DEGREE PROJECT IN TECHNOLOGY,
FIRST CYCLE, 15 CREDITS
STOCKHOLM, SWEDEN 2016

A comparative study between a simulated annealing and a genetic algorithm for solving a university timetabling problem

JONAS DAHL

RASMUS FREDRIKSON

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF COMPUTER SCIENCE AND COMMUNICATION

KTH Computer Science
and Communication

A comparative study between a simulated annealing and a genetic algorithm for solving a university timetabling problem

En jämförande studie mellan en algoritm baserad på simulerad glödgning och en genetisk algoritm för att lösa ett universitetsschemaläggningsproblem

JONAS DAHL
RASMUS FREDRIKSON

Degree Project in Computer Science, DD143X
Supervisor: Dilian Gurov
Examiner: Örjan Ekeberg

CSC, KTH. Stockholm, Sweden. May 11, 2016.

Abstract

The university timetabling problem is an NP-complete problem which schools all over the world face every semester. The aim of the problem is to schedule sets of events such as lectures and seminars into certain time slots without violating numerous specified constraints. This study aimed to automate this process with the help of simulated annealing and compare the results with a genetic algorithm.

The input data sets were inspired by the Royal Institute of Technology in Stockholm. The results showed a great run time difference between the two algorithms where the simulated annealing performed much better. They also showed that even though the simulated annealing algorithm was better during all stages, the genetic algorithm had a much better performance in early stages than it had in latter. This led to the conclusion that a more optimized, hybrid algorithm could be created from the two algorithms provided that the genetic algorithm could benefit from the improvements suggested in previous research.

Sammanfattning

Universitetsschemaläggningsproblemet är ett NP-fullständigt problem som skolor över hela världen måste hantera innan varje termin. Syftet med problemet är att schemalägga händelser, såsom föreläsningar och seminarier, utan att bryta flertalet fördefinierade villkor.

Denna studie hade som mål att automatisera denna process med hjälp av algoritmkonstuktionsmetoden simulerad glödgning och sedan jämföra resultatet med en genetisk algoritm. De datamängder som användes är inspirerade av den verkliga situationen på KTH. Resultaten visar stora tidsmässiga skillnader där algoritmen baserad på simulerad glödgning går snabbare. De visar dock också att den genetiska algoritmen har en bättre prestanda i tidigare stadier än i senare. Detta ledde till slutsatsen att en mer optimerad hybridalgoritm kan skapas av de två algoritmerna, förutsatt att den genetiska algoritmen kan dra nytta av förbättringar som föreslagits i tidigare forskning.

Contents

Chapter 1

Introduction

Each day we face the problem of getting our schedule to align with other people's. Scheduling is in general a very difficult problem that can be found everywhere: at universities and high schools, in public transport, at hospitals and a vast number of other institutions.

Universities all over the world need to solve the scheduling problem at least once before each semester. If done manually, massive amount of time need to be spent on making a suitable schedule. The schedule needs to fulfill several constraints. Common constraints are that only one teacher can teach one class at one specific time, a room can only be occupied by one class at a time and students should not have more than one class each time period. These constraints are often divided into hard and soft constraints. [4] The hard constraints are not allowed to be violated, while the soft constraints may be violated, but with the setback of a less optimal schedule.

Due to the huge amount of time and money spent on scheduling manually, there have been numerous attempts to automate this task with the help of computers. Research has shown that this problem is most commonly NP-complete [2], however this of course depends on how many and how complex the constraints are. Due to the difficulty of the problem and the many different constraints, there is no general algorithm which will find the optimal solution for every timetable problem.

To get around this problem, several optimization algorithms have been implemented. These algorithms are mostly meta-heuristic and range from local search algorithms like tabu search [14] and simulated annealing [9] to evolutionary algorithms like particle swarm optimization [5] and genetic algorithms [13]. The reason for the many different algorithms being implemented is because of the complex nature of the problem. Almost every school has different constraints and pre-conditions which need to be fulfilled. The evolutionary algorithms mostly performs better in the early stages of the process whereas the local search algorithms performs better in the late stages. This has led to the creation of many hybrid algorithms [11] which use evolutionary algorithms to narrow down the search space and local search algorithms to find the best solution in that space.

