Grammar-Based Genetic Programming for Timetabling

Mohamed Bader El Den and Riccardo Poli

***Abstract*— We present a grammar-based genetic programming framework for the solving the timetabling problem via the evolution of constructive heuristics. The grammar used for producing new generations is based on graph colouring heuristics that have previously proved to be effective in constructing timetables as well as different slot allocation heuristics. The framework is tested on a widely used benchmarks in the field of exam time-tabling and compared with highly-tuned state-of-the-art approaches. Results shows that the framework is very competitive with other constructive techniques.**

I. Introduction

Heuristics are a great asset for practical problem solving. However, the performance of heuristics may vary significantly from problem instance to problem instance. Furthermore, there is no easy way to ascertain which heuristic is going to be the most efficient in solving a given set of problems, or a certain problem instance. For some problems a heuristic method may work well for most instances, but occasionally performance is poor and other heuristics should be used. This degree of uncertainty and unreliability in the use of heuristics is really the main motivation behind *Hyper-Heuristics* (HHs). HHs could simply be defined as "heuristics to choose heuristics" [1]. More specifically, a HH manages the choice of which lower-level heuristic method should be applied at any given time, depending upon the characteristics of the heuristics and the region of the solution space currently under exploration.

There are different classes of HHs. In one class of HHs, the system is provided with a list of preexisting heuristics for solving a certain problem. Then the HH tries to discover what is the best sequence of application for these heuristics for the purpose of finding a solution. Different techniques have been used to build HH systems of this class, including: tabu search [2], case-based reasoning [3], genetic algorithms [4], ant-colony systems, and even algorithms inspired to marriage in honey-bees [5].

Another form of HH is one where the system produces (meta-)heuristics by specialising them from a generic template. The specialisation can take the form of one or more evolved components, which can modify the behaviour of the meta-heuristic or heuristic. This approach has given, for example, good results in [6] where the problem of evolving offline bin-packing heuristics was considered. There Genetic Programming (GP) (more on it below) was used to evolve strategies to guide a fixed solver. This approach was also taken in [7] where a technique called Inc* was proposed. This solver was applied to the satisfiability testing problem (known as SAT) with good success. The target in SAT is to determine whether it is possible to set the variables of a given Boolean expression in such a way to make the expression true. To further improve chances of success, a key element of Inc*, its strategy for adding and removing SAT clauses, was evolved using GP. A third approach used to build HH systems is to create (e.g., evolve) new heuristics by making use of the *components* of known heuristics. This is the approach adopted in this paper.

In the paper we present a grammar-based Genetic Programming (GP) framework for evolving constructive heuristics for timetabling, focusing in particular on exam timetabling, an important problem in higher education. The system is based on a grammar that is based on a collection of graph colouring heuristics that have previously been shown to be effective in constructing timetables. Also, the grammar contains slot allocation heuristics as we will describe in details below. The main advantage in our GP HH method over other HH methods is that GP allows us to make use of conditional branching, looping and other components. So, not only it can find different combination of heuristics that performs well on a certain problem, it can also discover new kinds of heuristics. One of the targets in this study is to compare GP as methods for building HH frameworks against other metaheuristics techniques. Exam timetabling works well as a reference problem because there are a number HH frameworks developed in the last few years to solve it.

The paper is organised as follows. In Section II we introduce the exam timetabling problem, then, in Section III, we review some HH methods and some of the best-known techniques for solving the problem. In Section IV, we introduce our GP-HH for evolving constructive timetable heuristics. A description of our GP setup is given in Section V, while results obtained, comparisons, and analysis of the framework are provided in Section VI. Finally, we draw some conclusions in Section VII.

II. The Exam Timetabling Problem

The exam timetabling problem is a common problem in most educational institutions. Although the problem's details tend to vary from one institution to another, the core of the problem is the same. There is a set of exams (tasks), which have to be assigned to a predefined set of slots and rooms (resources).

In our research we will be using on the following formulation for the exam timetabling problem. The problem consists of the a set of n exams $E = \{e_1, \dots e_n\}$, a set of m students

Mohamed Bahy Bader-El-Den with the Department of Computer Science, Loughborough University, Loughborough, UK, and the Department of Computer Science and Electronic Engineering, University of Essex, Wivenhoe Park, Colchester, UK (email: mbbade@essex.ac.uk).

Riccardo Poli are with the Department of Computer Science and Electronic Engineering, University of Essex, Wivenhoe Park, Colchester, UK (email: rpoli@essex.ac.uk).

$S = \{s_1, \dots s_m\}$, a set of q time slots $P = \{p_1, p_2, \dots p_q\}$ and a registration function $R : S \rightarrow E$, indicating which student is attending which exam. Seen as a set $R = \{(s_i, e_j) : 1 \leq i \geq m, 1 \leq j \geq n\}$, where student s_i is attending exam e_j. A scheduling algorithm assigns each exam to a certain slot. A solution then has the form $O : E \rightarrow P$ or, as a set, $O = \{(e_k, p_l) : 1 \leq k \geq n, 1 \leq l \geq q\}$.

