Development and Evaluation of Solutions for an Instance of University Timetable Problem

Corneel den Hartogh (formerly member of 'Geen Gaten')
Universiteit van Amsterdam
corneeldenhartogh@gmail.com
Date: 22 december 2016

1. Introduction

In this paper I will try to solve an unsolvable problem with the help of heuristics. Firstly, I will introduce the case. Secondly, I will explain the methods I used. Thirdly, I will show the results. Fourthly and finally I will discuss my conclusions.

The case at hand, 'Lectures & Lesroosters'[1], is an instance of the university timetable program. There are 29 courses which have activities: Lectures, tutorials and practical sessions. For the lectures all students are in the same group, but tutorials and practical sessions have limited capacity and therefore multiple groups. This leads to a total of 129 activities.

These activities need to be spread over time-room slots. There are seven rooms available (with different capacity) and four time-slots a day. In addition, the largest room has a fifth timeslot. This leads to a total of 145 timeslots.

1.1 State Space

Rooms are allowed to be rostered in the same time-room slot, which leaves us with 145^{129} (≈ 6.55E+278) possible solutions. Even, if only valid rosters (i.e. never more than one activity in a timeslot) are taken into account, the state space remains $\frac{145!}{16!}$ (≈3.85E+238) .

Moreover, there is an additional element that needs to be taken into account: The students. The 609 students need to be spread over the tutorials and practical sessions of their courses. For all activities combined this would result in approximately 5.28E+676[2] possible solutions for student spreading alone.

[1] For a description of the case (dutch), see:
http://heuristieken.nl/wiki/index.php?title=Lectures_%26_Lesroosters
[2] For precise calculations see vakken_calculations.xlsx, or request them by author

1.2 Soft Constraints

With this amount of possible solutions, the questioning rises what makes a solution ‘good’. This depends on satisfying the five soft constraints:

1. 1000 points when all the activities have unique time-room slots
2. 20 bonus points for every course that is spread optimally over the week
3. 10 malus points for every course of χ activities that are rostered on χ-1 days, 20 malus points for χ-2 days and 30 for χ-3 days and so on
4. 1 malus point for every student that exceeds the room capacity in which the activity, that they are assigned to, takes place
5. 1 malus point for every time a student has more than one activity in a timeslot
6. 50 malus points when one of the aforementioned fifth timeslots is used

Since 22 of the 29 courses have more than one activity, the perfect score would be 1440. Whether this score can be reached is highly doubtful. In addition, due to the size of state space it is not possible to calculate the scores of all possible rosters. Therefore, we need methods to approximate the optimal solution.

2. Methods

With regard to rostering problems literature suggests iterative algorithms, with on the one hand sophisticated versions of the hillclimber (HC) algorithm (simulated annealing (SA) and tabu search), and on the other hand genetic algorithms (GAs). More recent approaches include algorithms like particle swarm optimization, artificial bee colony and ant colony optimization.

Since information and examples of the HC, SA and GAs were provided by supervisors and time was limited, I decided to focus on these methods. I will explain them one by one in the following sections. Additionally, I discuss the local search heuristics that I applied.

2.1 Hill climber

The HC algorithm is quite simple in nature. In pseudocode:

```
Create a valid solution in a random manner
Apply local search heuristics
Evaluate this solution
While Stop Criterion is not met:
       Swap the contents of two slots
       Apply local search heuristics
       Evaluate new solution
       Maintain the best solution
Endwhile
```

I will only create valid solutions at the start. The small change consists of swapping the contents between two randomly selected time-room slots. After every 1000 iterations, the algorithm will verify if an improvement has been made; If not, the stop criterion is satisfied. Please note that

no improvement after 1000 iterations does not mean that a (local) optimum is reached. It only indicates that the roster seems to be close to a local optimum.

