Selection Perturbative Hyper-Heuristic for Curriculum-based Course Timetabling Problem

Isheanesu Joseph Dzingirai

August 2023

Contents

1 Introduction

Efficiently organizing courses within an academic institution is a complex task that involves multiple constraints and objectives. The Curriculum-based Course Timetabling problem (CB-CTT) seeks to allocate courses to time slots and rooms while satisfying various institutional constraints. Traditional methods for solving the CB-CTT often rely on manual intervention and heuristics, which can be time-consuming and sub-optimal.

This report presents an approach to tackling the CB-CTT problem using a selection-perturbative hyper-heuristic (SPHH). Hyper-heuristics offer a meta-level approach by generating and selecting heuristics or heuristic combinations to solve optimization problems. In this study, we introduce the concept of a SPHH specifically designed for the CB-CTT.

Selection Perturbative Hyper-Heuristic is an approach that combines the selection of lower-level heuristics with perturbative techniques to solve or optimize combinatorial optimization problems. It aims to improve the optimization process by dynamically choosing and applying different heuristics and perturbations based on the characteristics of the problem and the current state of the solution.

To evaluate the effectiveness of the SPHH, comprehensive experiments were conducted using benchmark datasets from the International Timetabling Competition 2007 and compared against competition's best results as well as the results from Professor Pillay's 2016 paper. The results showcase the capability of the SPHH to consistently generate high-quality solutions across different problem instances. Additionally, it demonstrates robustness in handling various constraints of the CB-CTT.

2 Experimental Setup

2.1 System

The experiments were performed on a Dell Latitude 5510 laptop with the following machine specifications:

• RAM :	31.6 GB
• Disk Space:	417 GB
• CPU:	Intel(R) Core(TM) i7-10610U CPU @ 1.80GHz, 2304 MHz, 4 Core(s)

2.2 Programming Language and Libraries

The selection perturbative hyper heuristic was implemented using Python. No external libraries except for the built-in data structures from Python were used.

2.3 Problem Instances

The problem instances used are:

- comp04.ctt.txt
- comp08.ctt.txt
- comp09.ctt.txt
- comp12.ctt.txt
- comp13.ctt.txt
- comp18.ctt.txt
- comp21.ctt.txt
- ToyProblem.ctt.txt

These instances are selected to represent a diverse range of scenarios and challenges commonly encountered in the curriculum-based course timetabling domain.

2.4 Parameter Values

• Number of Perturbations :	1000
• Iteration Limited Threshold Acceptance Limit:	10
• Iteration Limited Threshold Acceptance Threshold:	20

3 Data Description

The data used to test the SPHH was obtained from the Optimization Hub. The data was used in the International Timetabling Competition 2007.

The structure of the data was:

1. The first 7 lines gave basic information about the problem instance. The basic information is:
    - name of problem instance
    - number of courses
    - number of rooms
    - number of days in the timetable
    - number of periods per day
    - number of unavailability constraints
2. Course - The following information is provided about the course:
    - id for unique identification
    - id of teacher who will be lecturing the course
    - number of lectures to be scheduled
    - number of minimum working days between scheduled lectures of the course
    - number of students enrolled in course
3. Room - The following information is provided about the room:
    - id for unique identification
    - room capacity

4. Curriculum - The following information is provided about the curriculum:
    - id for unique identification
    - number of courses in the curriculum (x)
    - x course ids for the courses in the curriculum
5. Unavailability Constraints - The following information is provided about the unavailability constraints:
    - course id that is constrained
    - the day that course is restricted
    - the period on the day that the course is constrained

4 Methodology

4.1 Approach for Hyper-Heuristic

The two computational optimization approaches that can be used to solve the Curriculum-based Course Timetable problem are: **Single-Point Search Selection Perturbative Hyper-Heuristic (SPS-SPHH)** and **Multi-Point Search Selection Perturbative Hyper-Heuristic (MPS-SPHH)**.

SPS-SPHH, in particular, is concerned with the concept of employing a single heuristic at a time from a pool of accessible heuristics. In contrast to MPS-SPHH, which considers the simultaneous application of many heuristics, SPS-SPHH concentrates on iteratively applying one heuristic until a stopping requirement is fulfilled. When the requirement is met, the hyper-heuristic chooses a new heuristic or an alternative approach to continue the search.

The selected approach for CB-CTT is the SPS-SPHH. The reasons are:

- It is computationally efficient because it acts on a single timetable at a time. This eliminates the overhead associated with managing several timetables.
- It adapts effectively to the characteristics of the current solution, potentially leading to better exploration of the solution space.