The problem is similar to the graph colouring problem but it includes extra constraints, as shown by Welsh and Powell [8]. These constraints are categorised into two main types: (a) *Hard Constraints*, violating any of these constraints is not permitted since it would lead to an unfeasible solution, and (b) *Soft Constraints*, which are desirable but not crucial requirements. Violating any of the soft constraints will only affect the solution's quality. All hard constraints are equally important, while soft constraints are not. The importance of soft constraints vary. Usually a cost function is designed to calculate the cost of violating each of the soft constraints. Solutions with lower cost have better quality.

Most constraints relate to the main entities of the problem: students, exams, rooms and slots. Examples include controlling the maximum number of exams or students in each slot, limiting some exams to rooms with special resources, etc. There is no clear cut distinction between soft and hard constraints: selection between soft and hard constraints depends on each institution's requirements. A constraint that is soft for one institution could be considered hard by another. For example, the constraint that students cannot attend two exams in the same slot is hard for most institutions. However, some other institutions may accept violations of this if only few students are effected, because a special exam session can be set up for those students. Fortunately, in most cases, changing soft constraints combinations and their importance (weight) does not require changes in the timetabling algorithm itself. Only changing the cost function which evaluates the quality of each solution is required.

There are two main types of heuristics in timetabling: constructive heuristics and improvement heuristics. Constructive heuristics are used to construct solutions from scratch, while improvement heuristics are applied to previously constructed solutions to attempt to improve their quality. In this paper we will be focusing on constructive heuristics.

III. Related Work

Over the years, a number of approaches have been investigated for exam timetabling. These include graph colouring techniques, constraint programming, and different metaheuristic approaches, e.g., genetic algorithms, simulated annealing, tabu search, and other HHs. Below we will describe some of the HHs that have been developed for time tabling.

Fuzzy logic HHs were used in [9] to estimate the difficulty of each exam. The algorithm starts by ordering the hardest exams first. Each exam is given a weight indicating the exam difficulty. The ordering of exams is done on the basis of the following three criteria: largest degree, saturation degree and largest enrolment. In case of a deadlock, rescheduling is performed till all exams are scheduled.

A tabu search HH for timetabling was used in [10], [11], [12]. The main idea behind the Tabu Search technique is to avoid repeating the evaluation of recently visited solutions. This is done by having a memory structure that stores the k most recently evaluated solutions. In [12] the authors employed Tabu Search as a high-level metaheuristic to search through a space of heuristics for university course timetabling and nurse rostering problems. The heuristics used in the study were improving heuristics that work on previously constructed solutions. In [10] the low-level heuristics were constructive heuristics (graph colouring heuristics). A Tabu Search approach was employed to search for permutations of graph heuristics which are used for constructing timetables in exam and course timetabling problems. This underpins a multi-stage HH where the Tabu Search employs permutations upon a different number of graph heuristics in two stages. The representation of the solutions in these approaches (a string of heuristics) is similar to the representation used in [13] where genetic algorithms were used to find combinations of heuristics that perform well on certain problems. Each individual was represented as a set of strings of the same length as the number of exams in the problem instance. Each entry in the first string represented a code indicating which exam selection heuristic to use. The main difference between the two approaches is how they move in the search space: in the Tabu-based methods some local search heuristics were used beside random moves, while in the GA algorithm crossover and mutation were used.

Case Based Reasoning was used in [3] for constructing timetabling solutions. Heuristics that worked well in previous similar situations were memorised in a case base and were retrieved for solving the problem at hand. Knowledge discovery techniques were employed in two distinct scenarios. In the first, they were used to model the problem and for solving situations along with specific heuristics for the targeted problems. In the second, they were used to refine the case base and discard cases which proved to be useless.

IV. GP-HH for Time Tabling

Genetic Programming (GP) [14], [15], [16] is a program induction algorithm which is inspired by biological evolution. The target of a GP system is to find computer programs that perform a user-defined task. It is a specialisation of genetic algorithms where each individual is a computer program. GP optimises a population of computer programs based on a fitness function that measures each program's performance on a given task. GP uses genetic operators (e.g., crossover, mutation and reproduction) to transform the current populations of programs into new, hopefully better, populations. In section we introduce our grammar-based GP HH framework for evolving exam timetabling constructive heuristics.

The system makes use of a grammar. We developed a grammar that is able to draw different kinds of timetabling heuristics together. The grammar contains exam selection components, slot selection components, conditional branching and other components as we will describe later. GP uses the grammar while initialising the population, as well as during crossover and mutation.

The process of constructing the grammar starts simply by selecting a suitable set of heuristics that are known to be useful in solving a certain problem. Then, instead of directly feeding these heuristics to the HH system (as in the first type of HHs discussed in Section I), where possible the heuristics are first decomposed into their basic components. Different heuristics may share different basic components in their structure. However, during the decomposition process, information on how these components were connected with one another is lost. To avoid throwing away this important information, the mutual relations between components are captured by our grammar. Both the grammar and the heuristics components are given to the GP HH systems.

In [17] this approach was used to evolve heuristics for the SAT problem which are specialised to solve specific sets of instances of the problem. We should stress that here we use GP as an online learning method which evolves heuristics while solving the problem. The system keeps on evolving heuristics for each problem instance and evaluating them until a sufficiently good solution is found or some other stopping condition is met. This is different from the approach used in [17] for evolving SAT heuristics, where GP was used as an offline learning method. More specifically, in [17] GP was first applied to a training set of instances, then the best evolved heuristics were used to directly to solve other instances without further evolution. In [18] a simple Linear GP have been used in for evolving sequential heuristics for the travelling salesman problem.

