Noname manuscript No.
(will be inserted by the editor)

Automated Generation of Constructive Ordering Heuristics for Educational Timetabling

Nelishia Pillay · Ender Özcan

Received: date / Accepted: date

Abstract Construction heuristics play an important role in solving combinatorial optimization problems. These heuristics are usually used to create an initial solution to the problem which is improved using optimization techniques such as metaheuristics. For examination timetabling and university course timetabling problems essentially graph colouring heuristics have been used for this purpose. The process of deriving heuristics manually for educational timetabling is a time consuming task. Furthermore, according to the no free lunch theorem different heuristics will perform well for different problems and problem instances. Hence, automating the induction of construction heuristics will reduce the man hours involved in creating such heuristics, allow for the derivation of problem specific heuristics and possibly result in the derivation of heuristics that humans have not thought of. This paper presents generation construction hyper-heuristics for educational timetabling. The study investigates the automatic induction of two types of construction heuristics, namely, arithmetic heuristics and hierarchical heuristics. Genetic programming is used to evolve arithmetic heuristics. Genetic programming, genetic algorithms and the generation of random heuristic combinations is examined for the generation of hierarchical heuristics. The hyper-heuristics generating both types of heuristics are applied to the examination timetabling and the curriculum based university course timetabling problems. The evolved heuristics were found to

This research was funded by a Royal Society Newton International Interchange Grant (NI150199).

Nelishia Pillay
School of Mathematics, Statistics and Computer Science
University of KwaZulu-Natal, South Africa
E-mail: nelishiap@gmail.com

Ender Özcan
ASAP, School of Computer Science
University of Nottingham, UK
E-mail: Ender.Ozcan@nottingham.ac.uk

perform much better than the existing graph colouring heuristics used for this domain. Furthermore, it was found that the while the arithmetic heuristics were more effective for the examination timetabling problem, the hierarchical heuristics produced better results than the arithmetic heuristics for the curriculum based course timetabling problem. Genetic algorithms proved to be the most effective at inducing hierarchical heuristics.

Keywords Educational timetabling · Construction heuristics · Hyper-heuristics · Genetic programming · Genetic algorithms

1 Introduction

Generation construction hyper-heuristics have proven to be effective in deriving construction heuristics that perform better than existing manually derived construction heuristics for various combinatorial optimization problems including production scheduling [3], packing problems [8] and vehicle routing [17]. Genetic programming [9] or variations of genetic programming such as grammatical evolution [12] have chiefly been employed by generation construction hyper-heuristics to create new low-level construction heuristics. This is achieved by combining problem specific attributes with arithmetic and conditional operators. Genetic algorithms have also been used for the automatic induction of construction heuristics [13]. The hyper-heuristic achieves generality in that the same hyper-heuristic is applied to all problem instances. However, the evolved construction heuristics can be *disposable* or *reusable* [4]. Disposable heuristics are induced for a particular problem instance while reusable heuristics can be used to solve unseen problem instances, different from the problem instance/s that it was created for.

The research presented in this paper investigates the automatic derivation of construction heuristics for educational timetabling problems. While there has been a fair amount of research into the use of hyper-heuristics for educational timetabling, there has not been much research into the automatic generation of construction heuristics for this domain. Furthermore, there is very little work using a genetic programming or genetic algorithm hyper-heuristic for this purpose despite the success of genetic programming hyper-heuristics in inducing construction heuristics for other domains such as production scheduling and packing. The generation hyper-heuristic implemented by Bader-El-Den et al. [1] uses grammar-based genetic programming [11] to evolve construction heuristics for the examination timetabling problem. Existing construction heuristics, namely the graph colouring heuristics, for the examination timetabling problem are decomposed into their basic components and recombined together with period selection heuristics, arithmetic operators and an if-then-else operator to create new heuristics. The hyper-heuristic was used to generate disposable heuristics for the Toronto benchmark set. A similar study is conducted in [14] in which genetic programming is used to evolve construction heuristics in the form of rules for the examination timetabling problem. Each low-level heuristic represents a comparator

which decides which of two examinations should be given priority. Hence, each heuristic is a nested if-then-else operator with the conditions comprised of existing construction heuristics and arithmetic logical and logical operators. This hyper-heuristic also produced disposable heuristics for five problem instances of the Toronto benchmark set. In [15] construction heuristics are evolved for the school timetabling problem using genetic programming. Each heuristic is a priority function which assigns a numerical value to each class-teacher tuple to be allocated. The tuples are sorted in ascending order and allocated accordingly. Each heuristic is comprised of problem characteristics, e.g. the number of class-teacher meetings a teacher is involved in and arithmetic operators and an if-then-else operator. The generated disposable heuristics produced feasible timetables for all the problem instances in the Abramson benchmark set [2].

The study presented in this paper investigates the automatic generation of construction heuristics for educational timetabling problems. As in the study by Bader-El-Den et al. [1] the evolved heuristics are disposable. Two hyper-heuristics are examined, one generating arithmetic heuristics(AHH) and the second hierarchical heuristics (HHH). The hyper-heuristics were applied to two examination timetabling benchmark sets, namely, the Toronto benchmark set and the ITC 2007 benchmark set, as well as the ITC 2007 curriculum based course timetabling benchmark set. The construction heuristics evolved by the hyper-heuristic performed better than the existing heuristics for both the domains of examination timetabling and curriculum based timetabling. The study also revealed that the AHH produced better results for the examination timetabling problems while the hierarchical heuristics performed better for the curriculum based course timetabling problem.

The following contributions are made by the research presented in the paper:

- To the authors' knowledge this is the first detailed study investigating the use of genetic programming and genetic algorithm hyper-heuristics for educational timetabling in general, i.e. more than one timetabling problem. The use of genetic programming hyper-heuristics has been well researched for production scheduling [3] and packing problems [8], clearly showing its effectiveness for the automatic generation of construction heuristics for these domains. Furthermore, previous research into the use of genetic programming construction hyper-heuristics has been focussed on examination timetabling. To the authors' knowledge, genetic algorithms has not previously been investigated for automatic heuristic induction for educational timetabling.
- While previous research has examined the automatic induction of heuristics in the form of heuristic functions and rules using genetic programming hyper-heuristics, this is the first study investigating hierarchical heuristics.
- To the authors' knowledge this is the first study examining the use of genetic programming and genetic algorithm hyper-heuristics to evolve construction heuristics for the curriculum based course timetabling problem.

– This is the first study applying genetic programming and genetic algorithm construction hyper-heuristics to the ITC 2007 examination timetabling and curriculum based course timetabling benchmark sets.
– The study has revealed that different heuristics, namely, arithmetic and hierarchical, are effective for different educational timetabling problems.

The following section presents a critical reflection on generation construction hyper-heuristics in the context of educational timetabling, highlighting the purpose of generation construction hyper-heuristics, evaluation of the performance of these hyper-heuristics, and generality of the derived construction heuristics for educational timetabling. Section 3 provides an overview of educational timetabling and section 4 describes the generation construction hyper-heuristics for educational timetabling. The experimental setup used to evaluate the performance of the hyper-heuristics is presented in section 5 and the performance of the hyper-heuristics are discussed in section 6. The findings of the study are summarized in section 7 together with future extensions of the work.

2 Generation Construction Hyper-Heuristics

The role that generation construction hyper-heuristics play and hence how these hyper-heuristics should be evaluated is not consistent in the literature, with literature often being contradictory. Hence, this section provides a critical reflection, revisiting the purpose of generation construction hyper-heuristics, performance evaluation and generality. These aspects are examined in the context of deriving construction heuristics for educational timetabling.

2.1 Purpose of Generation Construction Hyper-Heuristics for Educational Timetabling

The main aim of generation construction hyper-heuristics is to automate the process of deriving low-level construction heuristics that are used to create an initial solution to the problem. This initial solution is then optimized by search techniques such as metaheuristics. As such the initial solution provides the search technique with a more informed starting point than a randomly created initial solution. The low-level construction heuristics are usually manually derived. This is a time-consuming process and hence the need for automation [3] [7]. Furthermore, by the no free lunch theorem and as shown in previous research in this field, different construction heuristics are more effective for some problem instances than others, however manually deriving heuristics specific to problem instances is not feasible. Furthermore, it is anticipated that the automated induction of such heuristics could result in the induction of heuristics that a human may not think of [8].

2.2 Performance Evaluation

As mentioned in the previous section the aim of the heuristics generated by the hyper-heuristic is to create an initial solution which is optimized further using optimization techniques to get a final solution. As such the performance of the evolved heuristics can not be expected to be comparable to state of the art techniques for the problem domain [6] [7]. The performance of the automatically induced low-level heuristics should be at least as good as that of the existing manually derived heuristics [5]. Furthermore, evolving these heuristics should not take as long as required to manually derive them [5]. Bader-El-Den et al. [1] state that it should take a few hours to at most a day for a heuristic to be induced. Hence, the performance of generation construction hyper-heuristics should be assessed in terms of how well the evolved construction heuristics perform compared to existing manually derived heuristics as well as the time taken to generate these heuristics.

