UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

COS790 Assignment 2

Selection Perturbative Hyper-Heuristics

Student Number: u15024522

1 Introduction

This assignment involves comparing the performance of a single-point selection perturbative hyper-heuristic and multi-point selection perturbative hyper-heuristic for solving the examination timetabling problem.

More specifically, the single-point hyper-heuristic employs a greedy selection technique whereas the multi-point hyper-heuristic explores a space of heuristic combinations through the use of a genetic algorithm. The hyper-heuristics are evaluated on at least two early, two late and two hidden problem instances from the ITC 2007 examination timetabling benchmark set [1] (available at `www.cs.qub.ac.uk/itc2007/examtrack/exam_track_index_files/outputformat.htm`).

As a precursor to further detailing the single and multi-point hyper-heuristics it is important to note how the initial solution is generated. For every exam that needs to be scheduled, a random room is selected. Then, the first period which will not lead to any hard constraint violations is selected and the booking added to the solution. If no viable period could be found then both the room and period for the exam are selected randomly.

Furthermore, the low-level heuristics which make up the search space being explored consist of 5 perturbative heuristics for improving upon an examination timetabling solution in which all exams have been scheduled; albeit, with possible hard-constraint violations. The heuristic set H is detailed in table 1.

Symbol	**Description**
A	Find the first 2 bookings that share the same period and have exams with the same students; then, reschedule both exams using to the rescheduling technique.
B	Find the first room that has been overbooked for a specific period; then, reschedule all exams that have been scheduled in the room for that period using to the rescheduling technique.
C	Find the first booking where the period is too short for the scheduled exam; then, reschedule the exam using the rescheduling technique.
D	Find the first 2 bookings that violate a period-related hard constraint; then, reschedule both exams using to the rescheduling technique.
E	Find the first booking that has violates a room-related hard constraint; then, reschedule the exam using the rescheduling technique.

Table 1: Low-Level Perturbative Heuristics (H)

Importantly, the rescheduling technique used by all low-level heuristics in table 1 reschedules an exam by following a series of steps. Firstly, the booking associated with the exam is removed from the current solution. Next, a new room is selected randomly from the set of rooms that are large enough for the exam. Penultimately, a random period is selected from the set of periods that are long enough for the exam and will not lead to a clash with another exam - if no such period exists, any random period is selected. Then, finally, a new booking for the exam is made using the selected room and period to produce a new solution.

2 Single-Point Search

The single-point search selection perturbative hyper-heuristic applies a low-level heuristic - from the heuristic set in table 1 - n times, where n has been arbitrarily chosen as the number of exams that need to be scheduled for the problem instance.

To select the low-level heuristic to apply a greedy selection technique is used. So, for each of the n iterations, all low-level heuristics are applied to the existing solution to produce 5 new solutions - one for each low-level heuristic. Then, out of these 5 solutions, the solution which violates the least hard constraints is chosen to replace the current solution. Algorithm 1 details this single-point search approach in its entirety.

Algorithm 1: Single-Point Search Selection Perturbative Hyper-Heuristic

```
currentSolution = initialSolution;
for i ← 1 to n do
    foreach heuristic in H do
        newSolution = heuristic.applyTo(currentSolution);
        if newSolution.violations() < bestSolution.violations() then
            bestSolution = newSolution;
    end
    currentSolution = bestSolution;
end
return currentSolution;
```

3 Multi-Point Search

As has been mentioned, the multi-point selection perturbative hyper-heuristic employs a genetic algorithm which explores the space of heuristic combinations. This approach is similar to that of Raghavjee and Pillay [2] in that low-level heuristic combinations are evolved to produce the combinations that best improve on an initial solution. In addition, Evohyp [3] - a Java toolkit for evolutionary algorithm hyper-heuristics - was used for the implementation of the genetic algorithm.

A single chromosome in the genetic algorithm population is represented as a heuristic combination - for example, *AEBDACA*. To evaluate the chromosome, the heuristic combination is applied to the initial solution to produce a new solution. Each low-level heuristic is applied to the solution until the solution has been changed n times, where n has again been arbitrarily chosen as the number of exams that need to be scheduled for the problem instance. The means of applying a perturbative heuristic sequence is shown in algorithm 2.

Following the evaluation a heuristic sequence, the fitness of a solution is calculated in the same manner as suggested by Pillay [4] in that fitness is a measure of the soft constraint cost, multiplied by the hard constraint cost incremented by one.

Algorithm 2: Applying a Perturbative Heuristic Sequence

currentSolution = initialSolution;

for $i \leftarrow 1$ **to** n **do**
    heuristic = H.elementAt($i \mod H.length$);
    currentSolution = heuristic.applyTo(currentSolution);
end

return currentSolution;

Furthermore, the genetic algorithm employs tournament selection as the selection method and uses crossover and mutation as genetic operators - these techniques are all supported by the Evohyp [3] toolkit. Of course, as is the case with any genetic algorithm approach, there are specific parameters required by the algorithm. The parameters, in this case, were largely chosen based on proportions that have been used previously with similar approaches to the same benchmark set [4, 2]. However, many parameters were scaled down to smaller values. The genetic parameters used were selected as follows:

- The Population Size was selected as 50.
- The Tournament Size was selected as 5.
- The Number of Generations was selected as 10.
- The Mutation Rate was selected as 0.3.
- The Crossover rate was selected as 0.7.
- The Maximum Initial Length was selected as 20.
- The Maximum Offspring Length was selected as 20.
- The Mutation Length was selected as 5.