A. Exam selection heuristics

Constructive graph colouring heuristics have been successfully used in constructing timetables [19]. These heuristic have been used by many HHs [9], [10] and other frameworks for constructing timetables. The exam selection heuristics used here are those proposed in [19], namely:

- Largest Degree Based Selection (LD)
- Largest Enrolment Based Selection (LE)
- Least Saturation Based Selection (SD)
- Largest Weighted Based Selection (LWD)
- Random Selection (RS)

Two exams are considered to be in conflict with each other if they cannot be scheduled in the same slot. In this study we consider two exams conflicting if there are one or more students registered in both exams. In the largest degree heuristics, LD, exams with the most conflicts are selected first. The largest enrolment, LE, heuristic selects first the exams with the largest numbers of students. The saturation degree, SD, heuristic selects exams with the least number of available slots first. This is the only heuristic that entails updating. That is, assigning an exam to a slot will make this slot unavailable for all other slots in conflict with this exam and the count of available slots for them will be decreased by one. Largest Weighted Degree, LWD, is the same as the LD heuristic but if there are more than one exam with the same number of conflicts, the tie is broken in favour of the exam with more students. Random Selection, RS, is the simplest heuristic in this group: it selects a random exam from the list of unscheduled exams. This heuristic brings some randomness into the scheduling process, which, in some cases, helps produce better results. On the other hand, most probably randomness will cause the system not to produce the same results when running the same group of heuristics more than once. Authors overcome this problem by simply running the same heuristics combination for more than one time to ensure robustness.

```
<exm>     ::=  random <eList> |
               first <eList>
<eList>   ::=  max-conflict <eList> |
               least-slot <eList> |
               max-students <eList> |
               all-exams |
```

Fig. 1: The set of heuristics that is responsible for selecting an exam in the GP-HH grammar.

Figure 1 shows the implementation of the exam selection heuristics in the GP-HH. The LD, LE and SD heuristics are represented in the grammar as `max-conflict`, `max-student` and `least-slot`, respectively. The grammar is designed in such a way that different combinations of heuristics could be nested together. The LWD heuristic has no direct representation in the grammar as it can simply be obtained by nesting the `max-conflict` and `max-student` heuristics together as follows:

```
random
      max-student
            max-conflict
                     all-exams
```

where line breaks and spaces were introduced to increase readability. This statement consists of a nested call of heuristics. The last part of the statement, `all-exams`, returns to `max-conflict` a list of all exams not scheduled, yet. If there are more than more than one exam with the highest number of conflicts, the tie is resolved by `max-student` in favour of higher number of students. If there is still a tie, this is resolved randomly (by `random`).

The ability of recursively cascading components present in our grammar gives GP the flexibility to evolve more sophisticated heuristics such as the LWD exemplified above.

B. Slot selection heuristics

The slot selection heuristics assign one of the available slots to a previously selected exam. We use the following heuristics for that:

- Least Cost Based Selection (LC).
- Least Blocking Based Selection (LB).
- Busiest Based Selection (BU).
- Least Busy Based Selection (LU).
- Random Selection (RS).

```
<slt>     ::=  random <sList> |
               first <sList>
<sList>   ::=  least-cost <sList> |
               least-busy <sList> |
               most-busy <sList> |
               least-blocking <sList> |
               all-slots
```

Fig. 2: Part of the GP-HH grammar which is responsible for selecting exams.

The LC heuristic selects the slot which causes the least increase in the solution's cost. The LB heuristic selects the slot which causes the least decrease in total number of available slots for all other unscheduled conflicting exams. The idea behind the BU heuristic is to select the busiest available slot with other exams to keep as much space as possible for other exams. This could specifically be effective in breaking ties between slots that have the same cost. The LU heuristics is the opposite of BU. It select the least occupied slot with other exams. The Random heuristic simply selects a random slot.

The part of the grammar representing these heuristics is shown in Figure 2. The grammar also allows these heuristics to be nested as described in the previous section with the exam selection heuristics.

C. GP-HH Full Grammar

The full GP-HH grammar developed for exam timetabling is shown in Figure 3. Beside what we have described before, the grammar contains conditional branching.

There are two types of condition. The first is probabilistic branching, based on some fixed probability. The second is based on how far the evolved heuristic is in constructing the timetable. This is because the performance of heuristics varies throughout the construction of a solution. Some may be efficient in the early stages, while others are more effectively at refining timetables. For example, the SD exam selection heuristic, which selects the exam with the least number of available slots, may not be efficient in the beginning of the timetable construction, when most or all of the slot are still empty and available for almost all exams. In this stage, a heuristic such the LD exam selection heuristic may be more effective because it selects exams based on the number of conflicts with other exams. On the other hand, the use of SD may be more effective at the late stages where the slots are blocked with other exams, and there are some exams with very slots few available. So, it is more reasonable to schedule these exams first. Using a form of branching which considers how far the evolved heuristic are in constructing the timetable allows GP to decide when and how to use these heuristics. More specifically, `vSmall` returns `true` if less than 25% of the exams are scheduled, `small` from 25% to 50%, `mid` 50% to 75% and `large` when more than 75% of the exams are scheduled.

```
S         ::=  assign <exm> <slt>

<exm>     ::=  random <eList> |
               first <eList>

<eList>   ::=  max-conflict <eList> |
               least-slot <eList> |
               max-students <eList> |
               all-exams |
               if <cond><eList><eList>

<slt>     ::=  random <sList> |
               first <sList>

<sList>   ::=  least-cost <sList> |
               least-busy <sList> |
               most-busy <sList> |
               least-blocking <sList> |
               all-slots
               if <cond><sList><sList>

<cond>    ::=  <pro> | <size>

<size>    ::=  vSmall | small |
               mid | large

<prob>    ::=  20 | 40 | 50 |
               70 | 90
```

Fig. 3: The complete GP-HH grammar.