2.3 Generality

Hyper-heuristics aim to achieve generality [4]. In the context of generation construction hyper-heuristics this is achieved by the same technique and set of parameters being used to evolve construction heuristics for different problem instances and problems. The heuristic evolved can however be either disposable or reusable as explained in [4]. Hyper-heuristics producing disposable heuristics perform online learning and the heuristic is tailored for the problem instance as in the studies conducted in [8] for packing problems, [1] for examination timetabling and [17] for vehicle routing. In the case of reusable heuristics the hyper-heuristic performs offline learning on a subset of problem instances, namely, the training set instead of a single problem instance. The evolved heuristics are applied to unseen problem instances which form a test set. Usually, problems instances for the particular domain are placed into classes with similar problem instances grouped together [3] [8]. The instances in each class are divided into training and testing sets and different low-level construction heuristics are produced for each class of problem instances.

3 Educational Timetabling Problems

Educational timetabling problems require events involving students to be allocated to timetable periods. In the domain of examination timetabling the events are examinations and for university course timetabling these are lectures for the different courses. Allocations must be made so as to satisfy certain problem constraints. Constraints that must be met are referred to as *hard constraints*. Timetables satisfying all hard constraints are *feasible* timetables. Examples of hard constraints include that all events must be scheduled, there must be no clashes, i.e. students, rooms, or teachers, must not be scheduled

more than once in a period. Soft constraints are constraints that it may not be possible to meet as these could be contradictory and we aim to minimize the number of soft constraints violated. The cost is a measure of the *quality* of the timetable, a lower cost indicating a better timetable.

This study investigates the automated derivation of construction ordering heuristics for educational timetabling problems. The genetic programming hyper-heuristics presented are tested on two educational timetabling domains, namely, examination timetabling and curriculum based university course timetabling. Two benchmark sets that are commonly used in the literature for examination timetabling are the Toronto benchmark set and the ITC 2007 benchmark set for the track on examination timetabling. Curriculum based university course timetabling was also a track for the second international timetabling competition (ITC 2007) and the benchmark set for this track is generally used to evaluate methods for solving the this problem. This section provides an overview of these benchmark sets which are used in this study. The section also looks at the role that low-level construction heuristics play in solving educational timetabling problems.

3.1 Toronto Benchmark Set

Table 1 lists the details of the problem instances in the Toronto benchmark set [16] used in this study. The density of the conflict matrix is a measure of the difficulty of the problem instance and is the ratio of the number of students that could potentially be involved in clashes to the total number of students. The hard constraints for the benchmark are that all examinations must be allocated and there must be no clashes, i.e. no student must be scheduled to write more than one examination in the same period.

The soft constraint for the benchmark set is that examinations must be well spread for all students. The proximity cost function in equation (1) below is used to calculate the soft constraint cost.

$$\frac{\sum w(|e_i - e_j|)N_{ij}}{S} \tag{1}$$

where: $|e_i - e_j|$is the distance between the periods of the examination pair e_i and e_j which have students in common; S is the total number of students; N_{ij} is the number of students common to both examinations; $w(1)=16$, $w(2)=8$, $w(3)=4$, $w(4)=2$, $w(5)=1$, $w(n)=0$ for $n > 5$.

3.2 ITC 2007 Examination Timetabling Benchmark Set

The details of the ITC 2007 benchmark set [10] are listed in Table 2. This benchmark set is more constrained than the Toronto benchmark set and represents the capacitated version of the examination timetabling problem.

The hard constraints for the benchmark set are:

Table 1: Toronto benchmark set

Instance	Institution	No. of Periods	No. of Exams	No. of Students	Density of Conflict Matrix
car-f-92 I	Carleton University	32	543	18419	0.14
car-f-91 I	Carleton University	35	682	16925	0.13
ear-f-83 I	Earl Haig Collegiate Institute	24	190	1125	0.27
hec-s-92 I	Ecoles des Hautes Etudes Commerciale	18	81	2823	0.42
kfu-s-93	King Fahd University of Petroluem and Minerals	20	461	5349	0.06
lse-f-91	London School of Economics	18	381	2726	0.06
pur-s-93 I	Purdue University	43	2419	30029	0.03
rye-s-93	Ryerson University	23	486	11483	0.08
sta-f-83 I	St Andrews Junior High School	13	139	611	0.14
tre-s-92	Trent University	23	261	4360	0.18
uta-s-92 I	Faculty of Arts and Sciences University of Toronto	35	622	21266	0.13
ute-s-92	Faculty of Engineering University of Toronto	10	184	2749	0.08
yor-f-83 I	York Mills Collegiate Institute	21	181	941	0.29

Table 2: ITC 2007 examination timetabling benchmark set

Instance	Exams	Students	Periods	Rooms	Conflict Density (%)
Exam1	607	7891	54	7	5.05
Exam2	870	12743	40	49	1.17
Exam3	934	16439	36	48	2.62
Exam4	873	5045	21	1	15.00
Exam5	1018	9253	42	3	0.87
Exam6	242	7909	16	8	6.16
Exam7	1096	14676	80	15	1.93
Exam8	598	7718	80	8	4.55

- There are no clashes, i.e. no students should be scheduled to write more than one examination at the same time.
- Room capacities must not be exceeded.
- Period durations must not be exceeded.
- Ordering requirements must be met, e.g. one examination being scheduled after another, two examinations being scheduled simultaneously.
- Room requirements must be met, e.g. an examination has to be scheduled in a particular venue such as Computer Science in a laboratory.

The following soft constraints are minimized:

- Two in a row: a student is scheduled to write two examinations successively.
- Two in a day: a student is scheduled to write two examinations in a day but the examinations are not successive.
- Period spread: Aims at ensuring that the examinations are well spread for each student. A penalty is added to the soft constraint cost for each examination scheduled within a specified period for a student.
- Mixed durations: A penalty is added to the soft constraint cost for each instance of examinations of different durations being scheduled in the same period in the same venue.
- Largest examinations at the beginning of the examination period: The aim is for the largest examinations to be scheduled at the beginning of the examination period. For each instance the beginning of the examination period is defined as the first n periods and the number of examinations that are in the category "largest examinations". If one of these examinations is not scheduled in the specified periods a penalty is added to the soft constraint cost.
- Room penalty: A penalty is associated with using certain rooms in an attempt to minimize the usage of certain rooms. If one of these rooms is used a penalty is added to the soft constraint cost.
- Period penalty: As with rooms there are also penalties associated with using certain periods. If examinations are scheduled in one of these periods a penalty is added to the soft constraint cost.

3.3 ITC 2007 Curriculum Based Course Timetabling Benchmark Set

The problem instances in the ITC 2007 curriculum based university course timetabling benchmark set [10] are listed in Table 3. The conflicts column lists the conflicts per lecture. This value is the number of pairs of potential conflicts divided by the total number of lectures.

The hard constraints are:

- Lecture allocations - The number of lectures specified for each course must be timetabled.
- Conflicts - Lectures for courses in a curriculum must be scheduled in different periods.
- RoomOccupancy - Each room must only be scheduled once in a period.

Table 3: ITC 2007 CB-CTT Problem Instances

Instance	Courses	Rooms	Curricula	Days	Periods	Conflicts
comp01	30	6	14	6	5	13.2
comp02	82	16	70	5	5	7.97
comp03	72	16	68	5	5	8.17
comp04	79	18	57	5	5	5.42
comp05	56	9	139	6	6	21.7
comp06	108	18	70	5	5	5.24
comp07	131	20	77	5	5	4.48
comp08	86	18	61	5	5	4.52
comp09	76	18	75	5	5	6.64
comp10	115	18	67	5	5	5.03
comp11	30	5	13	9	5	13.8
comp12	88	11	150	6	6	13.9
comp13	82	9	66	5	5	5.61
comp14	85	17	60	5	5	6.87
comp15	72	16	68	5	5	8.17
comp16	108	20	71	5	5	5.13
comp17	99	1	70	5	5	5.5
comp18	47	9	52	6	6	13.3
comp19	74	16	66	5	5	7.45
comp20	121	19	78	5	5	5.06
comp21	94	18	78	5	5	6.09

– Availabilities - Each teacher must be scheduled once in a period.

