AI for Exam Allocation

William Ke

December 21, 2020

Abstract

The main objective of this project is to design and implement an algorithm that is able to distribute the students in *TTK4555 Specialization Course* into 4 time periods available for examination. We were able to formulate this problem as a variant of the exam timetabling problem, with the necessary constraint represented using an objective function. Using an implementation of the hill climbing method, we were able to solve for a feasible solution.

Sammendrag

Hovedmålet med dette prosjektet er å designe og implementere en algoritme som er i stand til å distribuere studentene i *TTK4555 Teknisk kybernetikk, fordypningsemne* til 4 tidsperioder tilgjengelig for eksaminering. Vi var i stand til å formulere dette problemet som en variant av "exam timetabling" problemet, med de nødvendige begrensningene representert ved hjelp av en kostfunksjon. Gjennom en implementasjon av "hill climbing" metoden kunne vi løse for en tilstrekkelig god løsning.

Preface

This specialisation project marks the beginning of the end of my Master's degree in Cybernetics and Robotics at the Norwegian University of Science and Technology (NTNU). The project is written under the supervision and guidance of Professor Ole Morten Aamo, who did a great job assisting me from beginning to end. Moreover, I would like to thank all my friends and family for supporting me throughout these years.

The presented algorithm was implemented in MathWorks' MATLAB, and the input data was imported from Microsoft Office Excel sheets. The relevant theory necessary for understanding the project will be presented, but the reader is expected to be familiar with the fundamentals of mathematical optimisation and computer programming.

William Ke
Trondheim, 21st December 2020

Contents

Figures

Chapter 1

Introduction

1.1 Background and Motivation

The general timetabling problem is a well researched issue that appears in numerous forms, including educational timetabling [1], nurse scheduling [2], transportation timetabling[3, Chapter 51], and sports timetabling [3, Chapter 52]. These kinds of problems have been an important research area in both Operations research and Artificial intelligence. *Timetabling* and *scheduling* have been used interchangeably from what we have found in our literature review, It's largely been up the authors personal preference.

Burke et al. [1] gives a definition for the general timetabling problem, which covers most of the forms mentioned above:

> A timetabling problem is a problem with four parameters: T, a finite set of times; R, a finite set of resources; M, a finite set of meetings; and C, a finite set of constraints. The problem is to assign times and resources to the meetings so as to satisfy the constraints as far as possible.

Educational timetabling is one of the most important administrative task which is done periodically in academic institutions. The timetable must account for the needs of a broad range of different stakeholders such as administrators, lecturers, and students; as such, there exists a large number of variants for the timetabling problem in literature, since the exact needs and constraints will be different for different institutions. Multiple commercial software solutions have been developed to relieve institutions of developing timetabling software themselves, but they can be quite costly. Examples of providers for these commercial solutions include: EventMAP [4], Destiny Solutions [5], and Scientia [6]

Variants of educational timetabling include class-teacher scheduling, course timetabling, exam timetabling [7], faculty timetabling, and classroom assignment [8]. Some specific problems might fall in between these categories, as the distinction between categories are not clearly defined.

Exam timetabling can be described as the scheduling of exams for a set of university courses, while minimizing overlap of exams with common students, and spreading the exams for students as much as possible. We will focus on exam timetabling as the task we are trying to solve in this project can be seen as a subvariant within exam timetabling.

The motivation for this project is to automate the process of creating an exam timetable for the course *TTK4555 Specialization Course* at The Department of Engineering Cybernetics at NTNU. The process is currently done manually by having a person create an exam timetable and checking for conflicts. Automating the timetabling allows the administrator to spend their time on other tasks, it is much faster than manual timetabling, and it reduces the probability of conflicts.

1.2 Objective

The main objective of this specialization project, as paraphrased from my supervisor, is to develop an algorithm that automatically distributes the students in *TTK4555 Specialization Course* into 4 time periods available for examination. The algorithm should take these constraint into consideration:

- Avoid assigning a student to multiple exams in the same time slot
- Preferably avoid conflict with exams from other courses
- Preferably avoid assigning multiple exams to the same day.

The first constraint is a hard constraint and should be avoided at all costs. The second constraint is a soft constraint that should preferably be avoided. The third constraint is also a soft constraint but has low importance. The first task is to formulate the problem as an exam timetabling problem. The next task is to design and implement an algorithm that takes into consideration the constraints described above. It is acceptable that the solution is sub-optimal, as long as the one hard constraint is satisfied, i.e., Avoid assigning a student to multiple exams in the same time slot.

