algorithm of this track was presented in Müller et al. [47]. Their

approach is a two-stage generic algorithm that uses a third party constraint solver to create a good initial solution and improve it with an iterative forward search process. In Lu et al. [48] a three-stage *Adaptive Tabu Search* is presented. The first stage builds an initial feasible timetable through a fast heuristic, then intensification and diversification phases are alternatively applied using tabu search to reduce soft constraints violations. In Hao et al. [49] a partition-based approach is used to compute new lower bounds for theses instances. This divide and conquer approach uses iterative tabu search to partition the initial problem into sub-problems which are solved with an ILP solver. Computational results show that this approach is able to improve on the current best lower bounds for 12 out of the 21 benchmark instances, and to prove optimality for six of them. These new lower bounds are useful to estimate the quality of the upper bounds obtained with various heuristic approaches. Asín Achá et al. [50] give an application of several satisfiability solvers, where (by using different encodings) they were able to compute and obtain new best solutions. Recently, Cacchiani [51] computed new best solutions using an approach similar to [49]. However in this case the partition is based on soft constraints instead of hard constraints.

3. Solution Approach

Iterated local search (ILS) is a relatively simple methodology that has been successful in a variety of domains. It operates by iteratively alternating between applying a move operator to an incumbent solution and performing local search on the perturbed solution. This search principle has been rediscovered multiple times within different research communities and given different names [14]. The term *iterated local search* was proposed in Lourenço et al. [52]. In this study, we describe an ILS approach strengthened by a novel add-delete hyper-heuristic and investigate the performance of our approach across a variety of university course timetabling problem instances from ITC2007. The proposed approach is outlined in Algorithm 1, in which f is the fitness function measuring the cost for a given solution. An initially-constructed solution (s_0) goes through perturbation ($SimpleRandomPerturbation$) and local search ($ImprovementStage$) stages sequentially until the termination criteria are satisfied. Whenever a new solution is produced, we ensure that time and room capacity constraints are still respected using the data structures described in section 3.1. As required by the ITC2007 rules, execution terminates as soon as a given time limit is reached. At each step, only

non-worsening solutions (timetables) are accepted. The cost function for evaluating a given solution returns a pair of values as (hv, sv) as described in Section 2.2.1. The value of hv has priority over sv, hence a solution with a lower value of hv is always considered to be an improving solution regardless of the sv value, while a solution with a lower value of sv and higher value of hv is considered to be a worsening solution.

Algorithm 1 Iterated Local Search

```
1: s_0 = GenerateInitialSolution
2: s* = ImprovementStage(s_0) {Add-delete hyper-heuristic}
3: while !TerminationCriteria() do
4:     s* = SimpleRandomPerturbation(s*)
5:     s' = HillClimbing(s*)
6:     s*' = ImprovementStage(s') {Add-delete hyper-heuristic}
7:     if f(s*') ≤ f(s*) then
8:         s* = s*'
9:     end if
10: end while
11: return s*
```

Algorithm 1 presents the High-level ILS algorithm which incorporates an Add-Delete representation. Essentially this algorithm moves in the space of partial solutions via a "ruin and recreate" strategy that deletes and subsequently reschedules events. In line 5 hill climbing is used to improve the current solution. In this phase the algorithm searches for the variable/event with the highest number of conflicts and then iteratively applies a single heuristic until no improvement is possible. The heuristic is chosen uniformly at random. This procedure is applied sequentially for 2% of the variables having the highest number of conflicts. Preliminary experiments showed that this is a practical and fast way to reach local optima. The search phase in our ILS is performed by a novel hyper-heuristic as indicated by lines 2 and 6 of Algorithm 1. The hyper-heuristic is used to generate a sequence of add-delete operations. The *delete* operation removes (un-assigns) a variable (event) from the schedule, while the *add* operation reassigns an unscheduled event to a time-slot based on a set of constraint satisfaction low level heuristics. The add-delete list (ADL) is then used to modify a given solution. The same ADL could result in a different new solution given a different input. A complete timetable is processed based on this list and a new timetable is built. The details of the local search algorithm is described in Section 3.2.

3.1. Methodology of Design for University Course Timetabling

In order to deal with a given university course timetabling problem in a generic fashion (including instance-specific constraints) as provided in Section 2.2, the *methodology of design* developed by Soria-Alcaraz et al. [53, 54] is utilised. The main idea of this methodology is to add an extra layer of generality, the principal objective of which is to discard *by design* the largest number of instance-specific constraints, in order to build a search space where the heuristic strategy deals with a minimum number of constraints. Figure 2 shows graphically the main idea behind the Methodology of Design concept. The main effort of any heuristic after the application of this layer of generality is to search inside this feasible space in order to find an optimal solution, where both hard and soft constraints are satisfied.

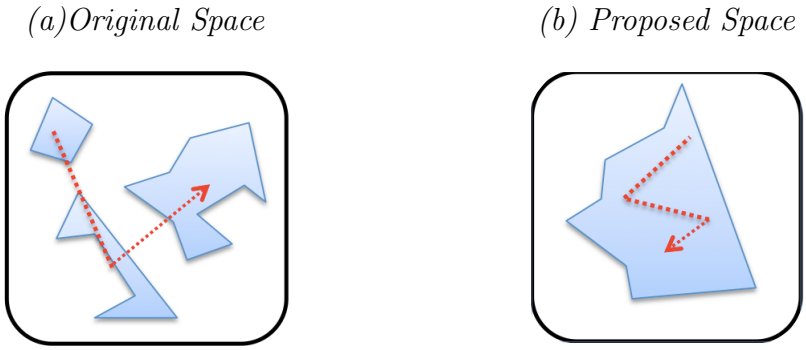


Figure 2: (*a*)In the original context-dependent space feasible regions are spread over the search space, a solver needs to manage infeasible solutions in order to travel between feasible regions.(*b*) In the Methodology of Design search space, there is only a single feasible region, at any given step a search algorithm can at least reach a real-world solution.

In order to work over a feasible space such as the one shown in Figure 2 (b) Soria et al. [54] proposes several generic structures, namely *MMA matrix, LPH list* and *LPA list* to resolve *by design* the following hard constraints:

- When one or more events are not assigned into the timetable.
- When one or more events do not have a room/laboratory assigned to them.

The satisfaction of these hard constraints are guaranteed by the selection of values from two main generic structures LPH and LPA defined as follows:

Figure 3: Example of offered timeslots

Time	*Monday* and *Wednesday*	*Tuesday* and *Thursday*	*Friday*
7 : 00 am to 8 : 50	$t1$	$t2$	$t9$
8 : 50 am to 10 : 30	$t3$	$t4$	$t10$
10 : 30 am to 12 : 15	$t5$	$t6$	$t11$
12 : 15 pm to 2 : 00	$t7$	$t8$	$t12$

Figure 4: LPH list

Events	*Timeslots*
$e_0 = Chemistry$	$(t1)$ or $(t3)$ or $(t5)$
$\vdots$	$\vdots$
e_n	$(t7)$ or $(t9)$

LPH list: This structure contains the timeslots for each event to be timetabled.