The soft constraints are:

– RoomCapacity -Room capacities must not be exceeded.
– MinimumWorkingDays - For some courses a minimum number of working days is specified. In this case the lectures for the course must not be scheduled over more than the specified minimum number of days. If this occurs a penalty is incurred.
– CurriculumCompactness - Lectures for courses belonging to the same curriculum should be scheduled adjacent to each other on a day.
– RoomStability - All the lectures for a course should be scheduled in the same venue.

3.4 Low-Level Construction Heuristics and Educational Timetabling

Low-level construction heuristics are used to create an initial solution to an educational timetabling problem. This initial solution forms input to an optimization technique which is used to solve the problem. For educational timetabling problems graph colouring heuristics have generally been used to create an initial solution. The low-level heuristic is a measure of the difficulty of scheduling an event, e.g. an examination or a lecture. The events to be scheduled are sorted in either ascending or descending order according to the heuristic and allocated to timetabling periods in order. The existing manually derived heuristics which are used for educational timetabling include:

- Largest enrolment (LE) - The events to be scheduled are sorted in descending order according to the number of students involved in the event. Events with a larger number of students are given priority to be scheduled.
- Largest degree (LD) - The number of potential clashes, i.e. other events the event has students in common with, is used to assess the difficulty of scheduling the event. The higher the number of clashes the more difficult the event is to schedule and is hence given priority.
- Largest weighted degree (LWD) - This heuristic is similar to the the largest degree heuristic but instead of counting the number of events an event has students in common with, it counts the number of students involved in both events. Events with a higher largest weighted degree are given priority.
- Largest colour degree(LCD) - The largest colour degree is a variation of the largest degree heuristic and is the number of potential clashes with events that have already been scheduled.
- Saturation degree (SD) - The saturation degree heuristic is the number of feasible periods, i.e. periods not resulting in hard constraint violations when the event is scheduled in it, at the current point of timetable construction. Events with fewer periods are scheduled first.

The first three heuristics are *static* heuristics and the values of these heuristics is determined prior to constructing the timetable and remain the same for each event throughout the process of timetable construction. The algorithm for constructing a timetable using a static low-level construction heuristic is illustrated in 1. All the events are sorted in ascending or descending order according to the heuristic value and are allocated in order to the timetable. The aim is to schedule each event to a feasible period p. If a feasible period cannot be found the event is either not scheduled or it is scheduled to a randomly selected period and the hard constraint cost is incremented. If more than one timetable period is feasible the period can be selected using a *period selection heuristic*. Examples of period selection heuristics include:

- First period (f) - The event is scheduled in the first feasible period found.
- Random period (r) - The period is randomly selected from the feasible periods.
- Minimum cost period (m) - The soft constraint cost of feasible periods is calculated given the current partial timetable. The event is scheduled in the period with the minimum soft constraint cost.

Algorithm 1 Timetable construction using a static heuristic

```
1: procedure CREATE TIMETABLE(Events[])
2:     Calculate the heuristic value h_i for each event e_i in Events[]
3:     Sort the events in Events[] according to h_i
4:     for i ← 1, n do
5:         Allocate event Event[i] to a timetable period p
6:     end for
7: end procedure
```

The saturation degree and largest colour degree heuristics are *dynamic* heuristics as the value for each unallocated event changes depending on the current partially created timetable. At the beginning of the timetable construction process all events have the same saturation degree, namely, the number of periods in the timetable, as there has not been allocations as yet. As the timetable is constructed the saturation degree of an event e that has students in common with the event just allocated is decreased if the event is scheduled in a period that does not contain other events in common with e. Similarly, the largest colour degree heuristic has an initial value of zero for all events. Once events are allocated the largest colour degree heuristic is incremented for an event e that has students in common with the event scheduled. The algorithm for using a dynamic heuristic to create a timetable is depicted in 2.

Algorithm 2 Timetable construction using a dynamic heuristic

```
1: procedure CREATE TIMETABLE(Events[])
2:     Specify an initial value for the heuristic value h_i for each event e_i in Events[]
3:     Sort the events in Events[] according to h_i
4:     while Events[] <> null do
5:         Allocate event Event[0] to a timetable period p
6:         Remove Event[0] from Events
7:         Update the heuristic value h_i for each unallocated event e_i in Events[]
8:     end while
9: end procedure
```

4 Evolving Construction Heuristics for Educational Timetabling

This section describes the generation hyper-heuristics employed to induce low-level construction heuristics for educational timetabling problems. The genetic programming hyper-heuristic inducing arithmetic heuristics is presented in section 4.1. The genetic programming, genetic algorithm and random generation hyper-heuristics inducing hierarchical heuristics are described in section 4.2. Both hyper-heuristics employ the generational algorithm to evolve heuristics [9]. The algorithm begins by creating an *initial population*. This initial population is iteratively refined by means of *evaluation*, *selection* and *regeneration* until a termination criterion is met. Each iteration is referred to as a *generation*. The termination condition used in this study is a maximum number of generations. The mutation and crossover operators are applied during regeneration to create the offspring of each generation.

4.1 Arithmetic Hyper-Heuristic (AHH)

This section describes the processes of initial population generation, evaluation and selection and regeneration in evolving low-level arithmetic construction heuristics in the following subsections.

4.1.1 Representation and Initial Population Generation

Each element of the genetic programming population is an expression tree representing a heuristic. A heuristic is composed of problem specific *attributes* that are combined with operators. The operators are arithmetic operators and an if-then-else operator including relational operators for use with this operator. Each heuristic is either an *arithmetic function* or an *arithmetic rule*. An arithmetic function combines the problem attributes with arithmetic operators such as addition and subtraction. An arithmetic rule is composed of a condition and two actions, if the condition is met the first action is performed and the second action otherwise. The actions are arithmetic expressions or rules. The condition combines problem attributes and relational operators and hence the action performed depends on the relation between the attributes specified in the condition. The operators form the function set and the problem attributes the terminal set. The function set used is depicted in table 4. The operators include the standard arithmetic operators. The division operator performs protected division which returns a value of one in the case of the denominator being zero. Relational operators are included in the set for use with the if-then-else operator.

Table 4: Function set

Type	Operators	Description
Arithmetic	+, -, *, /	Perform standard arithmetic operations.
Relational	<, >, <=, >=, ==, !=	Perform the standard relational operators. Used with the conditional operators.
Conditional	if-then-else	Perform the standard function of an if-then-else statement.

Each expression tree in the population is created by randomly selecting elements from the function and terminal sets until a specified maximum tree depth is reached. At the maximum permitted tree depth only elements from the terminal set are selected. The terminal set contains variables representing the problem attributes. While the function set remains the same from one problem to the next the terminal set differs for each problem domain or benchmark set. Factors taken into consideration when choosing the attributes to include are the problem definition, hard constraints and soft constraints. This is done manually and is one of the challenges in automating the process of heuristic derivation. As highlighted by Branke et al. [3] sufficient attributes need to be included to sufficiently represent the problem, however using too many attributes will increase the search space and hence runtimes. Table 5 lists the terminal set used for the Toronto benchmark set. The first column specifies the terminal and the second column the corresponding description. The first attribute a represents a problem characteristic related to the soft

constraint of the problem, namely, the examinations must be well spread for each student. There terminals b to f represent attributes that are related to the hard constraint that there must be no clashes. The last two terminals represent general problem characteristics, the total number of students and the number of periods.

Table 5: Terminal set for the Toronto benchmark set

T	**Description**
a	Measure of the distance between slots of examinations that share students. Is calculated to be the sum of the weighted distances using the following weights, where d is the distance and w the weight: w(1)=16; w(2)=8;w(3)=4;w(4)=2;w(5)=1;w(d)=0 where d<5. Initially all examinations have a value of zero for this attribute.
b	Number of potential clashes the examination has with unallocated examinations, i.e. the number of unallocated examinations that it has students in common with.
c	Number of potential clashes for the examination, i.e. the number of examinations it has students in common with.
d	Number of students taking the examination.
e	Number of periods in the timetable that the examination can be allocated to which will not result in hard constraint violations.
f	Number of potential clashes the examination has with allocated examinations, i.e. the number of allocated examinations that it has students in common with.
g	The total number of students.
h	The number of periods.

The terminal set used for the ITC 2007 examination timetabling problem is illustrated in table 6. Terminals a to f and h to j represent problem characteristics related to the hard constraints of the problem, while the terminal g is related to soft constraints of the problem. The last two terminals represent general problem characteristics. Examinations with a lower room degree (h) are scheduled first. However, certain rooms have a penalty associated with them which is added to the soft constraint cost whenever the room is used. Hence, a penalty of 0.5 is subtracted from the room degree if one of the available rooms has a penalty associated with it to indicate a more difficult exam to schedule. Similarly, certain periods have a penalty associated with using the period and a penalty of 0.5 is subtracted for each available period with a penalty.