1.3 Contributions

We propose a formulation of the optimisation problem. In addition to an implementation of an algorithm that considers the soft and hard constraints, and that is able to return a feasible solution to the optimisation problem. Lastly, we will give a qualitative and quantitative evaluation of our results.

1.4 Thesis Structure

Chapter 2 will present all the necessary theory for understanding and solving the main objective of this project. Chapter 3 is where we propose a formulation of the main objective in the form of a mathematical optimisation problem. The latter part of the chapter will be for presenting a data model that the algorithm will use as input, as well as the design and implementation of the algorithm itself. Chapter 4 presents the results and discussion from the output of our algorithm. Finally, Chapter 5 will give a conclusion of our work and results.

Chapter 2

Theory

This chapter is for giving an overview of the necessary theory that is relevant for solving the project. The first section will be briefly about Mathematical optimisation. All the subsequent sections will be about the exam timetabling problem. Schaerf [7], Gashgari et al. [9], and Qu et al. [10] have made three excellent survey papers that gives an general overview of the exam timetabling problem and solution approaches for this problem. Therefore, a majority of the theory presented in this chapter will be paraphrased in some form from these three survey papers.

2.1 Timetabling problem

Exam timetabling problems can be defined as assigning a set of exams $E = e_1, e_2, \ldots, e_e$ into a limited number of ordered timeslots $T = t_1, t_2, \ldots, t_t$, subject to a set of constraints $C = c_1, c_2, \ldots, c_c$ [10]. The timetabling problem can be either formulated as a *search problem* or as an *optimisation problem*. The former case consists of finding any timetable that satisfies all the constraints, while the latter consists of finding a timetable that satisfies all the *hard constraints* and minimises an *objective function* that represents the *soft constraints*

A solution that satisfies all the hard constraints are said to be *feasible*. *Soft constraints* are those constraints that are desirable, and is often used to measure the quality of a solution. The soft constraints may contradict each other in most practical cases and are therefore often impossible to fully satisfy.

Qu et al.[10] presents the hard and soft constraints that most commonly occur in exam timetabling literature.

Common Hard Constraints:

1. No exams with common students assigned simultaneously
2. Resources for exams needs to be sufficient (i.e size of exams must not exceed room capacity; enough room must exist for all exams.)

Common Soft Constraints:

1. Spread exams as evenly as possible, or not in consecutive time slots or days
2. Schedule all exams, or largest exams, as early as possible
3. Ordering (precedence) of exams need to be satisfied
4. Time requirements (e.g., certain exams should (not) to be in certain time slots)
5. Conflicting exams on the same day to be located nearby
6. Exams may be split into different rooms over similar locations
7. Only exams of the same length can be combined into the same room
8. Facility requirements

The needs for different institutions may not be the same. Consequently, exam timetabling problems might not contain all the constraints listed above and/or have constraints that are not listed. In addition, the weighting of the constraints might be different in the objective function.

The computational complexity of the exam timetabling problem is NP-complete in most practical cases [7]. Exact solutions are therefore only possible for simple cases (e.g., less than 10 exams). Real cases might involve hundreds of courses and students. Consequently, only heuristic methods are feasible for solving real cases [11].

2.2 Solution approaches/techniques

This section will give an overview of some of the approaches for solving the exam timetabling problem as described in the previous section. Figure 2.1 shows how some the methods that will be presented might be categorised.

2.2.1 Local Search Based Techniques

Local search based techniques [12], such as Hill Climbing, Tabu Search and simulated annealing are commonly seen as belonging to metaheuristics [13]. Metaheuristics are higher level heuristic methods that are designed to provide a suf-