LPA list: This structure contains the list of classrooms with suitable capacities allowed for each event.

A simple example showing why these lists satisfy the previous requirements follows. Consider an actual instance-based situation where an event named Q_1 needs to be scheduled two days per week. The timeslots offered by the institution are shown in Figure 3 and the LPH and LPA lists for this toy example can be seen in Figures 4 and 5. The teacher for this subject works at school only Monday and Wednesday from 7 am to 12:15pm. This Event also needs to be assigned into a laboratory with some specific equipment. The only rooms capable of satisfying demands for this event are: $\{L_1, L_2\}$.

With this information, a Cartesian product $LPH \times LPA$ with feasible time-space values for event Q_1 can be easily constructed, this product is: $\{(t1, L_1), (t1, L_2), (t3, L_1), (t3, L_2), (t5, L_1),\ (t5, L_2)\}$, and no matter which

Figure 5: LPA list

event	*Classrooms*
$e_0 = Chemistry$	$(L1)$or $(L2)$
$\vdots$	$\vdots$
e_n	(A)

pair ($(timeslot, room)$) we select from this list, this selection ensures (*by design*) the feasibility for event Q_1. The idea is to construct similar structures for each event in order to have a feasibility region for each possible subject-timeslot assignment. A detailed analysis of the construction of these structures in more complex scenarios can be found in Soria-Alcaraz et al. [53, 54].

For the soft constraints, our ADL hyper-heuristic searches in the feasible space (by means of the $LPH \times LPA$ list) for the elements that satisfy the largest number of time-related criteria. One specific type of common soft constraint is represented by the third generic structure: the MMA matrix. This matrix is used to calculate how many students are enrolled in two subjects/events and therefore these lectures must not be assigned to the same timeslot.

MMA matrix: This symmetric matrix contains the number of students in conflict for a given pair of events, i.e. the number of students who are enrolled in a given pair of courses (these courses should not be assigned in the same timeslot for any student). For example, the MMA matrix shown in Figure 6 indicates that there exists three students who should not be taking e_2 and e_3 at the same time. The MMA matrix is the principal structure required for the computation of equation (2).

All such soft constraint violations represented in terms of the basic structures discussed below can be then minimized using the following equations:

$$minimize(\sum_{\forall i} FA_{V_i}) \tag{1}$$

$$FA_{V_i} = \sum_{s=1}^{|V_i|} \sum_{l=s+1}^{|V_i|} |S_i(s) \cap S_i(l)| \tag{2}$$

where FA_{V_i} is the overall number of pairwise conflicts in a given timetable (solution) at the the i^{th} timeslot, V_i is the *vector* of events scheduled for the i^{th} timeslot which should not have been scheduled in the same timeslot, $S_i(s) \cap S_i(l)$ is the joint set of students that simultaneously attend the s^{th} and l^{th} events in the vector V_i.

By equation (2) a sum of student conflicts per timeslot is calculated and the total conflict is obtained by summation of all student conflicts in the timeslots. This is the main fitness function of the Methodology of Design. However, in real-world problems it is usual for further constraints to

be present in the fitness function, as is the case with ITC 2007 Tracks 2 and 3. In such cases, it is easy to add the new constraints into Equation (2) (where conflicts per timeslot are calculated) or Equation (1) (where total conflicts per timetabling are obtained). For example, global constraints such as *Timeslots used* can be added into Equation (1) simply by counting how many timeslots are evaluated and adding a penalization if the sum of timeslots used exceeds a certain rule. In this paper some additions are integrated into the fitness function in order to have an adequate metric for timetabling evaluation according to the rules of ITC 2007 Tracks 2 and 3 instances. These adaptations are discussed in section 3.1.1.

Finally, it is important to say that with this kind of generic representation the cost of maintaining a feasible solution across the search process is practically zero as long as the current solution meets the following requirements:

- Every event/subject is assigned into a pairwise timeslot-room.
- This Timeslot-Room pair is taken from the $LPH \times LPA$ list for each event.

The main effort is then to search for the best combination of these feasible values in order to reach a timetable with minimum conflicts.

3.1.1. Measuring Solution Quality Under Methodology of Design

Once we have our representation (based on $LPA \times LPH$ list), we can easily measure the number of conflicts that a specific solution has. As a toy example, consider a school that offers 5 subjects $e_1, \ldots, e_5$, with only two timeslots available t_1, t_2 and three classrooms l_1, l_2, l_3. Figure 6 shows the MMA matrix used in this example. The $LPA \times LPH$ list constructed for this example can be seen on Figure 7. This shows several columns, the first one contains the set of events/variables to be assigned and the second column contains the current solution generated by our approach. This solution is an integer array that represents a specific pair *(classroom,timeslot)* for each variable. This pair is taken from the Cartesian product $LPA \times LPH$ shown in bold font in the third column.

Suppose that a heuristic chooses as the current solution the next assignment: e_1 in (t_2, l_2), e_2 in (t_1, l_3), e_3 in (t_2, l_1), e_4 in $(t_1, l_1)$and e_5 in (t_2, l_3). This solution represents a feasible timetable, but it is necessary to measure how many conflicts it has. To do this we use Equations: (1) and (2) as follows:

Figure 6: MMA matrix example

	e_1	e_2	e_3	e_4	e_5
e_1	–	2	4	0	1
e_2	2	–	3	0	2
e_3	4	3	–	1	2
e_4	0	0	1	–	5
e_5	1	2	2	5	–

Figure 7: LPA x LPH example

$Variable/event$	$Selected\ value$	$Timetabling\ (LPAxLPH)$
e_1	2	$(t_1, l_1), (t_1, l_3), (\mathbf{t_2}, \mathbf{l_2})$
e_2	1	$(t_1, l_2), (\mathbf{t_1}, \mathbf{l_3}), (t_2, l_1)$
e_3	0	$(\mathbf{t_2}, \mathbf{l_1})$
e_4	0	$(\mathbf{t_1}, \mathbf{l_1})$
e_5	1	$(t_1, l_1), (\mathbf{t_2}, \mathbf{l_3})$

- We group each event by its current timeslot. In this toy example we have two timeslots: t_1 which is associated with e_2, e_4, and t_2 which is associate with (e_1, e_3, e_5)
- We calculate the conflict held by each vector/timeslot using the MMA matrix: In the case of t_1 we simply obtain the value $MMA_{e_2,e_4} = 0$. In the case of t_2 we get the conflict with Equation (2), The conflict in $t_2 = MMA_{e_1,e_3} + MMA_{e_1,e_5} + MMA_{e_3,e_5} = 4 + 1 + 2 = 7$
- We calculate the general conflict using Equation (1). In our case this is $7 + 0 = 7$.

The objective of any heuristic is to minimize the general conflict. This is the basic way to calculate conflicts under the methodology of design. Other characteristics desired by each institution can also be measured as well, for example: conflicts in rooms/laboratories, specific events that must span a minimum number of days or curriculum events that should be adjacent.