The original genetic algorithm was created by John Holland [6] in the early 1970's. The genetic algorithm is inspired by the evolution of life and was made to mimic some of life's evolutionary processes. It is an adaptive heuristic algorithm which uses an intelligent random search to find a solution to a problem. The intelligent random search is however by no means random. Instead it uses previous acquired information to find a more suitable candidate solution in the search space, thus fulfilling Darwin's quote *"Survival of the fittest"*.

The simulated annealing algorithm construction method was first proposed in 1983. [9] Annealing is a process in metallurgy where a metal is slowly cooled to make it stronger by reaching a low energy state. Based on this process the simulated annealing algorithm finds a solution to a problem. The simulated annealing algorithm is like the genetic algorithm also a heuristic algorithm.

Both algorithms have already previously been implemented and have successfully solved the university timetabling problem, for example by Andersson [1] as well as Pertoft and Yamazaki [13].

1.1 Purpose

Previous research has shown that evolutionary algorithms are good for exploring the whole search space. [7] As this might be the case for timetabling problems, a genetic algorithm is interesting for a comparison. Renman and Fristedt [14] presents a tabu search that does not perform as good as the genetic algorithm they compared it with. They do however state that there are other kinds of local searches, like simulated annealing, that might perform better than the genetic algorithm. Therefore, this study compares a simulated annealing algorithm with a genetic algorithm.

Hybrids of several different algorithms are commonly used nowadays to solve the timetabling problem. [14] Therefore it would be interesting to investigate whether or not the two algorithms would perform better as a hybrid than by themselves.

1.2 Problem statement

The main study will be to investigate which of the genetic algorithm constructed by Pertoft and Yamazaki [13] and the simulated annealing based algorithm constructed by the authors of this report, is fastest when executed on five different data sets.

The study will also investigate the potential benefit of creating a hybrid of the two algorithms.

1.2.1 Limitations

This study does not attempt to compare genetic algorithms with simulated annealing algorithms in general. The result of this study focuses on the differences between the two specific implementations presented in chapter 3. However, the conclusions can be used as an indication of how well implementations based on these heuristics will perform. There are also other types of algorithms, for example hybrid solvers [11], that combine solution methods to create faster and better algorithms. This study does only compare two specific kinds of solvers.

The data sets used as input to the algorithms are similar to the data sets used by Pertoft and Yamazaki [13], extended with a fifth data set inspired by the real course scheduling problem at the Royal Institute of Technology, KTH.

The algorithms use a common fitness function and the problem will be considered solved when the fitness value of a solution has reached 0.

The genetic algorithm that originally was written by Pertoft and Yamazaki [13], does not take soft constraints into consideration. Therefore, the simulated annealing algorithm does not implement these either. Essentially, the algorithms share fitness function to make them comparable. Due to this fact, an optimal solution will not be found, only an accepted.

1.3 Outline

The report is divided into six chapters. The first chapter introduces the subject, the problem statement and the purpose of the study. In chapter 2, Background, the university timetabling problem and the two different algorithms are described in general, whereas the third chapter, Method, consists of how the two specific algorithms are implemented. The results are shown in the fourth chapter and are discussed in the fifth, Discussion. Lastly the results are concluded in the final chapter, Conclusion.

Chapter 2

Background

This chapter starts with a presentation of the university timetabling problem, which is followed up by a section explaining constraints and three different classes of algorithms. Two algorithms for solving the university timetabling problem are lastly presented.

2.1 The university timetabling problem

The university timetabling problem is as aforementioned in the introduction, an NP-complete problem. The problem could be explained as followed: given a certain set of data and constraints a solution should be made which violates as few of the constraints as possible. The data set usually consists of the teachers, students and rooms and their capacity on the school. A room could also have certain abilities. For example, only laboratories can hold laboratory classes.

2.1.1 Time complexity

When the density of events increases and the amount of time slots are constant, the run time is increased more than linearly. Since the data sets and constraints differ so much between schools and because of the time complexity, an effective, general algorithm solver is infeasible to create. The problem is, however, considered NP-complete when using non-trivial constraints. [2]

2.2 Constraint based algorithms

Constraints can be divided into soft and hard constraints. [4] Hard constraints cannot be violated and should only be vital such as that a teacher can only have one class at a time and a room cannot hold more people than its capacity. Soft constraints consist of less important constraints such as: a student should not have long free time between classes or too many classes the same day. These constraints can be violated in favor of not violating a hard constraint, however with the result of a less optimal solution.