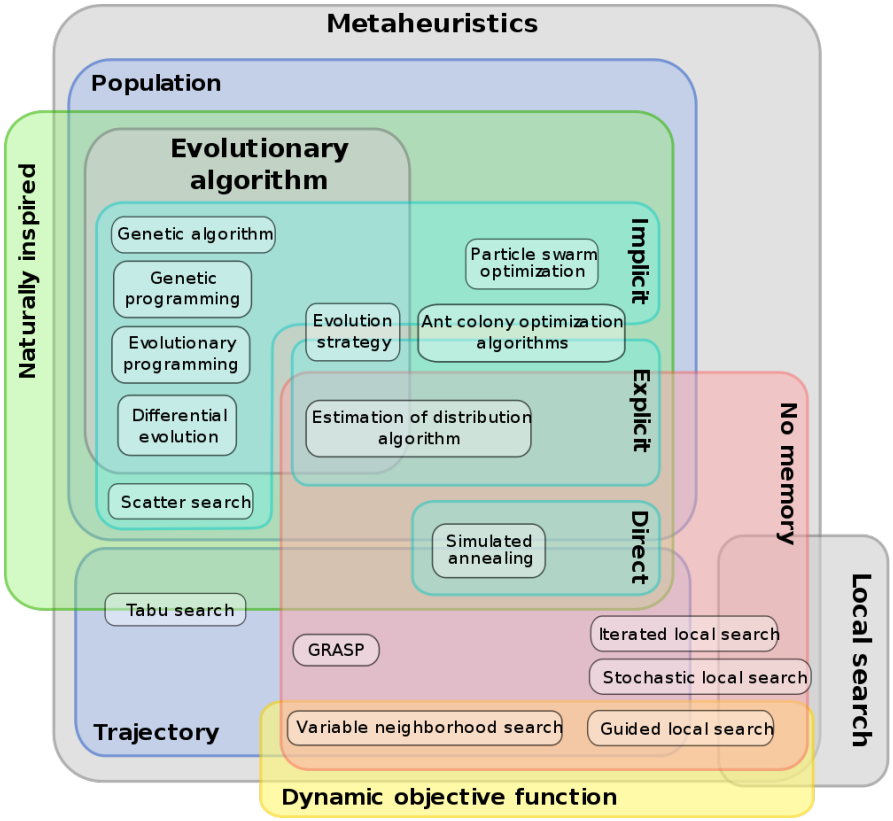

Figure 2.1: Metaheuristics classification

Source: https://commons.wikimedia.org/wiki/File:Metaheuristics_classification.svg.

ficiently good solution to an optimisation problem. It is most commonly used in cases with incomplete information or limited computational power [14]. Most literature on metaheuristics describe empirical result that are based on computer experiments. We have no mathematical proofs to guarantee optimal solutions, but the solutions that metaheuristic methods give are often sufficient in practice.

Local search methods solve problems by searching from an incumbent solution to its neighbourhood, hence the name local search. Local search methods differ in the way they define the neighbourhood and moving operators. An objective function is used to measure the quality of the resulting solutions.

Hill Climbing

Hill climbing is one of the simplest local search methods. It is an iterative algorithm that starts with an arbitrary initial solution and attempts to change to a better solution among the solutions in its neighbourhood. This will continue un-

til no further improvements can be found. Hill climbing is well suited for solving convex problems, as a locally optimal solution is also a global optimal solution. Hill climbing will plateau on local optima for non-convex problems. Variants of hill climbing include simple hill climbing, and steepest ascent hill climbing. The former is when the first better solution is chosen, while the latter is when the best solution in the neighbourhood is chosen.

Tabu search

Tabu search [15] can be seen as a modification to the hill climbing algorithm. The first modification is that the inferior solution might be selected if no better solutions are available. The second modification is done by disallowing revisiting a list of recent moves. These moves are kept in a *tabu list*, hence the name. The first modification makes the method able to escape local optima, while the second modification is to ensure that the method does not flip back and forth between the same two solutions. One disadvantage with the tabu search method is that the parameters need to be fine-tuned to the specific problem at hand in order for the method to perform well. Examples of these parameters include the size of the tabu list and the stopping conditions.

Simulated annealing

Simulated annealing [16] is a method that is inspired from the natural annealing process. It is a probabilistic local search technique that approximates a global optimum for objective functions with many local optima. The idea of the method is to search a wider area of the search space at the beginning of the algorithm by accepting worse solutions with a higher probability. The probability of accepting worse solution goes down as the search progresses. The parameters that can be tuned include: start and end probability, and the reduction rate of this probability. The motivation for choosing worse solutions is the same as in tabu search, which is to avoid getting stuck in local optima.

Summary of Local Search Based Techniques

Hill climbing is the simplest algorithm to implement, but it has the disadvantage of getting stuck in local optima for non-convex problems i.e., most practical cases. Tabu search and simulated annealing can be seen as modified hill climbing algorithm whereby accepting worse moves might aid in escaping local optima. One common drawback with Tabu search and simulated annealing is that they require tuning their associated parameters for the specific problem at hand in order to get

high quality solutions.

2.2.2 Population Based Algorithms

Genetic Algorithms, Memetic Algorithms, and Ant Algorithms are included in a family of population based algorithms known as Evolutionary algorithms. They are commonly characterized by their common inspiration from natural phenomena and behaviours.