D. Constraints

We adopted the most common hard and soft constraints in the literature to be able to compare our results with the largest number of available results in the literature. The hard constraints we considered in this paper represent the "conflicts" of scheduling two exams with common students into the same time slot. The soft constraints are concerned with spreading out each student's exams over the timetable so that students will not have to sit exams that are too close to each other. The objective is to schedule all of the exams into the time slots, while minimising the cost on the violations of the soft constraint per student.

The cost is calculated using the following function which was first presented in [19]:

$$C(t) = \frac{1}{S} \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} [w(|p_i - p_j|)a_{ij}] \qquad (1)$$

where N is the total number of exams in the problem, S the total number of student, $a(i, j)$ returns the number of students attending both exams i and j, p_i is the time slot where exam i is scheduled, $w(|p_i - p_j|)$ returns $2^{5-|p_i-p_j|}$

if $|p_i - p_j| \leq 5$, and 0 otherwise.[1]

V. Genetic Programming Setup

The GP system initialises the population by randomly drawing nodes from the function and terminal sets. This is done uniformly at random using the GROW method, except that the selection of the function (head) $assign$ is forced for the root node and is not allowed elsewhere. After initialisation, the population is manipulated by the following operators:

- Tournament selection, with tournament size of 5. Reselection is permitted.
- The reproduction rate is 0.1. Individuals that have not been affected by any genetic operator are not evaluated again to reduce the computation cost.
- The crossover rate is 0.8. Offspring are created using a specialised form of crossover. A random crossover point is selected in the first parent, then the grammar is used to select the crossover point from the second parent. It is randomly selected from all valid crossover points. If no point is available, the process is repeated again from the beginning until crossover is successful.
- Mutation is applied with a rate of 0.1. This is done by selecting a random node from the parent (including the root of the tree), deleting the sub-tree rooted there, and then regenerating it randomly as in the initialisation phase.
- Population size ranges between 500 and 1000 individuals.
- Number of generation ranges between 50 and 100.
- We run each individual 3 times. Best performing individuals are run for an extra 2 times.

The fitness function we used is the following:

$$f = \Big[\frac{1}{S}\sum_{i=1}^{M-1}\sum_{j=i+1}^{M} w(|p_i - p_j|)a_{ij}\Big] + C(N - M) \qquad (2)$$

where: N is the total number of exams in the problem, M is the total number of exams that have been successfully scheduled, $(N - M) \geq 0$ is the number of unscheduled exams, C is constant. The objective is to minimise this equation, so the lower the fitness value the better the individual.

The first part of Equation (2) is almost the same as the cost function in Equation (1). The second part adds extra penalty for each exam the heuristic (individual) has not been able to schedule. Even though solutions with unscheduled exams are considered to be invalid solutions, this extra penalty for unscheduled exams is introduced to give GP better ability to differentiate between individuals.

[1]This means that in the most undesirable situation, i.e., when a student has two exams scheduled one after the other, i will increase the cost function by a large value, namely $2^{5-1} = 16$. This factor rapidly decreases (following a negative exponential profile) as the size of the gap between exams increases.

TABLE I: Characteristics of benchmark exam timetabling problems we used. For each case we show: total number of exams, number of students registered in at least one exam, maximum number of slots available, maximum number of students registered in one exams, maximum number of exams registered by one student and matrix density.

Name	Exams	Std.	Slots	Max. S. Reg.	Max. E. Reg.	Matrix Density
car91	682	16925	35	1385	9	0.128
car92	543	18419	32	1566	7	0.138
ear83	190	1125	24	232	10	0.267
hec92	81	2823	18	634	7	0.421
kfu93	461	5349	20	1280	8	0.055
lse91	381	2726	18	382	8	0.063
sta83	139	611	13	237	11	0.144
tre92	261	4360	23	407	6	0.181
uta93	184	21266	35	1314	7	0.126
york83	181	941	21	175	14	0.289

VI. Results

We tested our GP HH method for timetabling by applying it to one of the most widely used benchmarks in exam timetabling, against which many state-of-the-art algorithms have been compared in the past. The benchmark was first presented in [19]. Its characteristics are shown in Table I. The size of the problems varies from 81 to 682 exams and from 611 to 18419 students. In Table I Max. S. Reg. is the maximum number of students registered in one exam; Max. E. Reg. is maximum number of exams registered by one student; matrix density is the density of the conflict matrix, which is given by the ratio of the number of conflicting exams over the total number of all possible pair exam combinations.

