Evolving Construction Ordering Heuristics for Educational Timetabling Problems

Nelishia Pillay · Ender Özcan

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

1 Introduction

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. This research presents generation hyper-heuristics [1] to evolve low-level construction heuristics for the examination timetabling and university course timetabling problems. The hyper-heuristics use genetic programming [2]. The following section presents the genetic programming (GP) hyper-heuristics, the experimental setup and preliminary results.

Nelishia Pillay
School of Mathematics, Statistics and Computer Science
University of KwaZulu-Natal, South Africa
E-mail: pillayn32@ukzn.ac.za

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

2 Genetic Programming Hyper-Heuristics and Experiments

The gentic programming hyper-heuristic employs the *generational control model* [2] which evolves the population over a set number of generations with the population size remaining the same for all generations and a new population of offspring replacing the population of the previous generation. The *grow method* [2] is used to create the initial population. This method randomly selects elements from the function and terminal set until the maximum permitted tree depth is reached, at which point nodes are only selected from the terminal set. The terminal set consists of variables representing problem characteristics and existing low-level heuristics commonly used for this domain. For example, for the examination timetabling problem the following terminals are used:

– a: Measure of the distance between slots of examinations that share students.
– b: Number of potential clashes the examination has with unallocated examinations.
– c: Number of potential clashes for the examination.
– d: Number of students taking the examination.
– e: Number of students potentially involved in clashes.
– f: Number of slots in the timetable that the examination can be allocated to which will not result in hard constraint violations.

Two different approaches are tested. The first genetic programming hyper-heuristic approach, denoted as AGPHH, combines the problem characteristics and low-level heuristics arithmetically and the second one, denoted as LGPHH, combines them logically. The function set for AGPHH includes arithmetic operators (+, -, *, /), relational operators (<, >, <=, >=. ==, !=) and an *if-than-else* operator. Each heuristic in LGPHH combines problem characteristics/existing low-level heuristics 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 AGPHH as well as variables representing three period selection heuristics, namely, first period (f), random period (r) and minimum penalty cost period (m). The * is used to combine 2 or 3 terminals. The terminals are applied hierarchically, i.e. the second terminal is used to break ties of the first terminal and the third terminal is used to break ties of the second terminal. The + is used to combine the terminals representing problem characteristics/low-level heuristics with one of the period selection heuristics. In the case of AGPHH an arithmetic value is the output which represents a measure of the difficulty of scheduling the event. For example, *(b+d)*e* will add the number of potential clashes for unallocated examinations to the number of students and multiply this by the number of students potentially involved in clashes. Events will be sorted in descending order according to this value and allocated accordingly. In the case of LGPHH the characteristics are applied logically in a hierarchical manner, namely, the first characteristic is applied and in the case of a tie the second characteristic is applied and so on. For example, **+fam* in which case events to be allocated will be sorted in ascending order according to the number of feasible slots to allocate the event to in the current version of the timetable, in the case of events being tied on this value the distance between slots that share students will be applied and the minimum penalty cost period will be used to decide which period to allocate the event to.

The fitness of each individual is calculated by using the heuristic to sort the exams or courses in descending order and allocating them to the timetable in order. The minimum penalty period is used to select a period in the case of AGPHH. The fitness is a product of the hard constraint cost plus one and the soft constraint cost of the constructed timetable. Tournament selection is used to select parents which the mutation and crossover operators are applied to for regeneration.

The Toronto benchmark set, the ITC 2007 examination timetabling benchmark set and the ITC 2007 curriculum based benchmark are used to evaluate AGPHH and LGPHH. Due to the stochastic nature of genetic programming thirty runs will be performed for each problem instance. For both AGPHH and LGPHH a population size of 500, 60 generations, with a tournament size of 4 and mutation and crossover application rates of 50% were found to be sufficient. A depth limit of 4 for the initial population for AGPHH was found to be most suitable. These values have been determined empirically by performing trial runs and there has been no formal parameter tuning conducted. Hypothesis tests using the Z-test will be used to test the significance of the results obtained in terms of the difference in means of AGPHH and LGPHH in solving the three problems. The genetic programming approaches have been implemented in Java and all simulations have been run on an HP Z80 workstation with Windows 7.

The the following experiments have been completed at this stage:

- AGPHH applied to the Toronto benchmark set
- AGPHH applied to the ITC 2007 examination timetabling set
- AGPHH applied to the ITC 2007 curriculum based university course timetabling set
- LGPHH applied to the ITC 2007 curriculum based university course timetabling set

The results obtained thus far are promising with the generated construction heuristics having performed better than the existing heuristics. Runtimes for AGPHH ranged from 1 minute to 19 hours and for LGPHH from 1 minute to 27 minutes. AGPHH did not perform as well for the curriculum based university course timetabling problem as it did for both the examination timetabling benchmark sets. LGPHH was found to outperform AGPHH for the curriculum based university course timetabling problem and was able to evolve heuristics producing feasible timetables in cases that existing heuristics and AGPHH could not. Different heuristics performed the best on each of the runs of AGPHH for each problem instance. In the case of LGPHH for all of the problem instances, with an exception of one problem instance which used the random period heuristic, the best evolved heuristic was the same or one of two heuristics for all runs.

We will present our approach and the experimental results at the conference. The simulations for the application of LGPHH to the examination timetabling benchmark sets are currently being conducted. The results of these will be presented together with a more detailed comparison of the AGPHH and LGPHH on these three benchmark sets including examples of the best performing heuristics.

References

1. E.K. Burke, M. Gendreau, M.R. Hyde, G. Kendall, G. Ochoa, E. Ozcan, R. Qu, Hyper-heuristics: A Survey of the State of the Art. Journal of the Operational Research Society 64(12), 1695, 1724 (2013)
2. J.R. Koza Genetic Programming: On the Programming of Computers by Means of Natural Selection, MIT Press, USA(1992)