Genetic Algorithms

Genetic Algorithms [17] simulated the natural evolutionary process by evolving and manipulating populations of solutions withing the search space. Solutions are encoded as *choromosones*(array of bits) and are evolved over multiple generations using crossover and mutation operators with the aim of incrementally getting better solutions. The parameters and operators of the algorithm needs to be properly defined. Consequently, the approach is usually more complicated to implement that local search based methods. The search strategy is fundamentally different from local search based methods in that multiple solution is managed simultaneously, instead of just a single solution as seen in local search.

Memetic Algorithms

Memetic algorithms [18] aims to improve upon genetic algorithms by combining them with local search methods. This works by using local search methods on individual solutions of a population in between generations. Memetic algorithms are able to combine the benefit of exploring large search space from population based algorithms with the benefit of improving individual solutions in a population in a slightly less random way from local search based methods. There are however some drawbacks with Memetic algorithms. The first drawback is that it increases the computational cost of the algorithm. The second drawback is that this introduces several more design parameters that need to be fine-tuned, including: how often local search should be applied, which solutions should local search be used on, how long should local search be run, and which local search algorithm should be used?

Ant Algorithms

Ant algorithms [19], as the name implies, take inspiration from the way ants behave. It aims to simulate the way ants search for the shortest route to food by using pheromones along the way.

Ants of some species will initially wander randomly in the natural world. When they find food, they will return to the colony while placing down pheromone trails along the path. Other ants are inclined to follow the pheromone path instead of wandering randomly, and they will in turn reinforce the pheromone trail if they also find food.

However, these pheromone trails do not last indefinitely and will over time evaporate and become weaker. Longer trails will have more time to evaporate than shorter trails. This results in the shortest trails having stronger pheromone trails than other paths over a period of time.

Summary of population based Algorithms

The ability to evaluate multiple solutions simultaneously is a huge benefit with population based algorithms. Each generation of population based algorithms has a larger computational cost associated with it compared to a single iteration in local search based techniques. The benefits of using population based algorithms are therefore not overly apparent, at least with respect to total computation time.

Chapter 3

Design & Implementation

This chapter is for presenting the problem formulation as it pertains for our specific problem. We propose in addition an objective function that considers all the necessary constraints.

3.1 Problem definition

All Master students in Cybernetics and Robotics are required to take *TTK4555 Specialization Course* [20] [21]. Each student is to take either two specialization subjects à 3,75 credits, or one à 7,5 credits. We will only be assigning the students that take the 3.75 credit subjects, so we can ignore those who take one 7.5 credit subject. The evaluation form of the subjects with 3.75 credits are oral examinations. The oral examinations will have a duration of 30 minutes each. As such, there is a limit to how many examinations a single sensor can oversee each day. The whole course is allocated two examination days with two time slots each (one for morning and one for afternoon), for a total of four time slots. Each specialization subjects à 3,75 credits will have a predetermined number of students that should be assigned into each time slot according to these rules:

- Subjects with less than 6 students have only one time slot on day 1
- Subjects with between 6 and 16 students have two time slots on day 1
- Subject with more than 16 students have two time slots each on day 1 and day 2, for a total of four time slots.

Our exam timetabling problem can be defined as assigning a set of students $S = s_1, s_2, \ldots, S_s$ into two exams $E = e_1, e_2$. Each exam has a predetermined amount of students that should be assigned into 4 time slots $T = F1, E1, F2, E2$ ($F1, E1, F2, E2$

corresponds to "formiddag, day 1", "ettermiddag, day 1", "formiddag, day 2", "ettermiddag, day 2" respectively.), according to a set of constraints C

The set of constraints C for our problem is the following:

Hard constrains, of which we only have one:

- Avoid assigning a student to multiple exams in the same time slot

Soft constraints, of which we have two:

- Preferably avoid conflicts with exams from other courses
- Preferably Avoid assigning multiple exams to the same day.

There is a "core" list of elective courses that Master students in Cybernetics and Robotics can take. These electives are guaranteed to not collide with obligatory courses and can be freely taken. As such, avoiding conflict with other exams is a soft constraint, because it is ultimately the students' responsibility that they choose elective courses that do not have conflicting exam dates.