Table II shows the performance of the GP-HH approach against other state-of-the-art algorithms. The table shows the cost of each solution for each problem instance in the benchmark. The cost is calculated using Equation (1). The table is divided into two parts. The upper part contains the algorithms that use construction heuristics only with or without backtracking (the first two methods in this group use constructions heuristics only; the other use construction with backtracking). GP-HH outperforms all other algorithms in this group on almost 50% of the instances. The bottom part of Table II shows a group of algorithms that use constructive heuristics followed by improvement heuristics. Clearly, it is not fair to compare the GP-HH algorithm with this group because GP-HH uses constructive methods only. However, we show the comparison here so that readers can see how close the GP-HH can get to the performance of improvement-based algorithms. GP HH has been able to outperform some of the algorithms in this group, and, amazingly, in some cases it got results that are very close to the best known results, as in the `car92` instance, where the cost achieved by the GP-HH, 4.15, is second best.

To provide an analysis of the performance the GP-HH, we collected data from experiments with different numbers of generations and different numbers of individuals so as to

TABLE II: Results from the GP-HH for time tabling among with the best results reported in literature on benchmark exam timetabling problems

		car91	car92	ear83	hec92	kfu93	lse91	sta83	tre92	uta93	york83
Constructive	GP-HH	5.19	4.15	37.24	11.96	14.83	11.17	158.63	8.69	3.43	40.05
	Burke et al [10]	5.41	4.84	38.19	12.72	15.76	13.15	141.08	8.85	3.88	40.13
	Asmuni et al [9]	5.20	4.52	37.02	11.78	15.81	12.09	160.42	8.67	3.57	40.66
	Carter et al [19]	7.10	6.20	36.40	10.80	14.00	10.50	161.50	9.60	3.50	41.70
	Multi-stage [10]	5.41	4.84	38.84	13.11	15.99	13.43	142.19	9.2	4.04	44.51
Improvement Methods	Abdullah et al [20]	5.21	4.36	34.87	10.28	13.46	10.24	150.28	8.13	3.63	36.110
	Burke &Newall 2002 [21]	4.60	4.00	37.05	11.54	13.90	10.82	168.73	8.35	3.20	36.80
	Burke,Bykov et al [22]	4.20	4.80	35.40	10.80	13.70	10.40	159.10	8.30	3.40	36.70
	Caramia et al [23]	6.60	6.00	29.30	09.20	13.80	09.60	158.20	9.40	3.50	36.20
	Casey & Thompson [24]	5.40	4.40	34.80	10.80	14.10	14.70	134.70	8.70	xx	37.50
	Di Gapero & Schaerf [11]	6.20	5.20	45.70	12.40	18.00	15.50	160.80	10.0	4.20	41.00
	Merlot et al [25]	5.10	4.30	35.10	10.60	13.50	10.50	157.30	8.40	3.50	37.40

give a broader view on the performance of the GP-HH. The number of generations ranged from 50 to 100 generations, while the number of individuals was 500 and 1000.

Figure 4 shows the total number of individuals throughout the generations that have been able to produce a complete and valid timetable that does not violate any hard constraints.

Figure 5 shows the total number of the exam selection grammar components in each generation, while Figure 6 shows the same but for the slot selection heuristics. The graphs on the left is drawn using data from all individuals, while the graphs on the right is drawn from using stats for the individuals with highest fitness in each generation. Generally speaking, a feature with an increasing number of nodes during the generations indicates that individuals with this feature (nodes) are more likely to survive the evolution selection process. This also gives an indication that this feature is probably more effective than other similar features in solving this particular problem. As shown by the graphs in the figure, there is no heuristic that is best on all the benchmark instances and throughout the generations, but one can understand from these graphs how good these heuristic are on specific problems. In future work we hope that by analysing this information and trying to feed it into the grammar we may further improve performance, e.g., by guiding GP in initialising the population rather than initialising the population randomly.

Figure 7 shows a comparison between the total count of the Random and First selection primitives of the grammar in all the individuals throughout the generations. Some of the graphs compare the First and Random selection heuristics on the exams selection. Other graphs compare the same heuristics on the slots selection. Figure 7c compares both of them. The data in Figures 7f and 7e are taken from the same experimental run. In the first one the data is taken from the 1000 individuals in the experiment, while in the second the data is collected from the best 50 individuals of each generation. The graphs in Figure 7 show that there is no single dominating strategy on all problems in the benchmark. However, we have noticed from the large number of experiments that we have done on these problem, that the Random selection heuristic is more frequent throughout the generations of different experiments suggesting that there are benefits in using some randomness.

VII. Conclusion

In this paper we have introduced a Grammar Based Genetic programming HH framework for evolving constructive heuristics for timetabling. The framework was tested on one of the most widely used benchmarks in the field of exam timetabling and compared with the best state-of-the-art approaches. Results show that the framework is very competitive with other constructive techniques, and did outperform other HH frameworks on many occasions. Also we provided some analyses of the behaviour of GP-HH. As for future work, we will look into feeding information gained from this study into the grammar, so as to better guide GP.

VIII. Acknowledgements

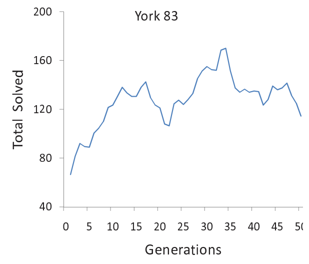
The authors acknowledge financial support from EPSRC (grants EP/C523377/1 and EP/C523385/1).