- When compared to MPS-SPHH approach, it is simpler to implement, understand and has a better convergence to high-quality solutions due to their focused nature (solutions are of high quality).

4.2 Objective Function

The goal of the SPS-SPHH is to find a feasible timetable that minimizes the combined cost of hard constraint violations and soft constraint violations. The definition of the constraints are:

- Hard Constraints - must not be broken under any circumstances. A feasible timetable is one that satisfies all of the hard constraints.
- Soft Constraints - desirable, but may be broken if all of them cannot be met. They differ in both nature and importance from one institution to the next. The quality of timetables is frequently judged by how much the soft constraints in the timetables are violated.

Overall, the SPS-SPHH finds a feasible timetable with no hard limitations and the fewest soft constraints.

4.2.1 Hard Constraints

The hard constraints for CB-CTT are:

- Lecture Allocations - The timetable must include the required number of lectures and each lecture must not be scheduled more than once in a period
- Conflicts - Lectures for courses in a curriculum must be arranged at distinct periods
- Room Occupancy - Each room must only be scheduled once in a period
- Teacher Availabilities - Each teacher must not be scheduled more than once in a period
- Lecture Availabilities - Each lecture must not be scheduled on the specific periods on specific days.

4.2.2 Soft Constraints

The soft constraints for CB-CTT are:

- Room Capacity - The number of students in a course assigned to a room for a lecture must not exceed the room's capacity
- Minimum Working Days - The lectures for a course must be spread out across the minimum number of working days indicated
- Curriculum Compactness - On each day, lectures for curriculum courses should be scheduled next to each other
- Room Stability - All lectures for a course should be held at the same room

4.3 Construction of Initial Timetable

4.3.1 Construction Heuristic

The initial timetable (solution) was constructed using the following construction heuristics:

- Saturation Degree (SD) - Priority is given to the lectures with the fewest number of feasible times in the timetable at the current stage of construction
- Largest Enrolment (LE) - Priority is given to the lectures with the most students

The saturation degree is a dynamic heuristic (its value is adjusted during the timetable construction process.) whereas largest enrolment is a static heuristic (computed before the timetable construction process and remain the same throughout the process). When selecting a course to schedule, if there are courses with the same saturation degree, the course with the greatest number of students enrolled is chosen.

4.3.2 Lecture Placement

The strategy used to select a period to schedule a lecture on the timetable is the Minimum Penalty Period (MPP). The reason why it was chosen instead of First Period (which schedules a lecture in the first feasible period in the timetable) and Random Period (which schedules a lecture in a randomly chosen period from all the feasible periods in the timetable) is because:

- Efficiency - starting with feasible solutions can help to speed up the optimization process. Because solutions generated by the Minimum Penalty Period strategy are more likely to be close to the optimal solution space, following optimization phases can concentrate on refining and increasing the quality of these solutions rather than rectifying serious violations (hard constraints).
- Because it actively seeks to reduce penalties, MPP tends to be more stable in terms of solution quality. Random First and First Period may provide very variable beginning solutions, making it impossible to predict the behavior of the optimization process.

With the construction heuristics and the lecture placement choice, the initial timetable was construction using the following algorithm:

Algorithm 1 Algorithm for Generating Initial Timetable

```
Input: The timetable object
Output: None
1: while ∃ course with lectures left to schedule do
2:     calculate the saturation degree for each course left to be scheduled
3:     select the course with the lowest saturation degree
4:     place the selected course in the most feasible slot
5:     update the number of lectures left to be scheduled for the course
6: end while
   =0
```

4.4 Low-Level Heuristics

A low-level heuristic (llh) is a domain-specific operation or modification that can be applied to a solution to make it more optimal or to explore new parts of the solution space. It is used in conjunction with a hyper-heuristic.

The low-level heuristics used with SPS-SPHH on CB-CTT are:

- Single Move - moves a single lecture selected a new feasible slot
- Swap Slots - swaps two slots selected at random in the timetable
- Swap Lectures - swaps two lectures selected at random in the timetable

4.5 Heuristic Selection

The technique used to select the low-level-heuristic to apply to the timetable is: **Reinforcement Learning**. Reinforcement Learning is a machine learning approach in which an agent learns to make sequential decisions based on feedback from the problem-solving environment to select and apply heuristics / functions.

In the context of the SPS-SPHH, it is employed as follows: Each of the heuristics is given a score. Initially, each heuristic has the same score. The score is increased or dropped dependent on the heuristic's performance.