3.2 Data model

3.3 Solution data format

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Navn	Tema 1	Tema 2	Tema1F1	Tema1E1	Tema1F2	Tema1E2	Tema2F1	Tema2E1	Tema2F2	Tema2E2	Blokk1	Blokk2
2	[redacted]	TTK2 Design og an	TTK1 Innovasjon:	0	0	1	0	0	0	0	1	0	0
3		TTK26 Biomedisins	TTK1 Innovasjon:	0	1	0	0	0	0	0	1	0	0
4		TTK8 Konstruksjon	TTK1 Innovasjon:	0	0	1	0	0	0	0	1	1	0
5		TTK8 Konstruksjon	TTK1 Innovasjon:	0	1	0	0	0	0	0	1	0	0
6		TTK10 Signal proce	TTK13 Ultrasounc	0	0	1	0	0	0	0	1	0	0
7		TTK10 Signal proce	TTK13 Ultrasounc	0	0	1	0	0	0	0	1	0	0
8		TTK10 Signal proce	TTK13 Ultrasounc	0	0	1	0	0	0	0	1	0	0
9		TTK10 Signal proce	TTK13 Ultrasounc	0	0	1	0	0	0	0	1	0	0
10		TTK10 Signal proce	TTK13 Ultrasounc	0	0	1	0	0	0	0	1	0	0

Figure 3.1: a screenshot of example data

A screenshot of the example data format can be seen in Figure 3.1. The names of the students have been erased to preserve anonymity. Each row is one student, and each student has 13 columns associated with them. The name column is used as a unique identifier for the table. "tema 1" and "tema 2" are the two subjects that each student has chosen. One-hot encoding has been used to represent which time slot the corresponding exam has. This was done to simplify the implementation of objective function and switching algorithm. The last two columns "Blokk 1" and

"Blokk 2" are used to signify if the student has a collision on a specific day with an exam from a different course. "Blokk 1" is used to signify a collision on day 1, and "Blokk 2" is used to signify a collision on day 2.

3.4 Objective function

The objective function that we will use to evaluate the quality of our solution will be as follows:

$$Z = \sum w_i g_i(X) \quad \text{for} \quad i \in 1, 2, 3 \tag{3.1}$$

w_i is the weighting coefficient corresponding to the i^{th} variable.

$g_i(X)$ are the constraints for our problem expressed as a function of the decision variables X, where X are all the possible time slots for each student. To keep the implementation simple, we will represent both our hard and soft constraints using the objective function. The only difference is a considerably larger weighting coefficient w_i for the hard constraints.

$g_1(X)$, $g_2(X)$, and $g_3(X)$ corresponds to the constraints described in Section 3.1 with the same order of appearance.

The exam timetable problem can then be represented as the following optimization problem

$$\min_x = Z(x) \tag{3.2}$$

3.5 The chosen algorithm

The Steepest-Ascent hill climbing method was chosen to be implemented first as it was the simplest to implement, and we could modify the hill climbing method into Tabu search or Simulated annealing if it later proved to be necessary. We found out during preliminary testing of the algorithm that the algorithm was sufficiently able to produce a feasible solution for the problem as described above. The main objective of this project was only to develop an algorithm that would give a feasible solution, and not a globally optimal solution. We decided that it was not necessary to implement and test the other algorithms outlined in Chapter 2, as we had already accomplished the main objective of the project.

The pseudocode for the hill climbing method is as follows:

Algorithm 1: Steepest-Ascent hill climbing method

```
Result: Sol_Best A solution table that is a local optima of the exam
        timetable problem
Sol_Best ← read(excelSheet);
Candidate_Best ← read(excelSheet);
while not stoppingConditions() do
    Sol_neighbors ← getneighborsof(Candidate_best);
    for neighbor ∈ Sol_neighbors do
        if computecost(neighbor) < computecost(Candidate_Best)
         then
            Candidate_Best ← neighbor;
        else
    if computecost(Candidate_Best) < computecost(Sol_Best) then
        Sol_Best ← Candidate_Best;
    else
```

The algorithm first reads the Excel sheet with the data model as described in Section 3.2. Then next it runs the $getneighborsof()$ function that returns a list of solutions that lies in the neighbourhood of the current candidate solution.

The $getneighborsof()$ first finds the student(row) that contributes most to the objective function. Next it identifies all the possible swaps in time slots between the costliest student and other students. Then it returns a list of solutions that is one swap away from starting solution, which is defined as the neighbourhood for our problem. A swap is only possible if the students have a common subject. The reason we do a swap is to conserve the number of students in each time slot since those are predetermined.

The algorithm then computes the value of the objective function for all the solutions in the neighborhood and selects the solution with the lowest value, this is designated as the best candidate ($Candidate\_Best$). The best candidate then becomes the best solution (Sol_Best) if it has a lower cost in the objective function.