References

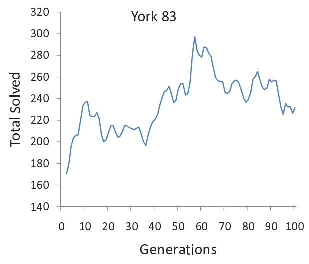
[1] E. Burke, G. Kendall, J. Newall, E. Hart, P. Ross, and S. Schulenburg, "Hyper-heuristics: An emerging direction in modern search technology," in *Handbook of Metaheuristics*, F. Glover and G. Kochenberger, Eds. Kluwer Academic Publishers, 2003, pp. 457–474.

[2] E. K. Burke, G. Kendall, and E. Soubeiga, "A tabu-search hyperheuristic for timetabling and rostering," *Journal of Heuristics*, vol. 9, no. 6, pp. 451–470, 2003.

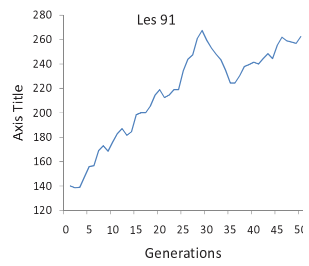
[3] E. K. Burke, S. Petrovic, and R. Qu, "Case-based heuristic selection for timetabling problems," *J. of Scheduling*, vol. 9, no. 2, pp. 115–132, 2006.



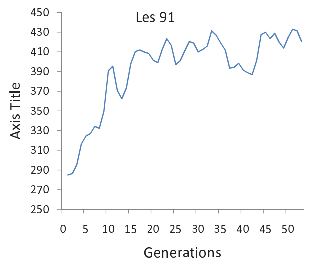
(a)



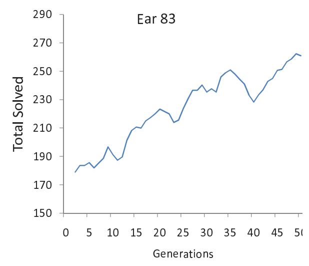
(b)



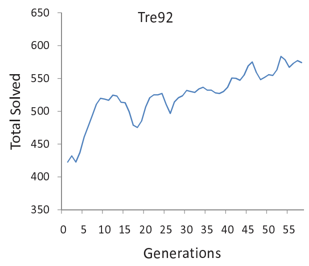
(c)



(d)



(e)



(f)

Fig. 4: The total number of individuals that have been able to generate a complete and valid timetable for a problem: (a) 500 individuals and 50 generations, (b) 500 individuals and 100 generations, (c) 500 individuals and 50 generations, (d) 1000 individuals and 50 generations, (e) 500 individuals and 50 generations, (f) 1000 individuals and 60 generations.

[4] P. Cowling, G. Kendall, and L. Han, "An investigation of a hyper-heuristic genetic algorithm applied to a trainer scheduling problem," in *Proceedings of the 2002 Congress on Evolutionary Computation CEC2002*. Washington, DC, USA: IEEE Computer Society, 2002, pp. 1185–1190.

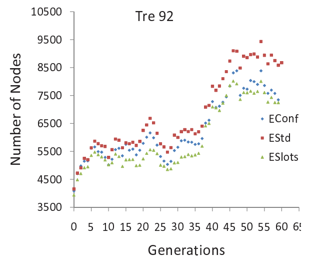
[5] H. A. Abbass, "Mbo: Marriage in honey bees optimization - a haplometrosis polygynous swarming approach," in *Proceedings of the 2001 Congress on Evolutionary Computation CEC2001*. IEEE Press, 27-30 May 2001, pp. 207–214.

[6] R. Poli, J. Woodward, and E. K. Burke, "A histogram-matching approach to the evolution of bin-packing strategies," in *2007 IEEE Congress on Evolutionary Computation*, D. Srinivasan and L. Wang, Eds., IEEE Computational Intelligence Society. Singapore: IEEE Press, 25-28 Sep. 2007, pp. 3500–3507.

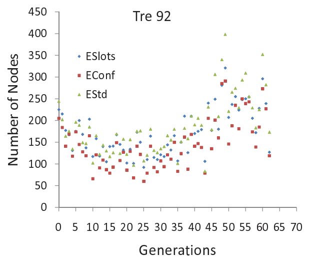
[7] M. B. Bader-El-Den and R. Poli, "Inc*: An incremental approach for improving local search heuristics," in *EvoCOP*, ser. Lecture Notes in Computer Science, J. I. van Hemert and C. Cotta, Eds., vol. 4972. Springer, 2008, pp. 194–205.

[8] D. Welsh and M. Powell, "An upper bound for the chromatic number of a graph and its application to timetabling problems," *The Computer Journal*, vol. 10, no. 1, pp. 85–87, 1967.

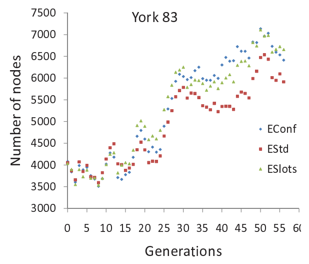
[9] H. Asmuni, E. K. Burke, J. M. Garibaldi, and B. McCollum, "Fuzzy multiple heuristic orderings for examination timetabling," in *PATAT*, ser. Lecture Notes in Computer Science, E. K. Burke and M. A. Trick, Eds., vol. 3616. Springer, 2004, pp. 334–353.