The terminal set used for the ITC 2007 curriculum based course timetabling problem is listed in table 7. As in the previous two benchmark sets the terminals represent attributes related to the problem in general, the hard constraints and the soft constraints. Terminals p, n and m represent general problem characteristics, c is related to a hard constraint of the problem and the remaining terminals to soft constraints.

An example of an element of the population, namely an arithmetic function, for examination timetabling is illustrated in Fig. 1. In this example the heuristic is the product of the number of potential clashes for an examination

Table 6: Terminal set for the ITC 2007 examination timetabling benchmark set

T	Description
a	Number of potential clashes for the examination, i.e. the number of examinations it has students in common with.
b	Number of students involved in potential clashes for the examination, i.e. the number of students the examination has in common with other examinations.
c	Number of students taking the examination.
d	Number of periods in the timetable that the examination can be allocated to which will not result in hard constraint violations.
e	Number of potential clashes the examination has with unallocated examinations, i.e. the number of unallocated examinations that it has students in common with.
f	Number of potential clashes the examination has with allocated examinations, i.e. the number of allocated examinations that it has students in common with.
g	Measure of the distance between slots of examinations that share students Is calculated to be the sum of the weighted distances using the following weights, where d is the distance and w the weight: w(1)=16; w(2)=8;w(3)=4;w(4)=2;w(5)=1;w(d)=0 where d<5. Initially all examinations have a value of zero for this attribute.
h	Room degree, i.e. the number of rooms with the required capacity for the examination. A value of 0.5 is subtracted for each room with a penalty associated with using it.
i	Period degree, i.e. the number of periods with the required duration for the examination. A value of 0.5 is subtracted for each period with a penalty associated with using it.
j	The hard constraints include certain sequence requirements, e.g. one examination must take place before another. This attribute is the sum of these requirements.
k	The total number of periods.
l	The total number of rooms.

Table 7: Terminal set for the ITC 2007 curriculum based course timetabling benchmark set

T	Description
s	Number of periods in the timetable that the lecture can be allocated to which will not result in hard constraint violations.
l	Number of potential clashes for the lecture.
d	Number of lectures for the course.
u	Number of students enrolled for the lecture.
c	Minimum number of working days over which the lectures for a course must be distributed.
r	Room degree i.e. the number of rooms with sufficient capacity for the the number of students enrolled for the course.
t	Teacher degree, i.e. the number of courses the teacher allocated to the course is required to teach.
g	Number of potential clashes for the lecture.
v	The number of periods during which the lectures for the course should not be scheduled.
p	The number of periods.
n	The total number of lectures for the course.
m	The total number of rooms.

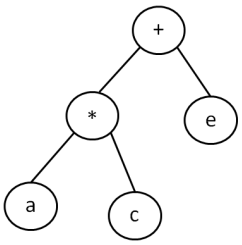

Fig. 1: Example of an arithmetic heuristic

and the number of students taking the examination added to the number of potential clashes with examinations not as yet allocated to the timetable.

The grammar below illustrates the syntax permitted for arithmetic functions and rules for using the terminal set for Toronto examination timetabling problem.

⟨*start*⟩ ::= ⟨*expr*⟩ ⟨*op*⟩ ⟨*expr*⟩ | if ⟨*cond*⟩ ⟨*expr*⟩ ⟨*expr*⟩
⟨*cond*⟩::=⟨*expr*⟩ ⟨*relop*⟩ ⟨*expr*⟩
⟨*expr*⟩ ::= ⟨*expr*⟩ ⟨*op*⟩⟨*expr*⟩ | if ⟨*cond*⟩ ⟨*expr*⟩ ⟨*expr*⟩ | ⟨*term*⟩
⟨*term*⟩::= a | b | c | d | e | f | g
⟨*op*⟩::= + | - | * | /
⟨*relop*⟩::= <| >| ≤ | ≥ | == | !=

When applying this to another problem all that will change are the terminals in the $< term >$ production rules. For example, for course timetabling these would be $< term >::= s|l|d|u|c|r|t|g|v|p|n|m$.

4.1.2 Evaluation and selection

Evaluating the population involves calculating the fitness of each element of the population. The fitness of an individual is calculated by firstly using the heuristic to create a timetable. The algorithm used to create a timetable using the heuristic is depicted in Algorithm 3. The algorithm begins by calculating the heuristic value for each event, i.e. examination or lecture, using the individual. The events are sorted in descending order according to this heuristic value. The first event in the sorted list is allocated to a feasible period. If there is more than one feasible period, the period with the minimum penalty is chosen. The minimum penalty is the soft constraint cost resulting from allocating the event to the period. If there is more than one period with the minimum penalty cost, a period is randomly chosen from these periods. If a feasible period cannot be found the event is not allocated and the hard constraint cost is incremented.

In applying search to solving timetabling problems the fitness or objective value generally used is the sum of the hard and soft constraint cost. However, we have ascertained empirically that taking the product of the hard constraint

Algorithm 3 Timetable construction using a low-level heuristic

```
1: procedure CreateTimetable(Events[], HeuristicTree)
2:     Calculate the heuristic value using HeuristicTree for each event in Events[]
3:     Sort Events in descending order according to heuristic
4:     while there are unallocated exams in Events[] do
5:         Find the list Options of feasible periods that Events[0] can be allocated to
6:         Allocate Events[0] to the period in Options with the minimum penalty cost
7:         if there is more than on feasible minimum penalty period then
8:             Randomly select one of these periods to allocate Event[0]
9:         end if
10:        if there are no feasible periods for Events[0] then
11:            Increment the hard constraint cost
12:        end if
13:        Remove Events[0] from Exams
14:        Recalculate the heuristic values for each exam in Events using HeuristicTree
15:        Sort the Events in descending order according to heuristic
16:    end while
17: end procedure
```

cost plus one and the soft constraint is better at distinguishing between different timetables, i.e. which of two timetables is better and would lead to further improvements in successive generations, than taking the sum of both the costs. The use of a Pareto fitness function which does a vector comparison of the hard and soft constraints, i.e. if the hard constraint costs are the same the timetable with a lower soft constraint is better, was also investigated. However, this did not provide an improvement on taking the product of the hard constraint cost plus one and the soft constraint cost. Hence, the fitness assigned to the individual is a function of the hard and soft constraint cost of the timetable constructed using the heuristic depicted in equation 2.

$$Fitness = (hcv + 1) * scv \tag{2}$$

where:

- hcv is the number of hard constraint violations
- scv is the number of soft constraint violations

Tournament selection [9] is used to choose the parents (individuals) from the population for regeneration. This selection method randomly chooses t individuals from the population representing a tournament. The fittest individual in the tournament is returned as a parent. Selection is with replacement so an individual may be chosen more than once to be a parent.

4.1.3 Regeneration

The standard mutation and crossover operators described in [9] are applied to selected parents to produce the offspring of each generation. The mutation operator randomly chooses a mutation point in the expression tree and the subtree rooted at this point is replaced by a randomly created tree. The crossover operator randomly selects crossover points in each of the parents and the subtrees rooted at these points are swapped to create two offspring.

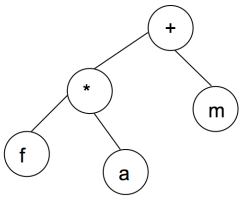

Fig. 2: An example of a hierarchical heuristic

4.2 Hierarchical Hyper-Heuristic (HHH)

This section describes the hyper-heuristic employed to induce hierarchical heuristics. Three approaches were investigated for this, namely, genetic programming using a parse tree representation for each heuristic; genetic algorithms using string representation and random generation using a string representation. The latter was investigated in order to determine whether optimization is necessary to produce effective hierarchical heuristics or whether it would be sufficient to randomly generate these.

4.2.1 Genetic Programming Hyper-Heuristic (HHH-GP)

Each heuristic in HHH-GP combines problem attributes with a period selection heuristic. The function set consists of two elements, namely, $*$ and $+$. The terminal set is comprised of the terminal set used by AHH as well as variables representing the three period selection heuristics, namely, first period (f), random period (r) and minimum cost period (m). The $*$ is used to combine problem attributes. The attributes are applied hierarchically, i.e. the second attribute is used to break ties of the first attribute and the third attribute is used to break ties of the second attribute. The $+$ is used to combine the terminals representing problem attributes with one of the period selection heuristics. The attributes are applied logically in a hierarchical manner, namely, the first attribute is applied and in the case of a tie the second characteristic is applied and so on. An example of a hierarchical heuristic for examination timetabling is illustrated in Fig. 2. The events to be scheduled will be sorted according to the value of the attribute *f*, i.e. the number of potential clashes for the examination with allocated examinations. The attribute *a*, i.e. the number of potential clashes the examination is involved in, will be used to break ties in cases where two events have the same value of *f*. The event will be scheduled in the minimum cost period.