The algorithm continues to iterate until there is no improvement in cost, or ideally if the cost reaches a value of zero.

Chapter 4

Results and Discussion

This chapter is for presenting the results and discussion using the model that was developed in Chapter 3. The weighting coefficient for multiple exams on the same time w_1 slot was set to be 10000, while the weighting coefficient for conflict with exams from other courses w_2 was set to be 1000. Lastly, the weighting coefficient for multiple exams in the same day w_3 was set to 10. This can be interpreted as the cost of the objective increasing by the respective amounts for each occurrence of the constraints for a student. e.g., if two student multiple exams on the same time slot and one student have a conflict with an exam from another course will result in the objective function returning a value of 5000.

The hill climbing algorithm was tested on real data consisting of 166 students from fourth year master's student studying Cybernetics and robotics at NTNU. The criteria that we will use to evaluate the algorithm will be as follows:

- The number of iterations of the algorithm
- The final value of the objective function
- How fast the objective value decreases as a function on the number of iterations.

4.1 Case 1

Figure 4.1 Shows value of objective function and occurrences of each constraint as a function of iterations. 18 of the students had exams from other courses that may conflict with one of the examination days. The starting cost for this case was 120860. The initial number of occurrences for constraint 1 g_1, constraint 2 g_2, and constraint 3 g_3 was 10, 0, and 86, respectively. The algorithm took 9 main loop iterations and 75.21 seconds to finish. The resulting solution after running

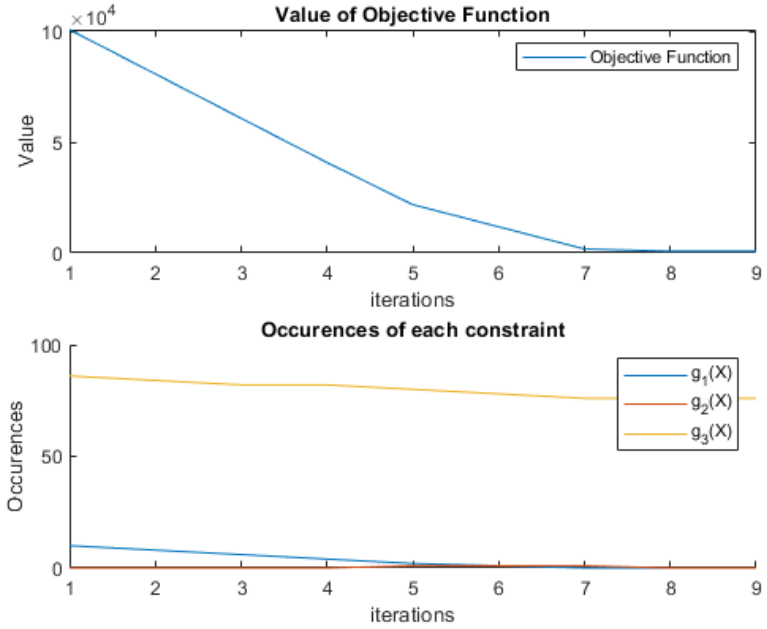

Figure 4.1: Plot of case 1 showing value of the objective function and occurrences of each constraint as a function of iterations

the hill climbing had a objective cost of 760. The final number of occurrences for constraint 1 g_1, constraint 2 g_2, and constraint 3 g_3 was 0, 0, and 76, respectively. The most numerous constraint was constraint 3 g_3. This reason for this is that there are some subjects with few students. This consequently results in that these subjects are assigned few time slots. The algorithm is therefore unable to find time slots that can fully satisfy constraint 3 g_3. The two most important constraints were able to be fully satisfied.

4.2 Case 2

Figure 4.2 Shows value of objective function and occurrences of each constraint as a function of iterations. Case 2 represents a more difficult, and somewhat unrealistic case. Random swaps were performed to generate a new initial solution. In addition, this time 62 of the students had exams from other courses that may conflict with one of the examination days. Historical data shows that about 10% to 15% percent of the students have conflicts with exams from other courses, from year to year. The starting cost for this case was 450970. The initial number of occurrences for constraint 1 g_1, constraint 2 g_2, and constraint 3 g_3 was 40, 50, and 97, respectively. The algorithm took 29 main loop iterations and 706.89 seconds to finish. The resulting solution after running the hill climbing had a objective