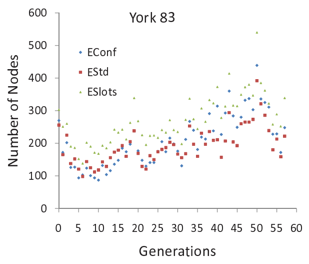
(a) All Individuals



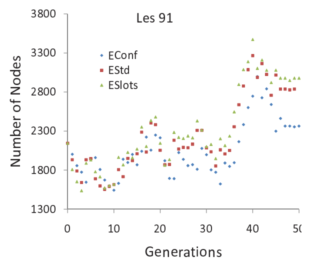
(b) Best 25 of 1000 individuals



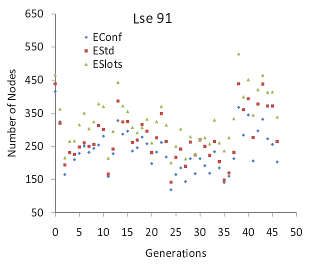
(c) All Individuals



(d) Best 50 of 1000 individuals



(e) All Individuals



(f) Best 50 of 1 individuals

Fig. 5: The total number of the exam selection grammar components in each generation, the data in the graphs on the left hand side is collected from all individuals exist in each generation, in (b) the data is collected from the best 25 individuals out of 1000 shown in (a), in (d) the data is collected from the best 50 individuals out of 1000 shown in (d), in (f) the data is collected from the best 25 individuals out of 1000 shown in (e).

[10] E. K. Burke, B. McCollum, A. Meisels, S. Petrovic, and R. Qu, "A graph-based hyper-heuristic for educational timetabling problems," *European Journal of Operational Research*, vol. 176, no. 1, pp. 177–192, January 2007. [Online]. Available: http://ideas.repec.org/a/eee/ejores/v176y2007i1p177-192.html

[11] L. D. Gaspero and A. Schaerf, "Tabu search techniques for examination timetabling," in *PATAT*, ser. Lecture Notes in Computer Science, E. K. Burke and W. Erben, Eds., vol. 2079. Springer, 2000, pp. 104–117.

[12] G. Kendall and N. M. Hussin, "An investigation of a tabu search based hyper-heuristic for examination timetabling," in *Multidisciplinary Scheduling: Theory and Applications*. Springer, 2005, pp. 309–328.

[13] H. Terashima-Marin, P. Ross, and M. Valenzuela-Rendon, "Evolution of constraint satisfaction strategies in examination timetabling," in *Proceedings of the Genetic and Evolutionary Computation Conference*, vol. 1. Orlando, Florida, USA: Morgan Kaufmann, 13-17 July 1999, pp. 635–642.

[14] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA, USA: MIT Press, 1992.

[15] W. B. Langdon and R. Poli, *Foundations of Genetic*

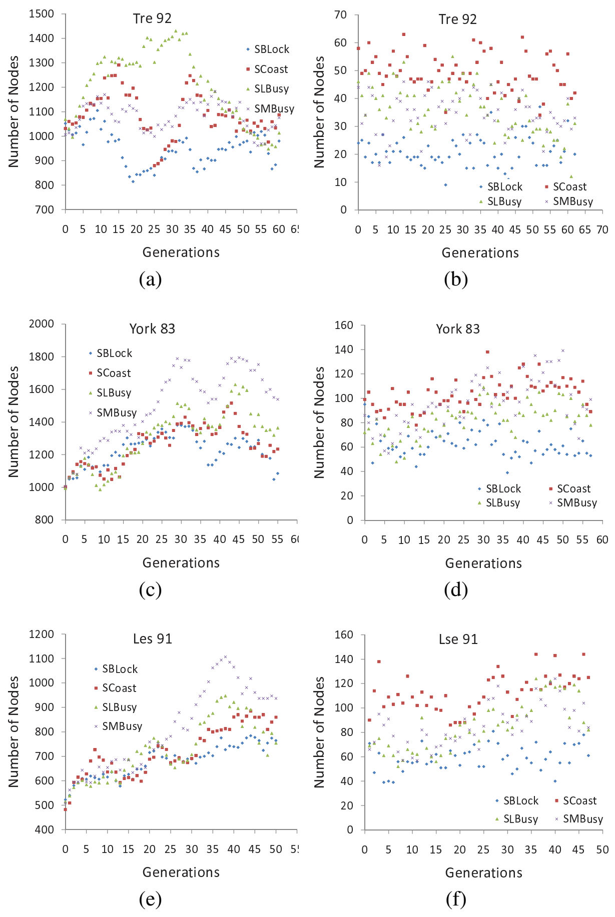


Fig. 6: The total number of the slot selection heuristics in each generation, the data in the graphs on the left is collected from all individuals exist in each generation, in (b) the data is collected from the best 25 individuals out of 1000 shown in (a), in (d) the data is collected from the best 50 individuals out of 1000 shown in (d), in (f) the data is collected from the best 25 individuals out of 1000 shown in (e).