The following are the reasons why it was chosen above other selection techniques:

- it provides a mechanism to choose the best low-level heuristics at each step of the perturbation process
- it assists the hyper-heuristic in selecting the best heuristic depending on the present state of the problem, ensuring that the algorithm remains effective even as the problem evolves.

Algorithm 2 Algorithm for Selecting Low Level Heuristic

```
Input: list of low-level heuristic
Output: low-level heuristic
1: select the low-level heuristic with the best score (highest score)
2: if there are multiple low-level heuristics with the same score then
3:     return randomly select one of the low-level heuristic with the best score
4: end if
5: return low-level heuristic with the best score
   =0
```

4.6 Move Acceptance

Move acceptance is used to determine whether to accept the perturbative heuristic's solution. The move acceptance employed in the SPS-SPHH for CB-CTT is: **Iterated limited threshold accepting (ILTA)**

ILTA typically accepts improving and equal movements, as well as moves that produce poorer timetables, if the current iteration exceeds the stated iteration limit, the hard constraints for the produced timetable are zero, and the soft constraints of the current timetable are greater than the previous timetable of the produced solution by an amount specified by the threshold.

The reasons ILTA was chosen over other move acceptance methods are:

- Avoidance of Local Optima - assists in avoiding such local optima by enabling the search to progress even if the new timetable is worse, increasing the chances of eventually finding a superior timetable.
- Balance of Exploration and Exploitation - gives a controlled method for balancing exploration and exploitation. It enables for the exploitation of improvements while still allowing for the investigation of potentially interesting movements by limiting the acceptance of moves to those that are better up to a particular threshold.

4.7 Algorithm

With the various SPS-SPHH components defined above. This is how they collaborate:

Algorithm 3 Algorithm for Selecting Low Level Heuristic

```
Input: problem instance
Output: feasible timetable
1: create an initial timetable using a constructive heuristic
2: store the initial timetable as a current timetable
2: for $count \leftarrow 1$ to $numOfGenerations$ do
2:     select a low-level heuristic from the list using the heuristic selection technique
2:     apply the low-level heuristic to the current timetable and store the new timetable as a temporary timetable
3: use the move acceptance method to see if the temporary timetable is accepted or not
4: if the temporary timetable is accepted then
5:     return update the current timetable to the temporary timetable
6: end if
6: end for
7: return current timetable
   =0
```

5 Results

5.1 comp04.ctt.txt

Seed	Hard Constraints Cost	Soft Constraints Cost	Execution Time (s)
5778856	0	36	348.3698
16223899	0	32	347.7916
22878896	0	32	350.6547
68625033	0	32	349.1685
101006424	0	36	353.5873
113139835	0	32	339.8837
125838671	0	32	346.8203
213232696	0	33	344.5567
311489368	0	34	343.0764
376423770	0	32	343.6810

Table 1: Constraint costs for problem instance comp04 for different runs with different seeds

5.2 comp08.ctt.txt

Seed	Hard Constraints Cost	Soft Constraints Cost	Execution Time (s)
5778856	0	31	461.2277
16223899	0	35	457.9260
22878896	0	33	459.8936
68625033	0	34	459.0906
101006424	0	33	458.9888
113139835	0	32	454.7071
125838671	0	34	452.2025
213232696	0	40	453.7278
311489368	0	33	457.3348
376423770	0	32	458.0043

Table 2: Constraint costs for problem instance comp08 for different runs with different seeds

5.3 comp09.ctt.txt

Seed	**Hard Constraints Cost**	**Soft Constraints Cost**	**Execution Time (s)**
5778856	0	35	379.0135
16223899	0	35	376.1171
22878896	0	35	374.3313
68625033	0	35	376.6497
101006424	0	35	377.2101
113139835	0	35	381.8481
125838671	0	35	372.6301
213232696	0	36	375.0068
311489368	0	35	376.9402
376423770	0	38	378.9523

Table 3: Constraint costs for problem instance comp09 for different runs with different seeds

5.4 comp12.ctt.txt

Seed	**Hard Constraints Cost**	**Soft Constraints Cost**	**Execution Time (s)**
5778856	0	42	759.4109
16223899	0	43	462.4363
22878896	0	42	469.1980
68625033	0	47	478.8111
101006424	0	42	442.0499
113139835	0	47	439.7605
125838671	0	43	449.1223
213232696	0	44	460.3338
311489368	0	43	459.0374
376423770	0	43	468.6426

Table 4: Constraint costs for problem instance comp09 for different runs with different seeds