2.2 Simulated Annealing

The most fundamental critique on HC is that it will get stuck in local optima, since it is impossible to accept a (small) deterioration. In order to remedy this problem, SA have been proposed. They Whether an SA algorithm accepts a deterioration depends on the size of deterioration and the *temperature*. Since SA is based on the cooling down in the process of creating artefacts out of metal or glass - which is called annealing, hence the name SA - the temperature is important variable. At the start it must be sufficiently high to allow for large deteriorations, but it cools down with every step. In the end, it the temperature is so low that no matter how small the deterioration, it will not be accepted (and the SA will behave like the above described HC). Several cooling down procedures are possible and discussed in literature (Nourani and Andresen, 1998). For this case I chose a logarithmic approach. In pseudocode:

```
Temperature = 100
Create a valid solution in a random manner
Apply local search heuristics
Evaluate this solution
While Stop Criterion is not met:
        Swap the contents of two slots
        Apply local search heuristics
        Evaluate new solution
        If score new solution - score old solution > 0:
                Maintain the new solution
        Else:
                Chance of acceptance = e^(delta/temperature)
                If chance of acceptance > random value between 0 and 1:
                        Maintain new solution
                Else:
                        Maintain old solution
        Temperature = 100 * 0,1^(iterations/10.0000)
Endwhile
```

The chance of acceptance is determined by the difference between the scores of the two solutions (this delta is necessarily negative) and the temperature. I choose this cooling down scheme after some preliminary testing with linear cooling down patterns and higher temperatures. This method delivered good results in a quick fashion.

2.3 Genetic Algorithms

Proposed by Holland (1992) genetic algorithms resemble the evolvement of species in the biological realm. A population of solutions is created and these solutions are combined in order to create novel solutions. After this process is completed, the population is updated. GAs are

characterized by a multitude of variables (for instance: number of population, selection method, amount of individuals selected, crossover method, mutation size, updating population method). For this case I adapted the following procedure:

```
Create a population of 20 valid solutions in random manner
Evaluate these solutions
While Stop Criterion is not met:
	Random selection of two individuals to recombine
	Application of crossover operator
	If necessary:
		Repair operator
	Else:
		Mutation operator
	Apply local search heuristics
	Add two resulting solutions to population
	Disregard the two solutions with lowest score
Endwhile
```

Based on Maenhout and Vanhoucke (2007) I decided for hybridization of crossover operators. While their research was into a different rostering problem (the Nurse Scheduling Problem) it inspired me to identify three crossover techniques: day-based, timeslot-based and subject-based. Since the score of days and timeslots is difficult to calculate (since the subject score is dependent on its spreading over a 5 days) I picked those randomly. The selection of subjects in the subject-based crossover was solely based on their subject-score. When an activity could not be added since this would result in an invalid roster, I tried to add the corresponding activity from the other parent. This ensured that the children really consisted of elements of their parents. The stop criterion was the same as with HC: After every 1000 iterations the best score is compared with the best score 1000 iterations before.

Due to time limitations I was unable to tune the earlier mentioned multitude of variables and scientifically compare the results. In addition, GAs are vulnerable to converge to soon, which results in getting stuck in local optima. In order to counter this threat, the score function itself is can be adapted to include a diversity dimension. Since I intended to compare the results from a GA with those of HC and SA, I decided against adapting the score function. Therefore, the results of the GA in the following section are by no means the optimal results that can be gathered with a GA in regard to this case.

2.4. Local search techniques

The local search heuristics employed are student optimization and room optimization. The student optimization evaluates conflicts in rosters for students and tries to solve these. It improved the scores drastically. The room optimization arranges the activities in a given timeslot