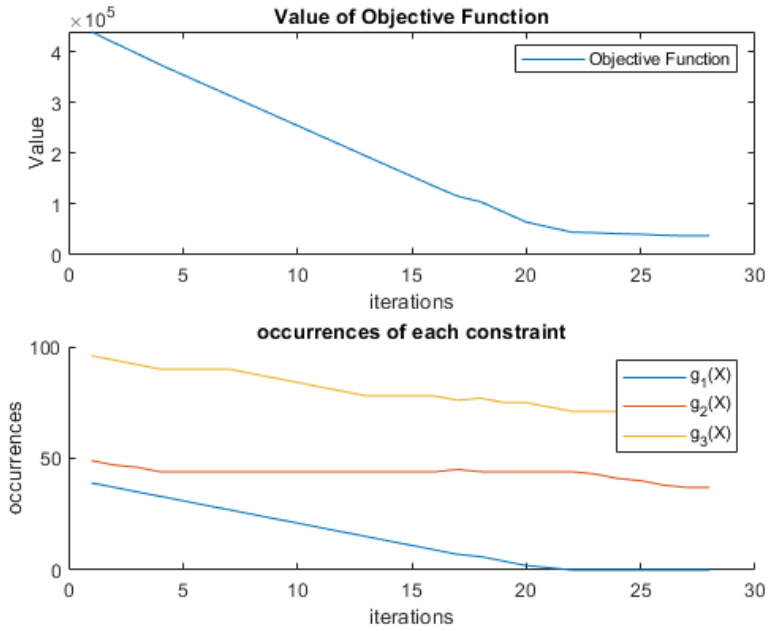

Figure 4.2: Plot of case 2 showing value of the objective function and occurrences of each constraint as a function of iterations

cost of 37670. The final number of occurrences for constraint 1 g_1, constraint 2 g_2, and constraint 3 g_3 was 0, 37, and 67, respectively. The one hard constraint we had was still able to be fully satisfied. In contrast, the soft constraints were unable to be fully satisfied. One drawback with the algorithm is that the running time is quite long when the initial objective function is high. This can be partially explained by that it takes some time to evaluate the objective function on the all the neighbourhood solutions to find the best one. This is however not that important for the practical functionality of the algorithm as the exam timetabling for *TTK4555 Specialization Course* is something that is done once a year.

Chapter 5

Conclusion

In the present work, a short overview of the theory and solution approaches behind exam timetabling was provided. We managed to formulate the exam timetable problem as an optimisation problem. Furthermore, we designed and implemented the Steepest-Ascent hill climbing algorithm to solve our problem.

The results showed that our algorithm was able to give solutions that fully satisfied the hard constraint in our problem. In contrast, the soft constraints were unable to be fully satisfied. The main objective was still fulfilled however, as it was acceptable that the solution were sub-optimal as long the hard constraint was satisfied. This shows that even the most simple method presented in Chapter 2 were able to solve our problem. It must be noted however that our specific exam timetabling problem is a simpler version compared to the other exam timetabling problems that exist in literature. The number of students we had to account for belonged only to one program of study, compared to a whole university. moreover, the number of exams we had to assign was limited. We only had to assign two exams on four different time slots across two days.

5.1 Further Work

We managed to accomplish the main objective of the specialization project. There are however some room for further work. One may try to implement some of the more advanced methods presented in Chapter 2. They may be able to further reduce the objective function of out solutions, compared to the hill climbing algorithm.

Bibliography

[1] E. Burke, D. de Werra, J. Kingston, J. L. Gross and J. Yellen, 'Applications to Timetabling,' *ResearchGate*, Jan. 2004. DOI: 10.1201/b16132-33.

[2] E. K. Burke, P. De Causmaecker, G. V. Berghe and H. Van Landeghem, 'The State of the Art of Nurse Rostering,' *J. Scheduling*, vol. 7, no. 6, pp. 441–499, Nov. 2004, ISSN: 1094-6136. DOI: 10.1023/B:JOSH.0000046076.75950.0b.

[3] J. Y.-T. Leung, *Handbook of Scheduling:Algorithms, Models, and Performance Analysis*. Andover, England, UK: Taylor & Francis, Apr. 2004, ISBN: 978-0-42920564-4. DOI: 10.1201/9780203489802.

[4] *EXAM: powerful exam scheduling software | EventMAP*, [Online; accessed 21. Dec. 2020], Aug. 2019. [Online]. Available: https://www.eventmapsolutions.com/exam.

[5] O. +. D. Solutions®, *Exam Scheduling Software | Destiny Solutions*, [Online; accessed 21. Dec. 2020], Dec. 2020. [Online]. Available: https://www.destinysolutions.com/products/proctor-and-exam-scheduling.