3.2. Improvement Using Add-Delete Lists

3.2.1. Forming a Feasible List of Add-Delete Operations

A solution to a timetabling problem can be reconstructed from a previous solution by successively deleting and adding (re-scheduling) events. In our hyper-heuristic framework, we employ a sequence of add-delete operations with a fixed length. This representation will be referred to as ADL from this point onward. A signed integer representation is used, allowing us to differentiate between the events that need to be *temporarily* unscheduled and the others to be reinserted into the timetable. A feasible ADL must adhere to certain conditions:

- The length of an ADL is always even. An ADL always contains two operations, namely add (+ value) and delete (- value) for each distinct event, e_i in the list.
- The first appearance of an event must be a *temporary delete* operation (-).
- Performing an add operation on an event indicates the assignment of an event e_i and can appear at any time after its deletion.
- Before and after the application of ADL the current solution must represent a complete and feasible timetable (each event is assigned a timeslot).

A simpler implementation of the ADL approach is given in Swan et al. [19] where a binary-based ADL is proposed for the examination timetabling problem. In this approach add-delete operations can be represented by a fixed-length binary string. In this binary context a 0 value represents the deletion of an event and a 1 value represents the re-scheduling of the previously deleted event to the partially constructed timetable. This binary string is used to identify in which order a event/variable will be deleted and reinserted into the timetable. Add and delete operations can be handled in many ways. The simplest approach for a *delete* operation is to choose an event randomly and put it into an 'unscheduled events' list. In contrast, an *add* operation requires two consecutive actions to be taken: firstly, an event should be selected from the list of unscheduled events and then a suitable period should be selected for scheduling. The ADL is used by our hyper-heuristic to construct a feasible timetable from a previous valid solution. This is achieved by the successive application of the operations (delete and add) coded in a given ADL. A feasible binary string of length $2n$ can be formed using Algorithm 2. For simplicity of explanation, let's assume we have global variables $String\ addDeleteList[MAX\text{-}SIZE]$ and $int\ list\_length = 0$:

Algorithm 2 *int GenerateAddDeleteList(int n, String bitString)*

```
1: if n = 1 then
2:    addDeleteList[list_length++] = bitString;
3:    return list_length;
4: else
5:    GenerateAddDeleteList(n − 1, bitString+'01');
6:    GenerateAddDeleteList(n − 1,'0'+bitString+'1');
7:    if bits doesn't have prefix '01' then
8:       GenerateAddDeleteList(n − 1,'01'+bitString);
9:    end if
10: end if
11: return list_length
    '+' denotes concatenation. Initialize with: GenerateAddDeleteList(n,"01")
```

Algorithm 2 outputs the list of all unique binary/add-delete strings containing n add and delete operations in the *addDeleteList* array and returns the size of the list in *list_length*. This algorithm shows a divide-and-conquer approach to generate add-delete strings for n events. For $n = 1$, the only such string is "01", for $n = 2$, we have {"0101","0011"} for $n = 3$, {"010101", "001011", "001101", "000111","010011"}. The parameter n is clearly important, because it determines the degree of impact that an ADL can have on the current solution. As might be expected, preliminary experiments suggest that small values of n produce a small perturbation in the current solution, i.e. a change to the time allocations of one or two events, while a larger n means that more events are re-assigned into different timeslots. A more detailed study of the impact of parameter n is given in Section 4. For a binary ADL list with $n = 2$ (e.g. 0011), we consequently have two initial delete operations (i.e. the elimination of two arbitrary events) followed by two re-schedulings of the previously deleted events [19].

In this paper, we enrich this binary scheme by proposing an integer-based ADL where, instead of only having values for deleting (0) and adding(1) we have an *id* value for each specific event with two possible signs: (-) where the specific event is chosen for temporary removal and (+) where the specific event is to be re-scheduled. So, given a solution S and $n = 2$, then an ADL of "$-4, -2, 2, 4$" derived from "0011" indicates that the events e_4 and e_2 are temporarily deleted from S, and a new solution S' is formed by rescheduling the events e_2 and then e_4. The actual length of the ADL (n) is a parameter fixed by the user. This enriched representation can encode our original binary ADL (0011) into the following sequences of actions

$(-4,-2,2,4)$,$(-4,-2,4,2)$,$(-2,-4,2,4)$ and $(-2,-4,4,2)$. Each of these integer-based ADLs describe two removal operations followed by two re-scheduling operations. However, this representation now describes specific **order** and **events** that we need to work with in the current perturbation. With this new information, we could explore more sophisticated methods for ruin-recreate heuristics. For example, we could keep a record of events that usually appear in the ADL history, we can promote (or inhibit) specific events to appear in future ADL's.

3.2.2. Rescheduling Events based on Add-Delete Lists

The negative values (-) in an ADL represent events that are *temporarily* removed from the current timetable. This causes a temporary relaxation in the restrictions to other events. A positive value (+) represents the reassignment of the selected event to a new timeslot. In our approach, an event is chosen for rescheduling in the order in which appear in an add-delete list. This rescheduling is performed by constraint satisfaction (CS) heuristics. The heuristic is randomly-selected for each entry in the list and is taken from the following set of low level heuristics:

- *Min-Conflicts* assigns an event to a valid timeslot generating the least number of conflicts. Pseudo code of this heuristic can be seen in Algorithm 3.

- *First-fit* assigns an event to a valid timeslot which contains the least number of events attached to it.

- *Worse-fit* assigns an event to a valid timeslot which contains the largest number of events attached to it.

- *Modified Brelaz Heuristic* was originally developed for graph colouring [55]. This heuristic chooses the timeslot with the smallest saturation degree (timeslot with the least number of events is selected), but in the case of a tie, it chooses from those the timeslots with the largest future degree. that is the one which has the largest number of unassigned events. This heuristic assigns an event to this specific timeslot. Pseudo code for this heuristic can be seen in Algorithm 4.

Algorithm 3 Min-Conflicts Heuristic for Timetabling

Require: **var** t: **Timetable**, **var** e: **Event**
1: $Timeslots = t.getNumberOfTimeSlots()$
2: **var** tList : List of Timetables
3: **for** $i \leftarrow 0$ **to** $Timeslots$ **do**
4: $t' = copy(t)$
5: $t' = AssignEventInTimeslot(e, i, t')$
6: $Evaluate(t')$
7: $tList.add(t')$
8: **end for**
9: $t* = tList.ReturnBestTimetable()$
10: **return** $t*$

Algorithm 4 Modified Brelaz Heuristic for Timetabling

Require: **var** t: **Timetable**, **var** e: **Event**
1: $TimeslotCount = t.getNumberOfTimeSlots()$
2: **var** tsList : List of Timeslots
3: **for** $i \leftarrow 0$ **to** $TimeslotCount$ **do**
4: $timeslot = t.getTimeslot(i)$
5: $EvaluateAvialiableSpace(timeslot)$
6: $tsList.add(timeslot)$
7: **end for**
8: $tsList.OrderbyMaxSpace()$
9: **if** $ExistsTie(tsList)$ **then**
10: **var** tset : List of Timeslots= $tsList.getBestTieSet()$
11: $tsList.InsertAtPosition(tset.getTimeslotwithMaxPossibleEvents(), 0)$
12: **end if**
13: $AssignEventInTimeslot(e, tsList.get(0), t)$
14: **return** t