5.5 comp13.ctt.txt

Seed	Hard Constraints Cost	Soft Constraints Cost	Execution Time (s)
5778856	0	35	1452.8921
16223899	0	35	2758.5491
22878896	0	35	2379.0245
68625033	0	34	1018.5747
101006424	0	33	1032.0647
113139835	0	33	1189.3324
125838671	0	33	1021.9081
213232696	0	34	1179.5452
311489368	0	34	1230.3120
376423770	0	34	1230.4430

Table 5: Constraint costs for problem instance comp13 for different runs with different seeds

5.6 comp18.ctt.txt

Seed	Hard Constraints Cost	Soft Constraints Cost	Execution Time (s)
5778856	0	52	275.5458
16223899	0	52	308.6065
22878896	0	55	320.8954
68625033	0	53	328.3469
101006424	0	54	288.8447
113139835	0	54	293.2312
125838671	0	54	293.8019
213232696	0	55	296.7530
311489368	0	57	293.0401
376423770	0	52	291.0489

Table 6: Constraint costs for problem instance comp18 for different runs with different seeds

5.7 comp21.ctt.txt

Seed	Hard Constraints Cost	Soft Constraints Cost	Execution Time (s)
5778856	0	61	1048.7025
16223899	0	61	1376.1707
22878896	0	61	1243.4223
68625033	0	60	1126.0839
101006424	0	59	1754.5429
113139835	0	61	2754.4116
125838671	0	62	2958.2137
213232696	0	59	1244.3661
311489368	0	62	975.2859
376423770	0	63	989.3356

Table 7: Constraint costs for different runs with different seeds

5.8 ToyProblem.ctt.txt

Seed	Hard Constraints Cost	Soft Constraints Cost	Execution Time (s)
5778856	0	0	0.0577
16223899	0	0	0.1049
22878896	0	0	0.0743
68625033	0	0	0.0552
101006424	0	0	0.0378
113139835	0	0	0.0353
125838671	0	0	0.0358
213232696	0	0	0.0359
311489368	0	0	0.0386
376423770	0	0	0.0400

Table 8: Constraint costs for problem instance TonyProblem for different runs with different seeds

5.9 Summary of Problem Instances

Since the all the timetables generated for the problem instances are feasible, all data related to the hard constraints has been omitted because the cost is always zero.

NB: SCC stands for Soft Constraints Cost

Problem	Best SCC	Avg. SCC	Std. SCC	Avg. Execution Time (s)
comp04	32	33.1	1.5780	346.759
comp08	32	33.7	2.3685	457.3103
comp09	35	35.4	0.9165	376.8699
comp12	42	43.6	1.8000	488.8802
comp13	33	34	0.7746	1449.2646
comp18	52	53.8	1.5362	299.0114
comp21	59	60.9	1.2206	1547.0535
ToyProblem	0	0	0	0.0516

Table 9: Statistics for the different problem instances

The table above shows the best, average and standard deviation of the soft constraints cost obtained for the different problem instances.

6 Discussion of Results

6.1 Comparison with Optimization Hub ITC 2007 Results

The SPS-SPHH received a lower cost for all of the problem instances on which it was run. In some cases, the difference was significant, although in others, the optimization hub problem instances cost were 10 percent more than the cost calculated using the hyper-heuristic.

6.2 Comparison with Prof. Pillay 2016 Research Paper

The SPS-SPHH received a lower cost for all of the problem instances on which it was run. The costs stated in the research paper were 90 percent more than the cost calculated using the hyper-heuristic.

6.3 Time Comparison

On average it did not take that much time to find a very good feasible timetable for most of the instances. It roughly took 6 minutes to find the most feasible / optimal timetable.

6.4 Standard Deviation

The low standard deviation values for the problem indicates that the performance of the hyper-heuristic is relatively consistent across different runs.

7 Conclusion

The effectiveness of the single-point search selection perturbative hyper-heuristic has been showcased in its ability to balance exploration and exploitation, allowing it to escape local optima and discover diverse solution regions. By dynamically choosing from a list of heuristics and incorporating perturbations, it can overcome stagnation and continue making progress even when faced with challenging optimization landscapes.

Based on the results obtained on the different problem instances on the hyper-heuristic and the fact that the hyper-heuristic significantly performed better than the results on optimization hub and the research paper, it is evident that the SPS-SPHH effective in solving real-world problems across various domains by intelligently combining various low-level heuristics and perturbation strategies.

8 Extra Notes

- Please refer to the README.md for instructions on how to run the hyper-heuristic
- Solutions of the different runs for the hyper-heuristic are in the solutions directory.
- The results of the hyper-heuristic were compared to the results from here: `https://opthub.uniud.it/problem/timetabling/edutt/ctt/cb-ctt-ud2`