The grammar below depicts the syntax of heuristics for the Toronto examination timetabling problem:

$\langle start \rangle ::= + \langle attrib \rangle \langle psh \rangle$
$\langle attrib \rangle ::= * \langle at \rangle$
$\langle at \rangle ::= \langle esh \rangle \mid \langle esh \rangle \langle at \rangle$

$\langle esh \rangle$ :: a | b | c | d | e | f | g | h
$\langle psh \rangle$::= f | r | m

The only difference in the grammar for a different problem is the terminal values for $< esh >$.

As for AHH the fitness of each heuristic is determined by using it to construct a timetable. The process is as outlined in section 4.1.2. The mutation and crossover operators are used to create the offspring of each generation. Strong typing is used to ensure that syntactically correct offspring are created. For example, the $+$ node can only appear as the root of a tree and a $*$ node as the left child of the $+$. Similarly, attribute nodes and period selection nodes can only be replaced with nodes of the same type.

4.2.2 Genetic Algorithm Hyper-Heuristic (HHH-GA)

Like HHH-GP, HHH-GA is generational with the heuristics evolved over a set number of generations. In the HHH-GA each heuristic is a combination of n problem attributes and a period selection heuristic. The problem attributes are the terminals used by the HHH-GP listed in tables 5, 6 and 7. However, instead of this being represented as a parse tree it is represented as a string. For example, the equivalent string representation for the heuristic in figure 2 is *fam*. Hence, each chromosome is a string representing a hierarchical heuristic. Each element of the population is created by randomly selecting characters representing the problem attributes for the first n characters of the string and the remaining character is randomly selected from the set representing the period selection heuristics. The chromosomes are of fixed length.

As in HHH-GP tournament selection is used to choose parents. The fitness of each chromosome is determined by using the heuristic it represents to create a solution to the problem. The fitness is calculated to be a function of the hard and soft constraint cost as calculated in equation 2.

The mutation and crossover operators are used to produce offspring. The mutation operator replaces a randomly selected character c in the chromosome with a a randomly selected character from the set of problem attributes if c is one of the first n characters of the chromosome and from the set of period selection heuristics if c is the last character in the chromosome. The crossover operator crosses over two chromosomes at a randomly selected crossover point to produce offspring of the same size.

4.2.3 Random Generation Hyper-Heuristic (HHH-RG)

The random generation hyper-heuristic randomly creates strings comprised of n characters representing the problem attributes and the last character representing a period selection heuristic. This hyper-heuristic basically performs the initial population generation of the genetic algorithm in section 4.2.2.

5 Experimental Setup

AHH and HHH were applied to the Toronto, ITC 2007 examination timetabling and ITC 2007 curriculum based course timetabling benchmark sets. Due to the stochastic nature of genetic programming and genetic algorithms and to ensure a normal distribution for statistical tests thirty runs were performed for each problem instance. Hypothesis tests are used to test the significance in the difference of means when comparing the performance of AHH and HHH in evolving construction heuristics.

The hyper-heuristics were implemented in Java and simulations were run on multicore clusters at the Centre for High Performance Computing (CHPC) in South Africa and University of Nottingham High Performance Computing facility.

The genetic programming parameter values used in this study are listed in table 8. These values have been determined empirically by performing trial runs. There is no limit set on the size of offspring produced by the AHH genetic operators. The chromosomes for HHH are fixed length, hence the size of the offspring remain constant.

Table 8: AHH and HHH parameter values

Number of generation	50
Population size	500
Tournament size	4
Mutation rate	50%
Crossover rate	50%
Maximum initial tree depth (for AHH)	4

6 Results and Discussion

This section discusses the performance of AHH and HHH in inducing low-level construction heuristics for the Toronto, ITC 2007 examination timetabling and the ITC 2007 curriculum based course timetabling benchmark sets. The first section examines the performance of the three different hierarchical hyper-heuristics, HHH-GP, HHH-GA and HHH-RG, on the ITC curriculum based course timetabling benchmark sets to see which approach, genetic programming, genetic algorithms or random generation, is most effective. The next section compares the performance of the best performing HHH with AHH in solving the three problems. Finally the performance of the evolved heuristics is compared to the existing graph colouring heuristics generally used to create initial solutions.

6.1 Hierarchical Hyper-Heuristics

This section firstly compares the performance of HHH-GP, HHH-GA and HHH-RG in generating construction heuristics for the curriculum based course timetabling problem. Table 9 compares the performance of the three hyper-heuristics. The table lists the minimum and average hard constraint cost and soft constraint cost (hard constraint cost/soft constraint cost) for each problem instance. The performance is compared using the minimum hard and soft constraint costs and in the case of these values being the same, the average hard and soft constraint costs. The main aim is to find a feasible solution, i.e. a hard constraint cost of zero, with the lowest soft constraint cost. Each heuristic is comprised of five problem attributes and a period selection heuristic. From table 9 it can be seen that HHH-GP and HHH-GA perform much better than HHH-RG. This was found to be statistical significant at the 1 % level of significance for all problem instances except *comp12*. HHH-GA performs slightly better than HHH-GP. For 11 of the 21 problem instances the performance of HHH-GP and HHH-GA are the same. For 8 of the problem instances HHH-GA produces better results than HHH-GP. This was found to be significant at the 1% level of significance for 4 of these problem instances. For two of the problem instances HHH-GP performs better than HHH-GA. This result was found to be significant at the 1% level of significance for one of these two problem instances.

Table 9: Comparison of HHH-GP and HHH-GA - Chromosome Length 6. Each column lists hard constraint cost / soft constraint cost.

	HHH-GP		HHH-GA		HHH-RG	
Instance	Minimum	Average	Minimum	Average	Minimum	Average
comp01	**4/63**	**4/63**	**4/63**	**4/63**	4/63	4/67.17
comp02	0/383	0/385	**0/383**	**0/383.13**	0/388	0/388.6
comp03	**0/271**	0/271	**0/271**	0/271	0/280	0/282.43
comp04	0/275	0/276.53	**0/275**	**0/276.5**	0/ 278	0/284.73
comp05	**0/974**	0/1100.23	0/1015	0.23/1096.82	0/1148	0.23/1393.2
comp06	0/344	0/349.87	**0/344**	**0/347.67**	355	0/385.13
comp07	0/412	0/415.9	**0/412**	**0/412.77**	0/419	0/446.4
comp08	**0/268**	**0/268**	**0/268**	**0/268**	0/282	0/287.93
comp09	**0/342**	**0/342**	**0/342**	**0/342**	0/342	0/354.27
comp10	0/336	0/336.77	**0/336**	**0/336.47**	0/343	0/344.47
comp11	**0/36**	**0/36**	**0/36**	**0/36**	0/40	0/41.53
comp12	**0/802**	**0/802**	**0/802**	**0/802**	0/802	0/805.57
comp13	**0/267**	**0/267**	**0/267**	**0/267**	0/273	0/285.07
comp14	**0/305**	**0/305**	**0/305**	**0/305**	0/305	0/315.33
comp15	**0/271**	**0/271**	**0/271**	**0/271**	0/280	0/285.6
comp16	**0/272**	**0/272**	0/272	0/272.37	0/283	0/288.93
comp17	**0/343**	**0/343**	**0/343**	**0/343**	0/353	0/361.63
comp18	**0/219**	**0/219**	**0/219**	**0/219**	0/219	0/220.9
comp19	0/307	0/307.23	**0/307**	**0/307**	0/316	0/319.83
comp20	0/471	0/480.43	**0/471**	**0/476**	0/490	0/514.2
comp21	0/396	0/399.5	**0/396**	**0/396.6**	0/405	0/405.1

Table 10 and 11 compares the performance of the heuristics evolved by HHH-GA comprised of 5, 10, 20 and 50 problem attributes to determine if the number of attributes included in the heuristic effects performance. Note that if a problem attribute appears more than once in a heuristic its effect at each point of application is different as the list of events at each point is different. The values of 5, 10, 20 and 50 were decided on empirically to provide a sufficient range of number of problem attributes. Table 10 displays the minimum and average hard and soft constraint costs over the 30 runs. The performance is compared in terms of the minimum hard and soft constraint cost and where this is the same the average hard and soft constraint costs are used. Table 11 lists the corresponding minimum and average fitness. From both these tables it can be seen that HHH-GA10 produces the best results for 20 of the 21 problem instances. Note that for the problem instance *comp05* while HHH-GA5 appears to produce more feasible solutions than HHH-GA10 over the 30 runs, HHH-GA10 produces feasible solutions with lower soft constraint costs and has a better fitness over the 30 runs. Hence, HHH-GA evolving heuristics comprised of 10 problem attributes will be used for HHH. The following section compares the performance of arithmetic heuristics evolved by AHH and hierarchical heuristics evolved by HHH-GA10.