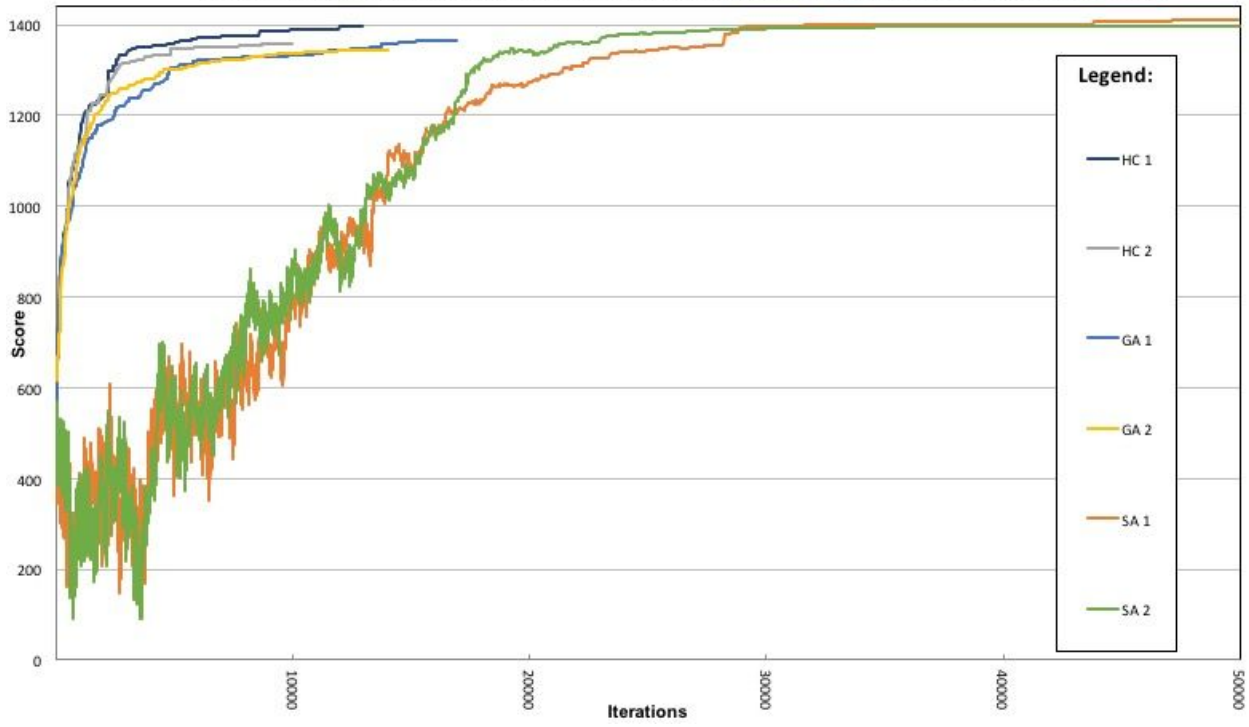
on the basis of their amount of students over the seven rooms. This optimization helped to bring down the number of iterations necessary to come close a local optimum.

3. Results

Before showing the results of the above described methods, I would like to provide the necessary context. As stated earlier the upper limit of the score function is 1440. The worst possible score, when including invalid rosters as well, is -6541.[3] Hence, the score range is 7981.. In addition, from 609 student, no less than 116 have four or more activities and none of them have same activities.[4] A perfect score seems therefore impossible.

In regard to the results, I will first show a two examples of the development of the score for each algorithm per iteration. Thereafter, I will present the scores of 70 runs per algorithm on two different computers and show their corresponding execution time. Thirdly, I will demonstrate the results of further optimization of the highest scoring rosters. Fourthly and finally, I will discuss the actual differences between the high scoring rosters.

Figure 1: Score development of algorithms, on the x-axis the number of iterations, on the y-axis the score



[3] For precise calculations see 'vakken_calculations.xlsx', or request them by author
[4] This is demonstrated in 'students_exploration.xlsx', in addition, it can be inferred from http://www.heuristieken.nl/resources/studenten_roostering.csv

3.1 Score Development

As is shown by Figure 1, after 30.000 iterations SA does not allow for deteriorations anymore. Therefore I set the stop criterion for SA to do at least 30.000 iterations and to verify after that at every 1000 iterations if there has been an improvement.

3.2 Final scores

From all algorithms two sets of 70 runs were selected randomly. Figures 2 and 3 (see below, computer 1 = Mac OS X 10.11.6, CPU 3 GHz Intel Core i7, computer 2 = Windows Server 2012, CPU 2.1 GHz Intel XEON E5-2620 v2) show that the scores are distributed very evenly per algorithm. The results of the figures combined demonstrate that while GA does not nearly have as much iterations as SA, it approaches its CPU time. This is due the fact that GA creates two rosters per iterations and has a complex crossover mechanism instead of simple swap. In addition, it is visible that the results of SA are all above 1360 (= >99% of perfect score).