[6] *Exam Scheduler- Scientia | Graphical Scheduling Solution*, [Online; accessed 21. Dec. 2020], Dec. 2020. [Online]. Available: https://www.scientia.com/product/exam-scheduler.

[7] A. Schaerf, 'A Survey of Automated Timetabling,' *Artificial Intelligence Review*, vol. 13, no. 2, Jan. 1996, ISSN: 0269-2821. DOI: 10.1023/A:1006576209967.

[8] V. A. Bardadym, 'Computer-aided school and university timetabling: The new wave,' in *Practice and Theory of Automated Timetabling*, Berlin, Germany: Springer, Jun. 2005, pp. 22–45, ISBN: 978-3-540-61794-5. DOI: 10.1007/3-540-61794-9_50.

[9] R. Gashgari, L. Alhashimi, R. Obaid, T. Palaniswamy, L. Aljawi and A. Alamoudi, 'A Survey on Exam Scheduling Techniques,' *2018 1st International Conference on Computer Applications & Information Security (ICCAIS)*, pp. 1–5, Apr. 2018. DOI: 10.1109/CAIS.2018.8441950.

[10] R. Qu, E. K. Burke, B. McCollum, L. T. G. Merlot and S. Y. Lee, 'A Survey of Search Methodologies and Automated Approaches for Examination Timetabling,' *Journal of Scheduling - SCHEDULING*, Jan. 2006, ISSN: 1094-6136.

[11] J. Pearl, *Heuristics: Intelligent Search Strategies for Computer Problem Solving (The Addison-Wesley series in artificial intelligence)*. Boston, MA, USA: Addison-Wesley, Apr. 1984, ISBN: 978-0-20105594-8.

[12] M. Pirlot, 'General local search methods,' *Eur. J. Oper. Res.*, vol. 92, no. 3, pp. 493–511, Aug. 1996, ISSN: 0377-2217. DOI: 10.1016/0377-2217(96)00007-0.

[13] M. Gendreau and J.-Y. Potvin, *Handbook of Metaheuristics*. Boston, MA, USA: Springer, Boston, MA, 2010, ISBN: 978-1-4419-1663-1. DOI: 10.1007/978-1-4419-1665-5.

[14] L. Bianchi, M. Dorigo, L. M. Gambardella and W. J. Gutjahr, 'A survey on metaheuristics for stochastic combinatorial optimization,' *Nat. Comput.*, vol. 8, no. 2, pp. 239–287, Jun. 2009, ISSN: 1572-9796. DOI: 10.1007/s11047-008-9098-4.

[15] M. Gendreau and J.-Y. Potvin, 'Tabu Search,' in *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, Boston, MA, USA: Springer, Boston, MA, 2005, pp. 165–186, ISBN: 978-0-387-23460-1. DOI: 10.1007/0-387-28356-0_6.

[16] E. Aarts, J. Korst and W. Michiels, 'Simulated Annealing,' in *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, Boston, MA, USA: Springer, Boston, MA, 2005, pp. 187–210, ISBN: 978-0-387-23460-1. DOI: 10.1007/0-387-28356-0_7.

[17] K. Sastry, D. Goldberg and G. Kendall, 'Genetic Algorithms,' in *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, Boston, MA, USA: Springer, Boston, MA, 2005, pp. 97–125, ISBN: 978-0-387-23460-1. DOI: 10.1007/0-387-28356-0_4.

[18] P. Moscato, L. Plata and M. G. Norman, 'A "Memetic" Approach for the Traveling Salesman Problem Implementation of a Computational Ecology for Combinatorial Optimization on Message-Passing Systems,' *ResearchGate*, vol. 1, Jul. 1999.

[19] M. Dorigo and C. Blum, 'Ant colony optimization theory: A survey,' *Theoret. Comput. Sci.*, vol. 344, no. 2, pp. 243–278, Nov. 2005, ISSN: 0304-3975. DOI: 10.1016/j.tcs.2005.05.020.

[20] *Course - Engineering Cybernetics, Specialization Course - TTK4555 - NTNU*, [Online; accessed 17. Dec. 2020], Dec. 2020. [Online]. Available: https://www.ntnu.edu/studies/courses/TTK4555#tab=omEmnet.

[21] *Fordypning høsten 2020*, [Online; accessed 17. Dec. 2020], Dec. 2020. [Online]. Available: https://www.itk.ntnu.no/emner/fordypning.