3.2.3. Add-Delete Hyper-heuristic

The add-delete hyper-heuristic, represented as a *ImprovementStage* (Algorithm 5), attempts to iteratively improve a given solution by constructing an effective ADL and applying it to the current solution. If the move generates a non-worsening solution after the application of the ADL (this means $\Delta > 0$) at time t, then this ADL influences the generation of a new ADL in time $t + 1$ (line 3, Algorithm 5) so it is illustrated in Algorithm 6. *ImprovementStage* maintains a history of recently used ADLs as a queue of size w, denoted as q. This list is initially built randomly and is updated at every step (line 9, Algorithm 5). An entry in q is a pair of values (ADL_t, Δ_t)

indicating the ADL used at a given time t and the cost change after its application. The update method described in line (9), Algorithm 5 is executed by a simple process: a new pair (ADL_t, Δ_t) is queued in q at a given step t while the oldest entry in the $|w+1|$ position is de-queued (q is a FIFO list of pairs(ADL_t, Δ_t) with size w). This process ends whenever the termination criteria is satisfied. This was fixed as a maximum number of iterations (1000), a value determined by preliminary experiments.

Algorithm 5 *ImprovementStage*

Require: $ls \leftarrow IncumbentSolution$, *Queue* q *ADLsize* n
1: $t = 0$
2: **while** !$LocalTerminationCriteria()$ **do**
3: $\quad ADL_t = constructADL(q, n)$
4: $\quad ls^* = apply(ADL_t, ls)$
5: $\quad \Delta_t = f(ls) - f(ls^*)$
6: $\quad$ **if** $\Delta_t \geq 0$ **then**
7: $\quad\quad ls = ls^*$
8: $\quad$ **end if**
9: $\quad q \leftarrow update(q, ADL_t, \Delta_t)$
10: $\quad t++$
11: **end while**
12: **return** ls

The proposed method for building an ADL in line 3, Algorithm 5 is an online learning approach which keeps track of events associated with improving ADLs. This method utilizes all pairs for which $\Delta > 0$ in the current list q (with non-zero Δ indicating that the associated ADL generated an improvement) to construct a new ADL. If there is no ADL which generates an improving move, then a random feasible string incorporating n add-delete operations is built (line 3) as described in section 3.2.1. If there is a set of ADLs which generate improving moves (latest and most successful add-delete lists reported in q), then these ADLs are taken from this record to identify which events will appear in the new ADL. Depending on the count of each event in this set, a roulette wheel strategy is used to choose events for the new ADL. The probability of each event e_i being chosen $(P(e_i))$ is computed using Equation 3. This equation calculates the probability $P(e_i)$ of an event e_i at a given time to appear in an ADL. This expression counts the number of appearances of a given event e_i from q considering only improving ADLs and divides this number between the total number of all events in it. For example, given $n = 2$, and assuming that we have two ADLs generating an

improvement in q; "-2,-4,4,2" in which distinct events are 2 and 4, and "-3,-4,4,3", in which the distinct events are 3 and 4, counting the number of each distinct event yields a singleton 2, singleton 3 and two 4s. Based on these counts, the roulette wheel strategy chooses two events. Singletons 2 and 3 may be chosen with a probability of 1/4, while 4 may be chosen with a probability of 1/2. Assuming that 4 and 3 are chosen, then a random feasible string is constructed, which could be "-3,3,-4,4".

$$P(e_i) = \frac{Number\ of\ appearances\ of\ e_i\ in\ getImprovingADLs(q)}{Total\ number\ of\ all\ events\ in\ getImprovingADLs(q)} \quad (3)$$

Where $getImprovingADLs()$ returns the ADLs from List q for which $(\Delta > 0)$

Algorithm 6 *constructADL*

Require: *Queue q*, *ADLsize n*
1: **if** $NumberOfOnlyImprovingADLs(q) > 0$ **then**
2: $ADL \leftarrow RouletteWheel(getImprovingADLs(q))$
3: **else**
4: $ADL \leftarrow$ Conversion To Integer ADL($GenerateAddDeleteList(n, "01")$)
5: **end if**
6: **return** *ADL*

The process of constructing an ADL can be viewed as a generative process for building perturbation operators. Once an ADL is built, it is used as a perturbative heuristic which processes and returns a complete solution. Hence, the proposed hyper-heuristic is a generative hyper-heuristic according to the categorization provided in [4]. The ADL construction algorithm is designed in such a way that the events that improves the overall cost of a timetable is preferred in building a new ADL. Using the same events that occur in the improving ADLs does not necessarily result in the same sequence of those events. As long as a given add-delete sequence and events make an improvement, they will be preferred during the search process. If the search stagnates with these events and a sequence of add-delete operators, then a diversification mechanism is applied to create a random feasible sequence of add-delete operators using randomly-chosen events. This construction approach takes a valid ADL binary string as a basis, an event e_i and a random pair (0,1) from the generated ADL to be converted into $(-e_i, e_i)$.

4. Computational Results

In this section, we present the results of the application of our approach to the ITC2007 benchmark.

4.1. Experimental Design and Performance Evaluation

Some initial experiments were executed to observe the influence of the ADL length on the performance of our approach. Then two more sets of experiments were performed for performance evaluation of our approach using the public instances of ITC2007[1], Tracks 2 and 3. The post-enrollment-based course timetabling and curriculum-based course timetabling competition tracks have 24 and 21 instances, respectively.

During the experiments, each trial uses the stopping criterion of ITC2007, which is a *time limit* determined by a benchmark tool for each track. We compare the performance of the proposed approach against top five algorithms reported in the scientific literature as the state of art for each track. The benchmarking tool, algorithms of the competitors, their performances can be seen at the ITC2007 webpage[1]. In all tracks, results are taken from 10 independent trials/runs following the ITC 2007 rules. Intel core i7 machines with 8 gigabytes of Ram and Java™JRE 18.4 on Linux Ubuntu 14.10 are used during the experiments.

In order to compare the average performance of approaches, a non-parametric statistical test is employed. In this study, we use CONTROLTEST, a tool for non-parametric comparison between algorithms [56][2]. Specifically, three non-parametric tests were conducted: Friedman [57], Aligned Friedman [58], and Quade [59] tests. All those tests give an average ranking to each algorithm across a complete dataset, a lower rank signifier a better algorithm. In our study, the null hypothesis H_0 represents no significant differences between algorithms. The *p-values* also indicate how significant the results are: the smaller the *p-value* the stronger the evidence against H_0.