Table 10: Comparison of HHH-GA with different heuristic sizes. Each column lists hard constraint cost / soft constraint cost.

	HHH-GA5		HHH-GA10		HHH-GA20		HHH-GA50	
Instance	Minimum	Average	Minimum	Average	Minimum	Average	Minimum	Average
comp01	**4/63**	**4/63.1**	**4/63**	**4/63.1**	4/64	4/64	4/64	4/64
comp02	0/383	0/383.03	**0/383**	**0/383**	0/383	0/383.03	0/383	0/383.77
comp03	**0/265**	**0/265**	**0/265**	**0/265**	**0/265**	**0/265**	**0/265**	**0/265**
comp04	0/274	0/274.1	0/274	0/274.03	**0/274**	**0/274**	0/274	0/274.13
comp05	0/995	0/1092.3	**0/981**	0.2/1069.21	0/1032	0.23/1086	0/999	0.13/1067.07
comp06	0/344	0/344.6	**0/344**	**0/344**	**0/344**	**0/344**	0/344	0/349.1
comp07	0/411	0/412.5	**0/411**	**0/411.2**	0/411	0/411.3	0/411	0/415.33
comp08	**0/268**	**0/268**	**0/268**	**0/268**	**0/268**	**0/268**	**0/268**	**0/268**
comp09	**0/342**	**0/342**	**0/342**	**0/342**	**0/342**	**0/342**	0/342	0/343.2
comp10	0/336	0/336.13	**0/336**	**0/336.07**	0/336	0/336.13	0/336	0/336.47
comp11	**0/36**	**0/36**	**0/36**	**0/36**	**0/36**	**0/36**	**0/36**	**0/36**
comp12	**0/802**	**0/802**	**0/802**	**0/802**	**0/802**	**0/802**	**0/802**	**0/802**
comp13	**0/260**	**0/260**	**0/260**	**0/260**	**0/260**	**0/260**	0/260	0/260.1
comp14	**0/305**	**0/305**	**0/305**	**0/305**	**0/305**	**0/305**	**0/305**	**0/305**
comp15	**0/265**	**0/265**	**0/265**	**0/265**	**0/265**	**0/265**	**0/265**	**0/265**
comp16	**0/272**	**0/272**	**0/272**	**0/272**	**0/272**	**0/272**	0/272	0/278.37
comp17	**0/343**	**0/343**	**0/343**	**0/343**	**0/343**	**0/343**	**0/343**	**0/343**
comp18	**0/219**	**0/219**	**0/219**	**0/219**	**0/219**	**0/219**	**0/219**	**0/219**
comp19	**0/307**	**0/307**	**0/307**	**0/307**	**0/307**	**0/307**	**0/307**	**0/307**
comp20	0/467	0/467.13	**0/467**	**0/467**	0/467	0/467.67	0/467	0/469.53
comp21	**0/396**	**0/396**	**0/396**	**0/396**	**0/396**	**0/396**	**0/396**	**0/396**

Table 11: Fitness Comparison of HHH-GA with different heuristic sizes.

	HHH-GA5		HHH-GA10		HHH-GA20		HHH-GA50	
Instance	Minimum	Average	Minimum	Average	Minimum	Average	Minimum	Average
comp01	**315**	**315**	315	315.5	320	320	320	320
comp02	383	383.13	**383**	**383**	383	383.03	383	383.77
comp03	271	271	**265**	**265**	**265**	**265**	**265**	**265**
comp04	275	276.5	274	274.03	**274**	**274**	274	274.13
comp05	1015	1362.133	**981**	1269.5	1032	1321.87	999	1207.4
comp06	344	347.67	**344**	**344**	**344**	**344**	344	349.1
comp07	412	412.77	**411**	**411.2**	411	411.3	411	415.33
comp08	**268**	**268**	**268**	**268**	**268**	**268**	**268**	**268**
comp09	**342**	**342**	**342**	**342**	**342**	**342**	342	343.2
comp10	336	336.47	**336**	**336.07**	336	336.13	336	336.47
comp11	**36**	**36**	**36**	**36**	**36**	**36**	**36**	**36**
comp12	**802**	**802**	**802**	**802**	**802**	**802**	**802**	**802**
comp13	267	267	**260**	**260**	**260**	**260**	260	260.1
comp14	3**05**	**305**	**305**	**305**	**305**	**305**	**305**	**305**
comp15	271	271	**265**	**265**	**265**	**265**	**265**	**265**
comp16	272	272.37	**272**	**272**	**272**	**272**	272	278.37
comp17	**343**	**343**	**343**	**343**	**343**	**343**	**343**	**343**
comp18	**219**	**219**	**219**	**219**	**219**	**219**	**219**	**219**
comp19	**307**	**307**	**307**	**307**	**307**	**307**	**307**	**307**
comp20	471	476	**467**	**467**	467	467.67	467	469.53
comp21	396	396.6	**396**	**396**	**396**	**396**	**396**	**396**

6.2 AHH and HHH-GA10 Performance

This section compares the performance of the arithmetic hyper-heuristic and hierarchical hyper-heuristic in creating initial solutions to educational timetabling problems. Table 12 lists the minimum and average hard constraint and soft constraint costs of the heuristics evolved using AHH and HHH-GA10 over thirty runs for the Toronto benchmark set. The average soft constraint cost is taken over those runs which produced feasible timetables. The heuristics produced by AHH produced feasible solutions on all runs for all problem instances with an exception of *car-f-92 I* for which a feasible solution was not produced on one of the runs. Just one hard constraint was violated in this timetable. HHH-GA10 produced feasible solutions on all runs for all data sets except *hec-s-92 I* and *yor-f-83 I*. For *hec-s-92 I* feasible solutions were not found on two of the runs and for *yor-f-83 I* on twenty seven of the runs. In all these cases the number of hard constraints violated was one. It is evident from table 12 that the arithmetic heuristics performed better than the hierarchical heuristics both in terms of feasibility and quality for all 13 problem instances. These results were found to be significant at the 1% level of significance.

The arithmetic heuristics evolved by AHH were arithmetic rules or arithmetic functions containing arithmetic rules. The heuristics evolved by AHH contain a large number of nodes and thus cannot be interpreted. Hence this can be considered as a blackbox approach producing instance specific construction heuristics. The heuristics produced by HHH-GA10 for the Toronto

Table 12: AHH and HHH-GA10 Performance for the Toronto benchmark set. Each column lists hard constraint cost / soft constraint cost.

	AHH		HHH-GA10	
Instance	Minimum	Average	Minimum	Average
car-f-92 I	**0/4.32**	0.03/4.51	0/9.63	0/9.63
car-f-91 I	**0/5.16**	0/5.24	0/8.67	0/11.36
ear-f-83 I	**0/36.52**	0/38.65	0/53.76	0/64.13
hec-s-92 I	**0/11.87**	0/12.4	0/15.07	0.07/16.55
kfu-s-93	**0/14.67**	0/ 15.15	0/24.45	0/32.59
lse-f-91	**0/10.81**	0/11.39	0/13.48	0/13.48
pur-s-93 I	**0/4.46**	0/4.64	0/4.83	0/4.83
rye-s-93	**0/9.48**	0/9.86	0/11.48	0/11.48
sta-f-83 I	**0/157.64**	0/158.11	0/163.7	0/164.93
tre-s-92	**0/8.48**	0/8.64	0/9.48	0/9.48
uta-s-92 I	**0/3.35**	0/3.45	0/3.59	0/3.59
ute-s-92	**0/27.16**	0/27.77	0/29.19	0/29.19
yor-f-83 I	**0/41.31**	0/43.02	0/51.19	0.9/52.58

benchmark instances are more readable as each heuristic is of length 11. Different heuristics were evolved on each run for each problem set. However, there are some similarities between the evolved heuristics for each problem instance. For example, the best heuristics induced on each run for each problem instance contained the same period or one of two period selection heuristics. Similarly, the first problem attribute in each heuristic was either the same or one of two or three on all runs for each problem instance.

The performance of AHH and HHH-GA10 for the ITC 2007 examination timetabling benchmark set is illustrated in table 13. As in table 12 the average soft constraint cost is taken over those runs producing feasible timetables. Consistent with the results for the Toronto benchmark set the arithmetic heuristics perform much better than hierarchical heuristics in constructing initial solutions to the problem. The heuristics evolved by HHH-GA10 were not able to produce feasible timetables for six of the eight problem instances. The result that AHH performs better than HHH-GA10 was found to be significant at the 1% level of significance.