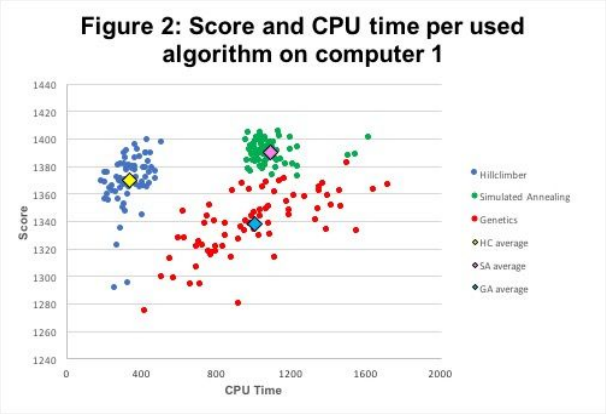


Figure 2: Score and CPU time per used algorithm on computer 1

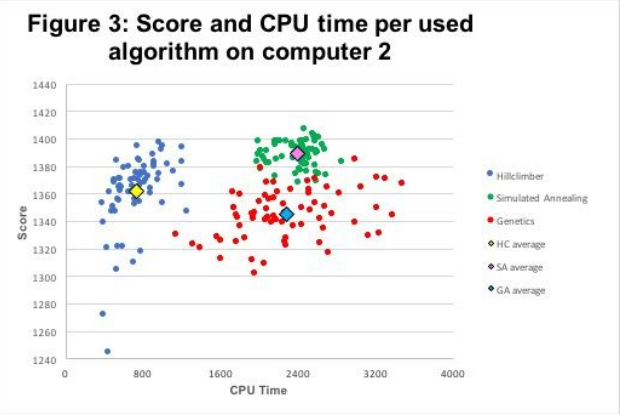


Figure 3: Score and CPU time per used algorithm on computer 2

3.3 Optimizing best rosters

I selected the rosters above 1400 (= >99.5% of perfect score) for further improvement. For those I conducted every possible slot swap. Those rosters improved from by approximately only 3.01.[5] However, the top score of 1411.00 was not improved.

3.4 Differences between the resulting rosters

Higher scoring rosters have, on average, better results on the spreading of subjects over the week.[6] However, some rosters above the 1400 are still not optimally spread, even after

[5] See difference in scores between imported_rosters and top_rosters or request them by author
[6] For calculations see 'activitySpread.xls', or request them by author

optimization.[7] In addition, the highest scoring roster has one case of overcapacity. With regard to students, there is only one student - with five courses - who has a conflict in all 20 high scoring rosters.[8] However, even for this student the conflict itself is always different. There is only one characteristic that holds for all solutions: None make use of the fifth timeslot of the large room. Besides that, the high scoring rosters not only differ from each other, but there imperfections are different as well.

4. Conclusion and Discussion

The state space of this type of problem is, due to its multitude of variables, very large. However, due to the soft constraints, good solutions are sparse. The SA algorithm was, empowered with local search techniques, able to generate several severely distinct, but equally strong, solutions. While a perfect score was not reached, strong solutions do not make use of the fifth timeslot of the large room and are (close to) optimally spreading the activities of the subjects over the week. Unfortunately, we were unable to prove that optimal spreading is necessary to reach a top score. In addition, avoiding overcapacity is not a necessity either. With regard to student optimization is must be pointed that the used optimization could be further developed and potentially lead to even higher scores.

This is interesting, since the results of the SA implementation, continuously providing solutions above 99%, are already impressive. While it is outside of the scope this paper, it does make you wonder where the limits of SA lie. How complex does a problem instance needs to be, before more complex solutions (GAs, particle swarm optimization and others) outperform a relatively simple SA implementation?

5. Acknowledgements

Due to unfortunate personal circumstances I was unable to participate fully in the ‘Programmertheorie’ course. However, thanks to exceptions made by Daan van den Berg and Jelle Assema I was enabled develop my ideas and present my findings.

In addition, I want to thank Luitzen de Vries. Running algorithms takes time and the availability of his server for this task was appreciated. This provided me early on with the confidence that I would be able to gather enough data for proper analysis.

[7] See rosters in visuals/visual.json, access rosters.html in browser, or request them by author
[8] Andre Putters, run exportIssues.py to see the number of occurences per conflict

6. References

Holland, J. H. (1992). Genetic algorithms. *Scientific american*, *267*(1), 66-72.

Maenhout, B., & Vanhoucke, M. (2007). A Comparison and Hybridization of Crossover Operators for the Nurse Scheduling Problem. *Journal of Entrepreneurship*, *16*(2), 147-171.

Nourani, Y., & Andresen, B. (1998). A comparison of simulated annealing cooling strategies. *Journal of Physics A: Mathematical and General*, *31*(41), 8373.