4.2. Tuning the ADL Length

The proposed approach has a crucial parameter, the length of the list. This parameter can be seen as the memory size of our algorithm. A short ADL length causes our hyper-heuristic with ADL (HHADL) approach to

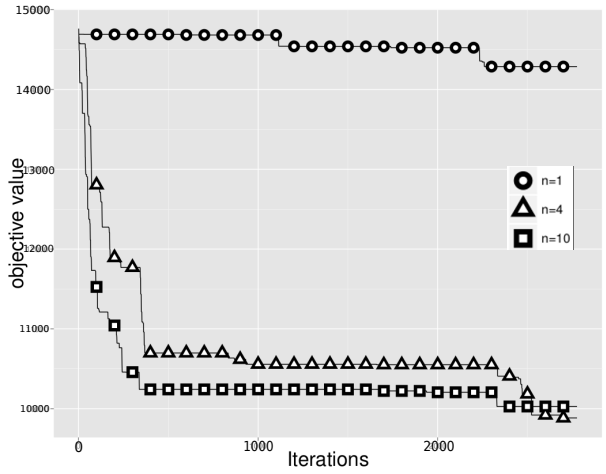
[1] http://www.cs.qub.ac.uk/itc2007/

[2] http://sci2s.ugr.es/sicidm/

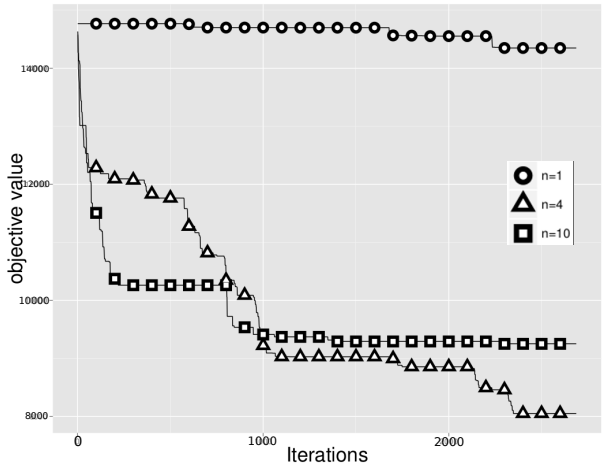
behave like a random selection strategy for the add-delete moves. On the other hand, a long ADL length could misguide the HHADL approach due to some degree of over-training, making it almost impossible to choose the next best add-delete move.

We performed a set of initial experiments to tune the ADL length, denoted as n with n=1, 4 and 10. We observed similar behaviors on all instances from each track: during the early stages of the search process, a longer lists (empirically determined to be $n > 4$) seem to be useful, while towards the final phase of the search process, a list of length $n = 4$ performs better. This phenomenon can be explained by the observation that n essentially parameterizes the intensification-diversification trade-off for our approach. Unfortunately, this simple approach is susceptible to being trapped in local optima. A single increment in the ADL length value can nonetheless guide the search to another point in the fitness landscape.

As a representative example, Figures 8(a) and 8(b) illustrate how the proposed approach behaves on two arbitrarily selected instances of ITC2007, that is Track 2, instance 1 and Track 3, instance 2. The plots show the progress of the number of soft-constraint violations during the search process. Three variants of the algorithm are distinguished only by the ADL length. In both cases, the algorithm using $n = 1$ performs the worst, effectively using random selection for choosing heuristics. The algorithm using $n = 4$ (indicated by triangles) delivers a better performance in balancing intensification and diversification. In the early stages, $n = 4$ intensifies achieving sudden improvements, and then a local optimum is reached. That variant of our approach diversifies and manages to escape. Using $n = 10$ initially performs well, but it is eventually beaten by $n = 4$. As the ADL length increases, the proposed approach consumes more time, so $n = 10$ is the worst setting for the approach in terms of run-time efficiency. Moreover, as Figure 8(b) illustrates, once attracted to a local optimum, it is difficult for this variant to escape, since an ADL of length of 10 generates solutions with many rescheduled events, often leading to improving solutions. Hence, the ADL length is set to $n = 4$ in the following experiments.



(a)



(b)

Figure 8: Performance of the proposed approach for n=1, 4 and 10 on (a) Track 2 - instance 1, (b) Track 2 - instance 3.

4.3. ADL vs Simple heuristics vs Random Selection

We proceed to test the most basic idea of any hyper-heuristic approach presented by Burke et al. [4] : "*different heuristics have different strengths and weaknesses, it makes sense to see whether they can be combined in some way so that each makes up for the weaknesses of another*". In this simulation, we simply compare the performance of our ADL hyper-heuristic with $n = 4$

Table 2: The mean performance of ADL, Random Selection and other Heuristics.

Instance	**HHADL**	*Random*	*Brelaz*	*Min-Conflicts*	*First-Fit*	*Worse-Fit*
Track 2-1	1170.1	1402.12	1430.5	1663.7	1702.3	1812.3
Track 2-4	969.3	1056.7	1125.3	1373.4	1433.2	1571.1
Track 2-7	536.2	635.2	673.1	733.1	814.5	889.3
Track 2-12	618.4	876.4	833.14	866.3	952.6	1215.7
Track 2-15	756.2	824.6	915.8	1063.12	1177.5	1546.6
Track 3-1	62.6	91.8	88.4	92.5	110.3	150.3
Track 3-6	73.5	93.5	106.7	120.8	127.8	166.2
Track 3-8	68.2	84.2	72.1	90	92	134
Track 3-13	110.1	151.6	146.4	157.6	163.3	210.2
Track 3-20	62.7	82.3	79.4	92.4	95.7	115.8

for 1000 iterations against 1000 successive applications of each heuristic alone. This experiment was repeated 30 times on a selected set of instances from ITC2007 Tracks 2 and 3. We ran each experiment with a random initial solution and each approach shared the same starting solution at each trial. Also, we compare the proposed approach to a simple iterative algorithm that randomly chooses and applies a heuristic at each step through 1000 iterations. We include this performance comparison to gather further evidence on the efficiency of our ADL approach as a search methodology.

The experimental results are summarized in Table 2 which shows the mean number of soft constraint violations (Equations (1) and (2)) over 30 trials after running 1000 iterations of ADL, Random Selection and each heuristic. HHADL consistently achieves the best results on all instances. This supports the fact that the success of our approach is not due to a single heuristic: rather it is due to the selection of different heuristics through ADL. Moreover, ADL automatically produces a viable search strategy producing a better mixture of heuristics selection through Random Selection.

4.4. ITC2007 Track 2

Table 3 shows the best results obtained by HHADL and the other competing algorithms in the *post-enrollment* course timetabling problem. Bold values represent the best known results for each instance. HHADL performs the best on the instances 8, 13 and 15, classified as *mid-size* problems. Instances 8 and 15 contain 500 students and 200 subjects each. Instance 13 has 400 students and 300 subjects. One important characteristic is that the proposed algorithm is always capable of finding a *feasible* solution. in contrast some of the competition-specific procedures were not (e.g. *Atsuna*

Table 3: Comparison against state-of-the-art approaches on the 24 *ITC-2007 Track 2* instances. Values indicate the best soft constraint (s) results (out of 10 runs), in all cases solutions are feasible, i.e. the hard constraints are 0.