Table 13: AHH and HHH-GA10 performance for the ITC 2007 examination timetabling benchmark set

	AHH		HHH-GA10	
Instance	Minimum	Average	Minimum	Average
Exam1	**0/9403**	0/9909.57	0/12647	0/12884.87
Exam2	**0/1891**	0/2249.03	0/3881	0/3947.77
Exam3	**0/13884**	0/15873.03	4/-	8.13/-
Exam4	**0/18760**	0/21783.83	23/-	27.13/-
Exam5	**0/5551**	0/6443.33	6/-	6.3/-
Exam6	**0/29695**	0/31296.5	13/-	13.57/-
Exam7	**0/10569**	0/11655.77	2/-	2.77/-
Exam8	**0/14024**	0/14263.63	2/-	2.03/-

As for the Toronto benchmark set the heuristics evolved by AHH on each run for each problem instance was different. The evolved heuristics are lengthy and hence not easily interpretable. On some runs the heuristics producing the best results are arithmetic functions for some problem instances and arithmetic rules for others, with the majority being arithmetic rules. Similarly, the best performing heuristics induced by HHH are different on each run for each problem instance. The evolved heuristics differ both in terms of the problem attributes and period selection heuristic on each run for each problem instance.

Table 14 depicts the performance of the heuristics evolved by AHH and HHH-GA10 for the ITC 2007 curriculum based course timetabling benchmark set. The heuristics evolved by both AHH and HHH-GA10 have produced feasible solutions for all problem instances except *comp01*. None of the heuristics evolved by AHH or HHH-GA10 were able to produce feasible timetables for this problem instance, the cost obtained is 4 on all thirty runs for both hyper-heuristics. The soft constraint average for this problem instance is summed over the infeasible solutions for the thirty runs. For this benchmark set the hierarchical heuristics have produced much better results than the arithmetic heuristics for all problem instances. These results were found to be significant at the 1% level of significance for all problem instances except *comp05*.

Table 14: AHH and HHH-GA10 performance for the ITC 2007 curriculum based course timetabling problem

	AHH		HHH-GA10	
Instance	Minimum	Average	Minimum	Average
comp01	4/119	4/141.3	**4/63**	4/63
comp02	0/630	0/644.4	**0/383**	0/383
comp03	0/608	0/626.13	**0/265**	0/265
comp04	0/553	0/573.83	**0/274**	0/274.03
comp05	0/1001	0/1053.8	**0/981**	0.2/1062.43
comp06	0/745	0/780.83	**0/344**	0/344
comp07	0/868	0/898.27	**0/411**	0/411.2
comp08	0/591	0/653.83	**0/268**	0/268
comp09	0/660	0/688.47	**0/342**	0/342
comp10	0/718	0/748.97	**0/336**	0/336.07
comp11	0/141	0/157.87	**0/36**	0/36
comp12	0/1279	0/1339.23	**0/802**	0/802
comp13	0/635	0/649.47	**0/260**	0/260
comp14	0/612	0/634.47	**0/305**	0/305
comp15	0/580	0/621.83	**0/265**	0/265
comp16	0/763	0/800.73	**0/272**	0/272
comp17	0/766	0/787.07	**0/343**	0/343
comp18	0/485	0/504.53	**0/219**	0/219
comp19	0/565	0/583.53	**0/307**	0/307
comp20	0/841	0/868.1	**0/467**	0/467
comp21	0/772	0/800.73	**0/396**	0/396

As in the case of the previous two benchmark sets AHH produced different heuristics on each run for all problem instances. The evolved heuristics are lengthy and hence not easily readable. Hence the AHH operates as blackbox.

The heuristics producing the best result over the thirty runs for each problem instance is an arithmetic rule. As the size of the hierachical heuristics evolved are limited, these heuristics are more readable. Different heuristics are evolved on each of the thirty runs for all problem instances. The best performing heuristic for each run for all problem instances with an exception of *comp05* included the minimum penalty period selection heuristic. For all thirty runs for *comp05* the period selection heuristic contained in the best performing evolved heuristic was random period. The first problem attribute in the best performing evolved heuristics was the same or one of two problem attributes for the thirty runs for all problem instances. Majority of the evolved heuristics across all problem domains included the saturation degree or room degree as the first problem attribute.

For both AHH and HHH-GA10 the same parameter values were used for the population size and number of generations with the termination criterion being the number of generations. Hence, the same number of fitness evaluations were performed for all problem instances. However, in terms of runtime AHH took longer than HHH-GA10 with runtime ranging from 1 minute to 19 hours for the former and 30 seconds to 10 hours for the latter depending on the problem instance being solved. In all cases it takes less than a day for a problem specific heuristic to be evolved.

6.3 Comparison to Existing Construction Heuristics

This section compares the performance of the heuristics evolved to the existing low-level construction heuristics described in section 3.4 used to create initial solutions for educational timetabling. Algorithms 1 and 2 presented in section 3.4 are used to create the timetables. Table 15 lists the hard constraint and soft constraint values for the existing low-level heuristics and evolved heuristics for the Toronto benchmark set. The heuristics induced by both AHH and HHH-GA10 have produced initial solutions with better feasibility and quality than the existing low-level construction heuristics with the exception of *lse-f-91* for which SD and LCD have a better soft constraint cost than HHH-GA10. From table 15 it can be seen that the heuristics produced by AHH have produced the best initial solutions for all 13 problem instances.

The performance of the heuristics evolved by AHH and HHH-GA10 is compared to that of the existing low-level heuristics for the ITC 2007 examination timetabling benchmark set in table 16. The heuristics evolved by AHH has produced the best initial solutions with respect to both feasibility and quality. For some problem instances the low-level heuristics produce better solutions than the heuristics evolved by HHH-GA10. Upon examination there does not appear to be any evident correlation between the characteristics of the problems instances as outlined in table 2 and the performance of HHH-GA10 and the low-level heuristics.

Table 17 compares the existing heuristics and evolved heuristics for the ITC 2007 curriculum based course timetabling problem. The largest weighted

Table 15: Performance comparison with low-level heuristics for the Toronto benchmark set. Each column lists hard constraint cost / soft constraint cost.

Instance	LD	LE	LWD	LCD	SD	AHH	HHH-GA10
car-f-92 I	6/4.89	10/4.62	12/4.63	6/4.97	5/4.97	**0/4.32**	0/9.63
car-f-91 I	13/5.89	18/5.09	14/5.58	6/ 5.58	2/0.66	**0/5.16**	0/8.67
ear-f-83 I	2/40.2	13/41.15	6/38	2/46.47	2/46.77	**0/36.52**	0/53.76
hec-s-92 I	2/14.22	7/12.92	6/11.83	2/13.4	2/13.38	**0/11.87**	0/15.07
kfu-s-93	4/18.11	4/16.27	6 /17.55	1/18.12	5/17.34	**0/14.67**	0/24.45
lse-f-91	4/14.05	8/13	3/12.86	0/12.6	0/12.53	**0/10.81**	0/13.48
pur-s-93 I	1/4.94	5/4.9	4/4.91	0/4.95	0/4.93	**0/4.46**	0/4.83
rye-s-93	2/12.76	6/10.83	6/ 4.96	0/11.92	0/12.05	**0/9.48**	0/11.48
sta-f-83 I	24/163.38	1/172.04	2/172.02	0/178.24	0/178.24	**0/157.64**	0/163.7
tre-s-92	6/10.39	7 /9.54	5/9.03	2 /0.37	2/10.42	**0/8.48**	0/9.48
uta-s-92 I	8/3.84	13/3.73	11 /3.62	2/10.41	2/3.98	**0/3.35**	0/3.59
ute-s-92	2/34.72	3/28.86	3/29.58	2/3.88	4/46.5	**0/27.16**	0/29.19
yor-f-83 I	6/44.62	10/40.55	17/41.03	5/45.34	1/32.82	**0/41.31**	0/51.19

Table 16: Performance comparison with low-level heuristics for the ITC 2007 examination timetabling benchmark set. Each column lists hard constraint cost / soft constraint cost.

Instance	LD	LWD	LE	LCD	SD	AHH	HHH-GA10
Exam1	0 /12870	0/11519	0 /12360	0/12609	0/12600	**0/9403**	0/12647
Exam2	0 /7492	7/ 18343	1/5668	0/4112	0/4599	**0/1891**	0/3881
Exam3	8/18397	7/18343	9/22153	0 /16568	8 /18508	**0/13884**	8/24303
Exam4	2/26457	50/38241	49/31238	4/25418	2/30683	**0/18760**	23/35264
Exam5	9/7189	10/13927	9 /13128	0/8548	9/6801	**0/5551**	6/8293
Exam6	4/33545	6/33800	11/36855	0/35800	8/32930	**0/29695**	13/32150
Exam7	4/19048	5/18241	5/19604	0/17312	1/23561	**0/10569**	2/15202
Exam8	1/20084	0/15702	0/16302	0 /15715	0/15336	**0/14024**	2/16808

degree is not included as the number of students that courses have in common is not provided as part of the benchmark set. The heuristics evolved by HHH-GA10 have generally outperformed the existing heuristics and those produced by AHH. For some of the problem instances the existing low-level heuristics have performed better than the heuristics evolved by AHH, producing feasible timetables with a lower soft constraint cost for some of the problem instances, e.g. *comp02*. As in the case of the ITC 2007 examination timetabling benchmark set there is no evident correlation between the problem characteristics listed in table 3 and the performance of the low-level heuristics and AHH.

