Selection Constructive Hyper-Heuristics

Marcus Bornman
Computer Science Department
University of Pretoria
Pretoria, South Africa
u15024522@tuks.co.za

***Abstract*—Selection constructive hyper-heuristics have previously been employed for solving the examination timetabling problem. However, most of this research has focused on selecting the heuristic to choose the examination to schedule next. This assignment involves extending this idea by employing a selection constructive hyper-heuristic that selects a heuristic to choose the examination and a heuristic to choose the period the examination should be placed in.**

***Index Terms*—Hyper-heuristics, Examination timetabling, Evolutionary algorithms**

I. Introduction

Selection constructive hyper-heuristics have previously been employed for solving the examination timetabling problem and results have been positive. This report aims to expand on existing approaches by including low-level heuristics for period selection in addition to those for examination selection.

A genetic algorithm selection hyper-heuristic is used to select a heuristic combination between 3 examination selection and 3 period selection heuristics.

Performance of the hyper-heuristic is tested with defined parameters using problem instances provided by The Second International Timetabling Competition [1].

Finally, results are summarized and compared to those of similar approaches in other relevant literature.

II. Background

This section provides an overview of the 3 low-level heuristics for examination selection, the 3 low-level heuristics for period selection and hyper-heuristic used for selection.

A. Low-level heuristics for examination selection

In terms of selecting the next exam to schedule, three different low-level heuristics were used.

The first of these three heuristics, hereon referred to as $MostClashes$, entails the unbooked exam with most clashes being selected next. The pseudocode for this heuristic can be described as follows:

```
unbookedExams[] = findUnbookedExams()
mostExam = unbookedExams[0]
for (exam : unbookedExams)
  if (exam.examClashes() >
      mostExam.examClashes())
    mostExam = exam
return mostExam
```

The second of the low-level heuristics for examination selection can be referred to as $MostStudents$. This heuristic entails selecting the unbooked exam with the most students next. The peudocode for this heuristic is as follows:

```
unbookedExams[] = unbookedExams()
sortByMostStudents(unbookedExams)
return unbookedExams[0]
```

The last heuristic for examination selection, described as $WeightedClashes$, is similar to $MostClashes$. However, instead of selecting the exam that clashes with the most other exams, the exam which has the most students that have clashes with other exams is selected. The pseudocode is as follows:

```
unbookedExams[] = findUnbookedExams()
mostExam = unbookedExams[0]
for (exam : unbookedExams)
  if (exam.studentClashes() >
      mostExam.studentClashes())
    mostExam = exam
return mostExam
```

B. Low-level heuristics for period selection

As for examination selection, three low-level heuristics were used for period selection.

Firstly, the $LeastBookings$ heuristic selects the period for which the least bookings have already been made. The pseudocode is included below:

```
leastPeriod = periods[0]
for (period : periods)
  if (period.bookings() >
      leastPeriod.bookings())
    leastPeriod = period
return leastPeriod
```

Next, the $LeastSameDay$ heuristic selects the period for which the least bookings have been made on the same day. The pseudocode for this heuristic is as follows:

```
leastPeriod = periods[0]
for (period : periods)
  if (period.bookingsOnSameDay() >
      leastPeriod.bookingsOnSameDay())
    leastPeriod = period
return leastPeriod
```

Finally, the $LeastStudents$ heuristic selects the period for which the least students have been booked in the same period. The pseudocode can be described as follows:

```
leastPeriod = periods[0]
for (period : periods)
  if (period.studentsBooked() >
      leastPeriod.studentsBooked())
    leastPeriod = period
return leastPeriod
```

C. Genetic algorithm hyper-heuristic

A genetic algorithm selection hyper-heuristic was used as part of a Java program to select and test the examination-selection and period-selection low-level heuristics. More specifically, the *EvoHyp* [2] Java Toolkit was employed to expose the implementation of the hyper-heuristic. As described by Pillay and Beckedahl [2], the general algorithm followed by the hyper-heuristic is as follows:

```
Create an initial population
repeat
   Evaluate the population
   Select parents
   Apply genetic operators
until termination criterion is met
```

In terms of the chromosomes explored by the genetic algorithm, each chromosome represents a combination of heuristics to apply to the solution in order. Notably, each gene of a chromosome consists of one heuristic for examination selection and one heuristic for period selection. Given the possible heuristics covered previously, this leads to 9 possible genes:

- 'A' consists of $MostClashes$ and $LeastBookings$
- 'B' consists of $MostClashes$ and $LeastSameDay$
- 'C' consists of $MostClashes$ and $LeastStudents$
- 'D' consists of $MostStudents$ and $LeastBookings$
- 'E' consists of $MostStudents$ and $LeastSameDay$
- 'F' consists of $MostStudents$ and $LeastStudents$
- 'G' consists of $WeightedClashes$ and $LeastBookings$
- 'H' consists of $WeightedClashes$ and $LeastSameDay$
- 'I' consists of $WeightedClashes$ and $LeastStudents$

This means, for example, that a single chromosome may be represented as 'CGECBCBBCCBCCB' or 'HCHAB'.

The genetic algorithm also required the selection of parameters which are detailed within the experimental setup.