ITC-2007	*Atsuna*	*Cambazard*	*Ceschia*	*Lewis*	*Jat & Yang*	*HHADL*
1	61	571	**59**	1166	501	630
2	547	993	**0**	1665	342	450
3	382	164	**148**	251	3770	300
4	529	310	**25**	424	234	602
5	5	5	**0**	47	**0**	6
6	**0**	**0**	**0**	412	**0**	**0**
7	**0**	6	**0**	6	**0**	**0**
8	**0**	**0**	**0**	65	**0**	**0**
9	**0**	1560	**0**	1819	989	640
10	**0**	2163	3	2091	499	663
11	548	**178**	142	288	246	344
12	869	**146**	267	474	172	198
13	**0**	**0**	1	298	**0**	**0**
14	**0**	1	**0**	127	**0**	35
15	379	**0**	**0**	108	**0**	**0**
16	191	2	**0**	138	**0**	**140**
17	1	**0**	**0**	**0**	**0**	**0**
18	**0**	**0**	**0**	25	**0**	**0**
19	x	1824	**0**	2146	84	400
20	1215	445	543	625	297	**150**
21	**0**	**0**	5	308	0	**0**
22	**0**	29	5	x	1142	32
23	438	**238**	1292	3101	963	238
24	720	21	**0**	841	274	**640**

for ITC2007-19). Instances from 20 to 24 were categorized as *big* instances having 1,000 students and 400 subjects. This is relevant because despite not being the best solution, our generic approach *HHADL* is still capable of obtaining feasible solutions on these larger instances.

In terms of average performance, a ranking of our approach across all ITC2007-2 datasets can be seen in Table 4. This table is ordered by QUADE descending results, and the p-values computed using our HHADL approach as control method were: Friedman 2.50×10^{-6}, Aligned Friedman 1.12×10^{-4} and Quade 2.64×10^{-7}. We also used *Contrast Estimation* based on medians [60, 61] in this study. This statistical technique assumes that the expected differences between algorithms are the same across problems. The magnitudes in this matrix reflect the performance difference between the pairs of competing algorithms [56]. A positive magnitude in a given row represents that the algorithm has better performance than the algorithm reported in the given column. In the case of a negative magnitude the performance is considered worse. Contrast Estimation for ITC2007 Track 2 experiments can be seen in Table 5. This estimation registers a better performance for our algo-

Table 4: Average Rankings of the algorithms for Track 2

Algorithm	Friedman	Alignment Friedman	QUADE
Ceschia	2.39	48.81	2.23
HHADL	2.87	63.43	3.02
Jat & Yang	2.85	64.52	3.13
Cambazar	3.20	69.33	3.23
Atsuna	3.791	74.70	3.70
Lewis	5.27	114.18	5.26

Table 5: Contrast Estimation for Track 2

	Atsuna	Cambazar	Ceschia	Lewis	Jat	HHADL
Atsuna	0.000	-17.08	-36.92	166.3	-32.00	-28.33
Cambazar	17.08	0.000	-19.83	183.4	-14.92	-11.25
Ceschia	36.92	19.83	0.000	203.3	4.917	8.583
Lewis	-166.3	-183.4	-203.3	0.000	-198.3	-194.7
Jat	32.00	14.92	-4.917	198.3	0.000	3.667
HHADL	28.33	11.25	-8.583	194.7	-3.667	0.000

rithm against *Cambazar*, *Lewis* and *Atsuna* algorithms but *adhoc* designed algorithms like *Ceschia* and *Jat* surpass it. This is understandable since our approach was designed with generality as an objective, whereas other approaches were designed specifically to solve these instances. The overall ranking of our generic approach against the top 5 participants of ITC2007 Track 2 is third.

4.5. ITC2007 Track 3

In this track, the performance of HHADL and the top five algorithms are compared across the *curriculum-based* course timetabling problem instances. The best and average performance of those approaches are provided in Table 6. Values in bold represent the best known results for each instance so far.

The best performance can be seen on instances 12 and 13. For those cases, HHADL was capable of obtaining better results than the other approaches. This is an interesting result since Müller's approach is a generic algorithm applied to the entire ITC2007 dataset, whereas *Asín* and *Cachiani* are both ITC-specific algorithms. Instance 9 can be categorized as a *small* instance with 30 courses, 6 periods per day and small curricula of 14 events. Instances 11, 12 and 13 are *mid-size* instances with an average of 85 courses to assign to 5 periods per day without violations on a curriculum order of 70 courses.

In terms of average performance, Table 7 shows the ranking of our approach based on average performance calculated by Friedman, Aligned Friedman and QUADE test, the $p - values$ computed with $HHADL$ as control algorithm were 2.22×10^{-16}, 3.9×10^{-4} and 1.11×10^{-16}, respectively. The contrast estimation can be seen in Table 8. According to the results, the HHADL approach performs as well as the state-of-the-art entry algorithms.

Table 6: Comparison of best results for ITC2007 Track 3

	Müller (2009)	Lü (2010)	Hao (2011)	Asín (2012)	Cacchiani (2013)	HHADL
ITC2007-1	5	5	4	**0**	5	5
ITC2007-2	60	55	**12**	16	16	15
ITC2007-3	84	71	38	**28**	52	30
ITC2007-4	37	43	**35**	**35**	**35**	**35**
ITC2007-5	330	309	183	**48**	166	68
ITC2007-6	48	53	22	27	**11**	18
ITC2007-7	20	28	**6**	**6**	**6**	**6**
ITC2007-8	41	49	**37**	**37**	**37**	**37**
ITC2007-9	109	105	72	**35**	92	**35**
ITC2007-10	16	21	4	4	**2**	4
ITC2007-11	**0**	**0**	**0**	**0**	**0**	**0**
ITC2007-12	333	343	109	99	100	**90**
ITC2007-13	285	66	59	59	57	**55**
ITC2007-14	61	61	51	51	**48**	50
ITC2007-15	84	71	38	**28**	52	30
ITC2007-16	36	39	16	18	**13**	15
ITC2007-17	83	91	**48**	56	**48**	50
ITC2007-18	83	69	**24**	27	52	27
ITC2007-19	70	65	56	**46**	48	49
ITC2007-20	27	47	**2**	4	4	4
ITC2007-21	103	106	61	**42**	**42**	43

Table 7: Average Rankings of the algorithms for Track 3

Algorithm	Friedman	Alignment Friedman	QUADE
HHADL	2.40	37.64	2.21
Asin	2.47	38.73	2.31
Cacchiani	2.61	51.26	2.72
Hao	2.78	48.30	2.87
Muller	5.30	104.66	5.53
Lu	5.40	100.38	5.42

Table 8: Contrast Estimation for Track 3

	Muller	Lu	Hao	Asin	Cacchiani	HHADL
Muller	0.000	1.333	-26.50	-25.50	-24.17	-28.17
Lu	-1.333	0.000	-27.83	-26.83	-25.50	-29.50
Hao	26.50	27.83	0.000	1.000	2.333	-1.667
Asin	25.50	26.83	-1.000	0.000	1.333	-2.667
Cacchiani	24.17	25.50	-2.333	-1.333	0.000	-4.000
HHADL	28.17	29.50	1.667	2.667	4.000	0.000

5. Conclusions and future work

This study describes a novel generative hyper-heuristic entitled 'Add-Delete Lists' and details their application across two different problem domains that is Tracks 2 and 3 of the 2007 International Timetabling Competition. An Add-Delete list is a 'ruin-and-recreate' sequence which is applied to a solution representation.