The performance of the evolved heuristics is also compared to that of other studies employing genetic programming hyper-heuristics to automatically induce construction heuristics for educational timetabling. There is not much work done in this domain and previous work in this area has been evaluated using the Toronto benchmark set. There has been no previous research into evolving heuristics using genetic programming hyper-heuristics for both the ITC 2007 benchmark sets. Table 18 compares the performance of both hyper-heuristics with the grammatical evolution hyper-heuristic employed in [1] and the genetic programming hyper-heuristic implemented in [14]. These studies

Table 17: Performance comparison with low-level heuristics for the ITC 2007 curriculum based course timetabling problem

Instance	LD	LE	LCD	SD	AHH	HHH-GA10
comp01	6/86	4/196	8/57	8/59	4/119	**4/63**
comp02	9/455	4/843	2/360	0/437	0/630	**0/383**
comp03	4/390	0/741	2/34	0/328	0/608	**0/265**
comp04	1/382	0/702	7/292	0/320	0/553	**0/274**
comp05	9/589	3/1579	6/571	4/656	0/1001	**0/981**
comp06	4/463	2/969	9/399	4/356	0/745	**0/344**
comp07	7/543	1/1098	3/459	1/459	0/868	**0/411**
comp08	0/333	0/757	1/331	0/346	0/591	**0/268**
comp09	0/413	0/788	0/328	0/383	0/660	**0/342**
comp10	0/403	0/969	3/453	0/404	0/718	**0/336**
comp11	7/94	0/221	0/67	1/34	0/141	**0/36**
comp12	4/830	6/1701	6/805	0/839	0/1279	**0/802**
comp13	1/348	0/806	5/341	0/328	0/635	**0/260**
comp14	5/391	1/808	8/302	0/324	0/612	**0/305**
comp15	4/390	2/805	2/345	0/328	0/580	**0/265**
comp16	5/433	0/931	3/367	0/338	0/763	**0/272**
comp17	8/516	1/880	4/420	0/371	0/766	**0/343**
comp18	0/246	0/690	0/243	0/256	0/485	**0/219**
comp19	2/352	2/715	0/333	3/363	0/565	**0/307**
comp20	12/555	2/1048	4/495	13/451	0/841	**0/467**
comp21	4/482	2/1013	2/472	0/411	0/772	**0/396**

are described in section 1. From table 18 it is evident that the heuristics evolved by AHH outperform the other evolved heuristics for the benchmark set.

Table 18: Performance comparison with previous work for the Toronto benchmark set. Each column lists hard constraint cost / soft constraint cost.

Instance	[1]	[14]	AHH	HHH-GA10
car-f-92 I	0/4.46	-	**0/4.32**	0/9.63
car-f-91 I	0/textbf5.12	-	0/5.16	0/8.67
ear-f-83 I	0/37.10	0/37.39	**0/36.52**	0/53.76
hec-s-92 I	0/11.78	**0/11.43**	0/11.87	0/15.07
kfu-s-93	0/14.72	-	**0/14.67**	0/24.45
lse-f-91	0/11.11	-	**0/10.81**	0/13.48
pur-s-93 I	-	-	**0/4.46**	0/4.83
rye-s-93	-	-	**0/9.48**	0/11.48
sta-f-83 I	0/158.70	0/158.38	**0/157.64**	0/163.7
tre-s-92	0/8.62	-	**0/8.48**	0/9.48
uta-s-92 I	0/3.47	-	**0/3.35**	0/3.59
ute-s-92	-	0/27.31	**0/27.16**	0/29.19
yor-f-83 I	**0/40.56**	0/ 39.96	0/41.31	0/51.19

7 Conclusion

The research presented in this paper investigates the use of generation construction hyper-heuristics to automate the process of low-level construction

heuristic generation for educational timetabling. Two hyper-heuristics, AHH for evolving arithmetic heuristics and HHH-GA10 to generate hierarchical heuristics have been implemented and applied to solving two examination timetabling problems and a curriculum based course timetabling problem. AHH was found to be more effective for the examination timetabling problem, outperforming both the heuristics induced by HHH-GA10 and the existing low-level construction heuristics for this domain when applied to the Toronto and ITC 2007 benchmark sets. However, AHH did not perform as well for the curriculum based course timetabling problem and HHH-GA10 produced the best results for this problem also outperforming the existing heuristics for the ITC 2007 curriculum based course timetabling benchmark set. Hence, the study has revealed that different types of heuristics are more effective for different educational timetabling problems.

Given this future work will investigate a hyper-heuristic that can cater for more than one type of heuristic representation and also evaluating the hyper-heuristic on the post enrolment course based and high school timetabling problems. This study focussed on evolving disposable heuristics based on previous work on genetic programming construction heuristics for examination timetabling. Future work will also investigate the possibility of inducing reusable heuristics for educational timetabling.

Acknowledgements The authors would like to thank the reviewers for their helpful comments to improve the quality of the paper. The facilities made available by the Centre for High Performance Computing (CHPC) in South Africa and University of Nottingham High Performance Computing facility to run simulations for the experiments is acknowledged.

References

1. Bader-El-Den, M., Poli, R., Fatima, S.: Evolving timetabling heuristics using grammar-based genetic programming hyper-heuristic framework. Memetic Computing **1**, 205–219 (2009)
2. Beasley, J.: Or-library. URL http://people.brunel.ac.uk/ mastjjb/jeb/orlib/tableinfo.html
3. Branke, J., Nguyean, S., Pickardt, C.W., Zhang, M.: Automated design of production scheduling heuristics: A review. IEEE Transactions on Evolutionary Computation **20**(1), 110–124 (2015)
4. Burke, E.K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Ozcan, E.: Hyper-heuristics: A survey of the state of the art. Journal of Operational Research Society **64**, 1695–1724 (2013)
5. Burke, E.K., Hyde, M., Kendall, G., Woodward, J.: A genetic programming hyper-heuristic approach for evolving two dimensional strip packing heuristics. IEEE Transactions on Evolutionary Computation pp. 942–958 (2010)
6. Burke, E.K., McCollum, B., Meisels, A., Petrovic, S., Qu, R.: A graph-based hyper-heuristic for educational timetabling problems. European Journal of Operational Research **176**, 177–192 (2007)
7. Drake, J.H., Hyde, M., Ibrahim, K., Özcan, E.: A genetic programming hyper-heuristic for the multidimensional knapsack problem. Kybernetes **43**(9/10), 1500–1511 (2014)
8. Hyde, M.: A genetic programming hyper-heuristic approach to automated packing. Ph.D. thesis, School of Computer Science (2010)
9. Koza, J.: Genetic Programming: On the Programming of Computers by Means of Natural Selection, 1st edn. MIT (1992)

10. McCollum, B., McMullan, P., Paechter, B., Lewis, R., Schaerf, A., DiGaspero, L., Parkes, A., Qu, R., Burke, E.: Setting the research agenda in automated timetabling: The second international timetabling competition. INFORMS Journal of Computing **22**(1), 120–130 (2008)
11. McKay, R.I., Hoai, N.X., Whigham, P., O'Neill, M.: Grammar-based genetic programming: A survey. Genetic Programming and Evolvable Machines **11**(3), 365–396 (2010)
12. O'Neill, M., Ryan, C.: Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language. Springer (2003)
13. Özcan, E., Parkes, A.: Policy matrix evolution for generation of heuristics. In: Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation, pp. 2011–2018 (2011)
14. Pillay, N.: Evolving hyper-heuristics for the uncapacitated examination timetabling problem. In: Proceedings of the Multidisciplinary International Conference on Scheduling, pp. 409–422 (2009)
15. Pillay, N.: Evolving heuristics for the school timetabling problem. In: Proceedings of the 2011 IEEE Conference on Intelligent Computing and Intelligent Systems (ICIS 2011), vol. 3, pp. 281–286. IEEE (2011)
16. Qu, R., Burke, E., McCollum, B., Merlot, L., Lee, S.: A survey of search methodologies and automated system development for examination timetabling. Journal of Scheduling **12**(1), 55–89 (2009)
17. Sim, K., Hart, E.: A combined generative and selective hyper-heuristic for the vehicle routing problem. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '16), pp. 1093–1100. ACM (2016)