III. Experimental Setup

This section will describe parameter values used for the algorithms, problem instances used and technical specifications of the machine used to develop and run simulations. Note that the code, problem instances and results of this assignment are available at the following public github repository: *https://github.com/marcus-bornman/heuristic_exam_timetable*.

A. Parameter values

As mentioned, the genetic algorithm selection hyper-heuristic that was used required the selection of various parameters. The selected values for these parameters are detailed below.

- The population sizes was selected as 50.
- The tournament size was selected as 10.
- The number of generations was selected as 100.
- The mutation rate was selected as 5%.
- The crossover rate was selected as 75%.
- The initial max length was selected as 10.
- The offspring max length was selected as 50.
- The mutation length was selected as 5.

In addition to these parameters, in the context of the genetic algorithm, it is important to make note of the values and parameters used for evaluating the fitness of solutions. Since the data sets provided by The Second International Timetabling Competition [1] were used, the hard and soft constraints presented for the competition were also used to calculate fitness. More details regarding these constraints can be found at *http://www.cs.qub.ac.uk/itc2007/examtrack/exam_track_index_files/examevaluation.htm* but, briefly, the fitness method can be summarised as follows:

```
double fitness = 0;

// Hard constraints
fitness += 100 * calcUnbookedExams()
fitness += 100 * calcConflictingExams()
fitness += 100 * calcOverbookedPeriods()
fitness += 100 * calcTooShortPeriods()
fitness += 100 * calcPeriodViolations()
fitness += 100 * calcRoomViolations()

// Soft constraints
fitness += calcTwoInARowPenalty()
fitness += calcTwoInADayPenalty()
fitness += calcPeriodSpreadPenalty()
fitness += calcMixedDurationsPenalty()
fitness += calcFrontloadPenalty()
fitness += calcRoomPenalty()
fitness += calcPeriodPenalty()

return fitness
```

As can be seen, a static weight of 100 was applied to all hard constraint violations whilst soft constraint violation weightings were dependant on the problem instance.

B. Problem instances

The Second International Timetabling Competition [1] was conducted using 4 early, 4 late and 4 hidden data sets. The combination of these data sets was used to test the genetic algorithm selection hyper-heuristic against a total of all 12 data sets.

As described in the specification of the competition, all instances comprise real data and come from the University of Udine. Also, all instances have at least one feasible solution, but the optimal solution to minimize soft constraints

is not known. Each instance is provided as a single file. The exact format is described at *http://www.cs.qub.ac.uk/itc2007/examtrack/exam_track_index_files/Inputformat.htm*.

To compile a comprehensive analysis of the hyper-heuristic's performance against each problem instance a total of 10 test runs was completed against every instance.

C. Technical specifications

To put run times into context it is important to note the technical specifications of the machine used to develop the program and run simulations. The machine used 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.

IV. Research Results

A. Summary of Results

As has been mentioned, 10 test runs were completed against every problem instance using the genetic algorithm selection hyper-heuristic to select the combinations of low-level heuristics which provided the best initial solution to the problem instance. The table below summarizes the best objective value, average objective value and standard deviation of the best objective value from the optimum over the ten runs for each problem instance.

Problem Instance	Lowest	Mean	Std. Dev.	Avg. Runtime
1	58560	59031.2	348.4	15:53
2	86400	86591.5	107.39	14:35
3	92840	92947	58.66	12:49
4	26725	26765	40	12:32
5	97590	98558	511.15	16:30
6	23845	23850	15	0:23
7	107110	108611	601.07	15:03
8	56073	56522.4	379.99	17:53
9	15200	15588	235.76	13:47
10	20855	21048.5	94.13	6:16
11	92850	92948	80.97	4:26
12	7610	7650	66.33	0:27

B. Discussion of Results

When considering the performance of the tests run, we want to compare the results to reported by similar approaches. Hence, we will consider "Evolving hyper-heuristics for a highly constrained examination timetabling problem" [3], "A graph coloring constructive hyper-heuristic for examination timetabling problems" [4] and "Adaptive selection of heuristics for assigning time slots and rooms in exam timetables" [5].

Unfortunately, due to the parameters applied to the genetic algorithm selection hyper-heuristic in this report, no feasible solutions were found for any of the problem sets. Consequently, a direct comparison of the results to those in the other relevant literature is not possible. However, the results do support some general claims. Specifically, that this type of approach results in higher run times, better results for large problem sets and consistency in terms of convergence.

V. Conclusion(s)

This report explored the application of selection constructive hyper-heuristics in the selection of both examination selection and period selection heuristics for the examination timetabling problem.

The report included 6 low-level heuristics to make up the search dimensions as well as a genetic algorithm selection hyper-heuristic to explore the search space.

While some expected results - such as higher run times - were confirmed, no complete solutions to any problem sets were found so direct comparisons with results from related research were not possible.

Additional work is necessary to explore means of adjusting the hyper-heuristic to provide complete solutions which can be compared to the available literature.

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] 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.

[3] 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.

[4] N. R. Sabar, M. Ayob, R. Qu, and G. Kendall, "A graph coloring constructive hyper-heuristic for examination timetabling problems," *Applied Intelligence*, vol. 37, no. 1, pp. 1–11, 2012.

[5] A. Soghier and R. Qu, "Adaptive selection of heuristics for assigning time slots and rooms in exam timetables," *Applied Intelligence*, vol. 39, no. 2, pp. 438–450, 2013.