The timetabling domain-description used here employs a layer of generality called *Methodology of Design*. This layer represents problem-specifics as a collection of generic structures, such that it can operate across competition tracks without modification (beyond the inevitable translation of problem-specific constraints).

The ADL approach achieved competitive results when compared with the five papers which report state of the art results in Tracks 2 and 3 and in some cases reported best known results (more specifically in Track 3). It is important to note that none of the track winners are the same, due essentially to the existence of the *ad hoc* adaptation of the winning algorithms to specific domains. Conversely, our proposed approach can be applied to both domains and our results were obtained without requiring specific adaptation to a specific competition track. This demonstrates that the ADL approach is genuinely a hyper-heuristic, i.e. competitive across problem types without requiring domain-specific knowledge to be manually embedded in the solver.

With regard to future work, the next step is to experiment with a dynamic or self-adaptive mechanism for the ADL length. In addition, it might be beneficial to investigate a new implementation of the window list used by the ILS in which additional ADL traits are stored. In this paper only two traits (appearance of specific events and event order) were used. In future implementations, the ADL length and the specific CSP heuristic used to

reinsert an event could provide more information, allowing a more-informed ADL construction at each search step.

Acknowledgements

This work was supported by Consejo Nacional de Ciencia y Tecnología (CONACYT) México, the Engineering and Physical Sciences Research Council (EPSRC) Grant GR/S70197/01 and the University of Stirling UK.

[1] E. Özcan, B. Bilgin, E. E. Korkmaz, A comprehensive analysis of hyper-heuristics, Intell. Data Anal. 12 (1) (2008) 3–23.

[2] E. K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Ozcan, R. Qu, Hyper-heuristics: A survey of the state of the art, J Oper Res Soc 64 (12) (2013) 1695–1724.

[3] P. Cowling, G. Kendall, E. Soubeiga, A hyperheuristic approach to scheduling a sales summit, in: E. Burke, W. Erben (Eds.), Practice and Theory of Automated Timetabling III, Vol. 2079 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2001, pp. 176–190.

[4] E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Ozcan, J. Woodward, Handbook of Metaheuristics, Vol. 146 of International Series in Operations Research & Management Science, Springer, 2010, Ch. A Classification of Hyper-heuristic Approaches, pp. 449–468, chapter 15.

[5] S. Even, A. Itai, A. Shamir, On the complexity of timetable and multicommodity flow problems, SIAM J. Comput. 5 (4) (1976) 691–703.

[6] T. B. Cooper, J. H. Kingston, The compexity of timetable construction problems, Ph.D. thesis, The University of Sydney (1995).

[7] R. J. Willemen, School timetable constructrion: Algorithms and complexity, Ph.D. thesis, Institutefor Programming research and Algorithms (2002).

[8] B. McCollum, A. Schaerf, B. Paechter, P. McMullan, R. Lewis, A. J. Parkes, L. D. Gaspero, R. Qu, E. K. Burke, Setting the research agenda in automated timetabling: The second international timetabling competition, Informs Journal on computing 22 (1) (2010) 120–130.

[9] R. Lewis, Metaheuristics for university course timetabling, Ph.D. thesis, University of Notthingham. (August 2006).

[10] E. K. Burke, G. Kendall, E. Soubeiga, A tabu-search hyperheuristic for timetabling and rostering, Journal of Heuristics 9 (6) (2003) 451–470.

[11] H. Fisher, G. L. Thompson, Probabilistic learning combinations of local job-shop scheduling rules, in: J. F. Muth, G. L. Thompson (Eds.), Industrial Scheduling, Prentice-Hall, Inc, New Jersey, 1963, pp. 225–251.

[12] W. B. Crowston, F. Glover, G. L. Thompson, J. D. Trawick, Probabilistic and parametric learning combinations of local job shop scheduling rules, ONR Research memorandum, GSIA, Carnegie Mellon University, Pittsburgh (117).

[13] Y. Hamadi, E. Monfroy, F. Saubion (Eds.), Autonomous Search, Springer, 2012.

[14] R. Battiti, M. Brunato, F. Mascia, Reactive Search and Intelligent Optimization, Vol. 45 of Operations Research/Computer Science Interfaces Series, Springer, 2009.

[15] J. Maturana, F. Lardeux, F. Saubion, Autonomous operator management for evolutionary algorithms, Journal of Heuristics 16 (2010) 881–909.

[16] Y. S. Ong, M. H. Lim, N. Zhu, K. W. Wong, Classification of adaptive memetic algorithms: a comparative study, IEEE Transactions on Systems, Man, and Cybernetics, Part B 36 (1) (2006) 141–152.

[17] M. Birattari, Tuning Metaheuristics: A Machine Learning Perspective, Springer, 2009.

[18] F. Lobo, C. Lima, Z. Michalewicz (Eds.), Parameter Setting in Evolutionary Algorithms, Vol. 54 of Studies in Computational Intelligence, Springer, 2007.

[19] J. Swan, E. Özcan, G. Kendall, Co-evolving add and delete heuristics, in: Proceedings of the Ninth International Conference on the Practice and Theory of Automated Timetabling (PATAT 2012), 2012, pp. 395–399.

[20] G. Schrimpf, J. Schneider, H. Stamm-Wilbrandt, G. Dueck, Record breaking optimization results using the ruin and recreate principle, Journal of Computational Physics 159 (2) (2000) 139–171.

[21] A. Misevicius, Genetic algorithm hybridized with ruin and recreate procedure: application to the quadratic assignment problem, Knowledge-Based Systems 16 (5) (2003) 261–268.

[22] A. Misevicius, Ruin and recreate principle based approach for the quadratic assignment problem, in: Genetic and Evolutionary ComputationGECCO 2003, Springer, 2003, pp. 598–609.

[23] P. Ross, Hyper-heuristics, in: E. K. Burke, G. Kendall (Eds.), Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques, Springer, 2005, Ch. 17, pp. 529–556.

[24] E. K. Burke, M. R. Hyde, G. Kendall, G. Ochoa, E. Özcan, J. R. Woodward, Exploring hyper-heuristic methodologies with genetic programming, in: J. Kacprzyk, L. C. Jain, C. L. Mumford, L. C. Jain (Eds.), Computational Intelligence, Vol. 1 of Intelligent Systems Reference Library, Springer Berlin Heidelberg, 2009, pp. 177–201.

[25] E. Özcan, A. J. Parkes, Policy matrix evolution for generation of heuristics, in: Proceedings of the 13th annual conference on Genetic and evolutionary computation, GECCO '11, ACM, New York, NY, USA, 2011, pp. 2011–2018.