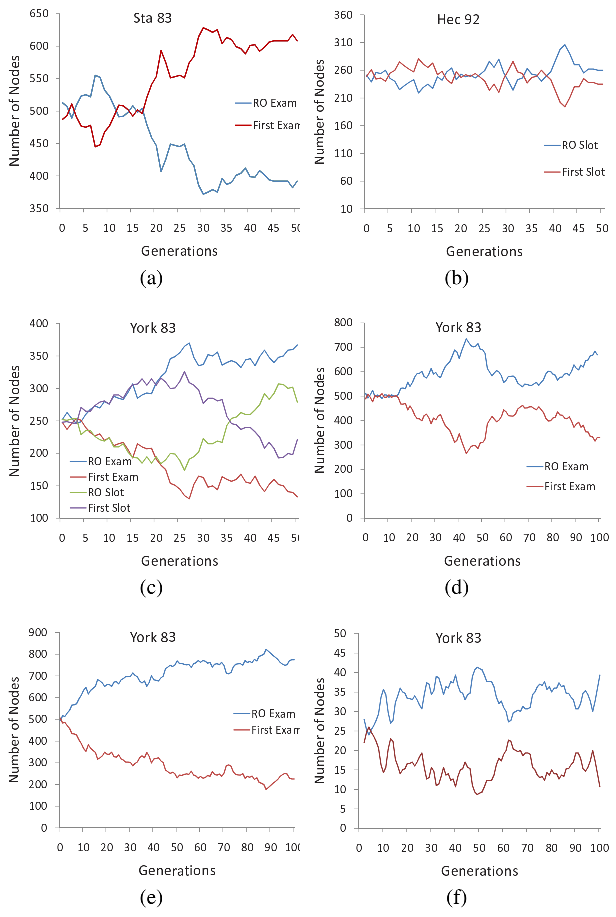


Fig. 7: A comparison between the total count of the grammar random and first selection nodes in all the individualise through out the generations. (a) 1000 individuals and 50 generation, (b) 500 individuals and 50 generation, (c) 500 individuals and 60 generation, (d) 1000 individuals and 100 generation, (e) 1000 individuals and 100 generation, (f) best 50 individuals out of 1000 and 100 generation.

Programming. Springer-Verlag, 2002. [Online]. Available: http://www.cs.ucl.ac.uk/staff/W.Langdon/FOGP/

[16] R. Poli, W. B. Langdon, and N. F. McPhee, *A field guide to genetic programming*. Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`, 2008. [Online]. Available: http://www.gp-field-guide.org.uk

[17] M. B. Bader-El-Den and R. Poli, "Generating sat local-search heuristics using a gp hyper-heuristic framework," in *Artificial Evolution*, ser. Lecture Notes in Computer Science, N. Monmarché, E.-G. Talbi, P. Collet, M. Schoenauer, and E. Lutton, Eds., vol. 4926. Springer, 2007, pp. 37–49.

[18] R. E. Keller and R. Poli, "Self-adaptive hyperheuristic and greedy search," in *2008 IEEE World Congress on Computational Intelligence*, J. Wang, Ed., IEEE Computational Intelligence Society. Hong Kong: IEEE Press, 1-6 Jun. 2008.

[19] M. W. Carter, G. Laporte, and S. Y. Lee, "Examination timetabling: Algorithmic strategies and," *Journal of Operational Research Society*, vol. 47, pp. 73–83, 1996.

[20] S. Abdullah, S. Ahmadi, E. K. Burke, and M. Dror, "Investigating ahuja-orlin's large neighbourhood search for examination timetabling," Tech. Rep., 2004.

[21] E. K. Burke and J. P. Newall, "Enhancing timetable solutions with local search methods," in *PATAT*, ser. Lecture Notes in Computer Science, E. K. Burke and P. D. Causmaecker, Eds., vol. 2740. Springer, 2002, pp. 195–206.

[22] E. Burke, Y. Bykov, J. Newall, and S. Petrovic, "A time-predefined local search approach to exam timetabling problems," *IIE Transactions*, vol. 36, no. 6, pp. 509–528, Jun 2004. [Online]. Available: http://www.asap.cs.nott.ac.uk/publications/pdf/Tpls.pdf

[23] M. Caramia, P. Dell'Olmo, and G. F. Italiano, "New algorithms for examination timetabling," in *WAE '00: Proceedings of the 4th International Workshop on Algorithm Engineering*. London, UK: Springer-Verlag, 2001, pp. 230–242.

[24] S. Casey and J. Thompson, "Grasping the examination scheduling problem," in *PATAT*, ser. Lecture Notes in Computer Science, E. K. Burke and P. D. Causmaecker, Eds., vol. 2740. Springer, 2002, pp. 232–246.

[25] L. T. G. Merlot, N. Boland, B. D. Hughes, and P. J. Stuckey, "A hybrid algorithm for the examination timetabling problem," in *PATAT*, ser. Lecture Notes in Computer Science, E. K. Burke and P. D. Causmaecker, Eds., vol. 2740. Springer, 2002, pp. 207–231.

[26] E. K. Burke and P. D. Causmaecker, Eds., *Practice and Theory of Automated Timetabling IV, 4th International Conference, PATAT 2002, Gent, Belgium, August 21-23, 2002, Selected Revised Papers*, ser. Lecture Notes in Computer Science, vol. 2740. Springer, 2003.