4 Experimental Setup

As has also been mentioned, all problem instances were extracted from the ITC 2007 examination timetabling benchmark set [1] (available at `www.cs.qub.ac.uk/itc2007/examtrack/exam_track_index_files/outputformat.htm`). From the benchmark set [1] two early, two late and two hidden problem instances were used to run experiments on. These instances were comprised of *exam_comp_set1.exam*, *exam_comp_set4.exam*, *exam_comp_set6.exam*, *exam_comp_set8.exam*, *exam_comp_set9.exam* and *exam_comp_set12.exam*. A total of 10 test runs were run on each problem instance for both the single and multi-point search approaches.

In addition, the machine used for development and testing purposes was a 2015 15-inch Macbook Pro with a 2.5 GHz Quad-Core Intel Core i7 processor, 16GB of memory and 500GB of storage. At the time of experimentation the machine had a battery cycle count of 684.

Finally, all of the code and data necessary to reproduce the results is available at `https://github.com/marcus-bornman/cos_790_assignment_2`.

5 Results

As has been covered in section 4, 10 test runs were executed for each problem instance, for each selection perturbative hyper-heuristic. The experimental results are shown in table 2. The table includes the best objective value, average objective value, standard deviation and average runtime (in milliseconds) for each of the approaches on each of the problem instances.

Approach	Best Obj. Value	Avg. Obj. Value	Std. Deviation	Avg. Runtime
examcompset1.exam				
Single-Point	1253112	1592636	279123	8928
Multi-Point	2437039	2437039	374749	84562
examcompset4.exam				
Single-Point	3192210	4204729	691143	523
Multi-Point	4166792	5009148	792348	15637
examcompset6.exam				
Single-Point	2636550	5528555	1169006	641
Multi-Point	3995915	5626109	800689	8876
examcompset8.exam				
Single-Point	8702334	18938160	6314929	14529
Multi-Point	7035282	10595753	1335078	345627
examcompset9.exam				
Single-Point	76400	132675	34774	361
Multi-Point	32340	48952	13381	7342
examcompset12.exam				
Single-Point	788840	1474645	396605	305
Multi-Point	885300	989088	61439	3875

Table 2: Experimental Results

6 Discussion

When considering the performance of the two approaches used, one must consider their efficacy in solving the examination timetabling problems from the benchmark set as well as their ability to generalise across different problem instances. In addition, these considerations must be made when comparing the approaches to one another and to other approaches used in the literature.

In comparing the efficacy of the two approaches in solving the given problem instances, there is little evidence to suggest that one approach is more effective. Each approach achieved the better average objective value for 3 of the 6 problem instances while the single-point approach achieved the best overall objective value for 4 of the 6 instances. This indicates relatively similar efficacy in solving the given problems. It is worthwhile to consider, however, that one approach may fair better with larger problem instances but there is insufficient evidence to support this.

When considering the generalisation of the two approaches it would seem that the multi-point search approach does perform slightly better, considering the

proportionate standard deviations across problem instances as opposed to the single-point approach. However, it does seem this comes at the cost of performance, with the multi-point search displaying immensely longer runtimes.

Finally, the results of both approaches are not comparable to other approaches in the literature, such as those covered by Pillay [5], due to their inability to find a viable solution. However, an avenue for future research may be to explore an increase in size of the genetic parameters for the multi-point search to discover whether a feasible solution can be found given a larger population or more generations.

References

[1] B. McCollum, A. Schaerf, B. Paechter, P. McMullan, R. Lewis, A. J. Parkes, L. D. Gaspero, R. Qu, and E. K. Burke, "Setting the research agenda in automated timetabling: The second international timetabling competition," *INFORMS Journal on Computing*, vol. 22, no. 1, pp. 120–130, 2010.

[2] R. Raghavjee and N. Pillay, "A genetic algorithm selection perturbative hyper-heuristic for solving the school timetabling problem," *ORiON*, vol. 31, no. 1, pp. 39–60, 2015.

[3] N. Pillay and D. Beckedahl, "Evohyp-a java toolkit for evolutionary algorithm hyper-heuristics," in *2017 IEEE Congress on Evolutionary Computation (CEC)*, pp. 2706–2713, IEEE, 2017.

[4] N. Pillay, "Evolving hyper-heuristics for a highly constrained examination timetabling problem," in *Proceedings of the 8th International Conference on the Practice and Theory of Automated Timetabling (PATAT'10)*, pp. 336–346, 2010.

[5] N. Pillay, "A review of hyper-heuristics for educational timetabling," *Annals of Operations Research*, vol. 239, no. 1, pp. 3–38, 2016.