[26] D. de Werra, An introduction to timetabling, European Journal of Operational Research 19 (2) (1985) 151 – 162.

[27] M. Carter, A survey of practical applications of examination timetabling algorithms, Operations Research 34 (1986) 193–202.

[28] G. Lajos, Complete university modular timetabling using constraint logic programming, In E Burke and P Ross editors. Practice and Theory of Automated Timetabling (PATAT) Incs 1153 (1996) 146–161.

[29] P. Boizumault, Y. Delon, L. Peridy, Logic programming for examination timetabling, Logic Program 26 (1996) 217–233.

[30] Z. Lü, J.-K. Hao, Adaptive tabu search for course timetabling, European Journal of Operational Research 200 (1) (2010) 235 – 244.

[31] A. Colorni, M. Dorigo, V. Maniezzo, Metaheuristics for high-school timetabling, Computational Optimization and Applications 9 (1997) 277–298.

[32] E. Yu, K. S. Sung, A genetic algorithm for a university weekly courses timetabling problem, International Transactions in Operational Research 9 (2002) 703–717.

[33] A. Mayer, C. Nothegger, A. Chwatal, G. Raidl, Solving the post enrolment course timetabling problem by ant colony optimization, International Timetabling Compertition 2007.

[34] K. Socha, J. Knowles, M. Samples, A max-min ant system for the university course timetabling problem, in: M. Dorigo, G. D. caro, M. Samples (Eds.), Proceedings of Ants 2002 - Third International Workshop on Ant Algorithms, Lecture Notes in Computer Science, Berlin: Springer-Verlag, 2002, pp. 1–13.

[35] E. Burke, A. Eckersley, B. McCollum, S. Petrovic, R. Qu, Hybrid variable neighbourhood approaches to university exam timetabling, European Journal of Operational Research 206 (1) (2010) 46 – 53.

[36] J. M. Thompson, K. A. Dowsland, A robust simulated annealing based examination timetabling system, Computers and Operations Research 25 (1998) 637–648.

[37] H. Rudová, T. Müller, K. Murray, Complex university course timetabling, Journal of Scheduling 14 (2011) 187–207.

[38] H. Cambazard, E. Hebrard, B. OŚullivan, A. Papadopoulos, Local search and constraint programming for the post enrolment-based course timetabling problem, Annals of Operations Research 194 (2012) 111–135.

[39] E. K. Burke, B. McCollum, A. Meisels, S. Petrovic, R. Qu, A graph-based hyper-heuristic for educational timetabling problems, European Journal of Operational Research 176 (1) (2007) 177 – 192.

[40] J. A. Soria-Alcaraz, H. Terashima-Marin, M. Carpio, Academic timetabling design using hyper-heuristics, Advances in Soft Computing, ITT Springer-Verlag 1 (2010) 158–164.

[41] R. Qu, E. K. Burke, B. McCollum, Adaptive automated construction of hybrid heuristics for exam timetabling and graph colouring problems, European Journal of Operational Research 198 (2) (2009) 392 – 404.

[42] M. Atsuta, K. Nonobe, T. Ibaraki1, Itc-2007 track2: An approach using general csp solver, International Timetabling Compertition 2007.

[43] K. Nonobe, T. Ibaraki, An improved tabu search method for the weighted constraint satisfaction problem, INFOR 39 (2) (2001) 131–151.

[44] S. Ceschia, L. D. Gaspero, A. Schaerf, Design, engineering, and experimental analysis of a simulated annealing approach to the post-enrolment course timetabling problem, Computers & Operations Research 39 (7) (2012) 1615 – 1624.

[45] R. Lewis, A time-dependent metaheuristic algorithm for post enrolment-based course timetabling, Annals of Operations Research 194 (2012) 273–289.

[46] S. Jat, S. Yang, A hybrid genetic algorithm and tabu search approach for post enrolment course timetabling, Journal of Scheduling 14 (6) (2011) 617–637.

[47] T. Müller, Itc2007 solver description: a hybrid approach, Annals OR 172 (1) (2009) 429–446.

[48] Z. Lü, J.-K. Hao, Adaptive tabu search for course timetabling, European Journal of Operational Research 200 (1) (2010) 235–244.

[49] J.-K. Hao, U. Benlic, Lower bounds for the itc-2007 curriculum-based course timetabling problem, European Journal of Operational Research 212 (3) (2011) 464–472.

[50] R. Asín Achá, R. Nieuwenhuis, Curriculum-based course timetabling with sat and maxsat, Annals of Operations Research (2012) 1–21.

[51] V. Cacchiani, A. Caprara, R. Roberti, P. Toth, A new lower bound for curriculum-based course timetabling, Computers & Operations Research 40 (10) (2013) 2466–2477.

[52] H. Lourenço, O. Martin, T. Stützle, Iterated local search, in: F. Glover, G. Kochenberger, F. S. Hillier (Eds.), Handbook of Metaheuristics, Vol. 57 of International Series in Operations Research & Management Science, Springer New York, 2003, pp. 320–353.

[53] A. Soria-Alcaraz Jorge, M. Carpio, H. Puga, M. Sotelo-Figueroa, Comparison of Metaheuristic Algorithms with a Methodology of Design for the Evaluation of Hard Constraints over the Course Timetabling Problem, Vol. 451 of Studies in Computational Intelligence, Springer Berlin Heidelberg, 2013.

[54] A. Soria-Alcaraz Jorge, M. Carpio, H. Puga, M. Sotelo-Figueroa, 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, Vol. 294 of Studies in Fuzziness and Soft Computing, Springer Berlin Heidelberg, 2013, pp. 287–319.

[55] B. Smith, The brlaz heuristic and optimal static orderings, in: J. Jaffar (Ed.), Principles and Practice of Constraint Programming CP99, Vol. 1713 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 1999, pp. 405–418.

[56] J. Derrac, S. García, D. Molina, F. Herrera, A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms, Swarm and Evolutionary Computation 1 (1) (2011) 3 – 18.

[57] M. Friedman, The use of ranks to avoid the assumption of normality implicit in the analysis of variance, Journal of statistical association 32 (1937) 647–701.

[58] M. Friedman, A comparison of alternative tests of significance for the problem of m rankings., Annals of Mathematical statistics 11 (1940) 86–92.

[59] D. Quade, Using weighted rankings in the analysis of complete blocks with additive blocks effects, Journal of the American Statistical Assosiation 74 (1979) 680–683.

[60] K. Doksum, Robust procedures for some linear models with one observation per cell., Annals of Mathematical statistics 38 (1967) 878–883.

[61] S. Garcia, A. Fernandez, J. Luengo, F. Herrera, Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining., Experimental analysis of power, Information science. 180 (2010) 2044–2064.

[62] Jörg Denzinger and Matthias Fuchs and Marc Fuchs, High Performance ATP Systems by Combining Several AI Methods, In proc. fifteenth international joint conference on artificial intelligence (IJCAI 97),1997,pp 102–107.