2.2.1 Three different classes of constraint based algorithms

There are three main classes of university timetabling algorithms according to Lewis [10]: "one-stage optimisation algorithms, two-stage optimisation algorithms, and algorithms that allow relaxations". Each of the algorithm types has its own advantages and disadvantages, and they have different efficiency for different kinds of problems. The algorithm that allows relaxation is redundant for this study and is therefore not described.

One-stage algorithms

The one-stage algorithms have one clear goal and a function returning a value of how close to the goal the solution is. Therefore, these algorithms can break both hard and soft constraints. High values are assigned to the hard constraints to avoid breaking them, thus forcing the algorithm to choose a solution which at worse only breaks the soft constraints. This category contains simple simulated annealing attempts and local search implementations, provided they are made in a way so that they do not start with a valid solution. [10]

Two-stage algorithms

The two-stage algorithms have two phases. In the first, only hard constraints are taken into account. After the first stage, a valid solution will exist. However, the solution found after stage one is not guaranteed to be optimized at all. The second stage is about refining the solution from the first stage to make it closer to the optimal solution. This stage also uses a function as in one-stage algorithms, but do not need the hard constraints to be weighted with a very high weight to be taken into account. This is because the solution during stage two only will be refined, and never invalid. The simulated annealing can be used as an example in this category too, if a valid solution is created before running the actual annealing. [10]

2.3 Meta-heuristic algorithms

Many different algorithms have been constructed to solve the university timetabling problem. These are most commonly meta-heuristic algorithms using the power of evolution such as the genetic algorithm or local search such as simulated annealing. These kinds of algorithms calculate an approximate solution rather than the optimal one. This is to severely decrease the run time of the program and still get an acceptable solution.

2.3.1 Genetic algorithm

A genetic algorithm starts with a set of random solutions to the problem. The initial solution is randomized and therefore crude. Each solution is called a chromosome and consists of several genes which are values corresponding to certain properties in the solution. These genes can then be used to control the fitness of the chromosome. Based on the chromosomes' fitness, they are crossed with each other to create a new offspring. These offsprings are then randomly mutated to create a bigger search space. When an offspring matches a specified fitness condition, this means an acceptable solution has been found and the algorithm terminates. [12] There are two main stages in the genetic algorithm: the selection and the crossover.

Selection

When to select which chromosomes are to be crossed there are a few different ways. Some of these are elitism selection, roulette-wheel selection and tournament selection. [13]

Crossover

It may vary which genes are carried over when two chromosomes are being crossed. To decide this there are few different methods. Some of them are single point crossover, two point crossover and uniform crossover. [13]

2.3.2 Simulated annealing

Simulated annealing is based on neighborhood search with the special property of sometimes accepting a worse solution to avoid getting caught in a local optimum and instead finding the global one. [8] The idea of simulated annealing is inspired by the annealing process in metal work. The colder a metal is, the more stable its shape is. To change the shape of the metal it is heated up and then processed while it is cooling down, ultimately freezing its shape until reheated.

Simulated annealing works in a similar way, where it has a temperature variable controlling the heating process. The temperature variable is initially set to a high value and is then slowly decreased while the algorithm runs. The higher the temperature is, the more probable the algorithm is to choose a worse solution than the current one. This gives the algorithm the chance of avoiding getting stuck in a local optimum early on. As the temperature decreases, so does the chances of the algorithm choosing a worse solution, which in the end leads to a local search in a much more narrow search space and hopefully finding a, close to, optimal solution.

Algorithms using only downhill search have a very large chance of getting stuck in a local optimum, whereas a better global optimum might be found just a few neighbors away. The graduated cooling process terminates this problem effectively and makes it much better than the downhill algorithms on large search space with numerous local optima. [9]

Acceptance function

For the algorithm to be able to determine whether or not it should accept a worse solution, an acceptance function is implemented. This function will return a value between 0 and 1, and represents the probability of choosing the newly created solution.

Commonly, the acceptance function will return a greater value when the temperature is high, and a lower value when the temperature is low. It will also depend on the difference between the two solutions. If the new solution is far worse than the current, the acceptance function will return a small value. The acceptance function should also immediately accept a better solution.

Process

A general description of the simulated annealing process can be viewed as followed:

1. A random solution, a specified temperature and a cool down rate is initially set as start values.

2. The algorithm iterates until a stop condition is met. This condition could be based on a time limit, the finding of an acceptable solution or the temperature reaching zero.

3. After each iteration the solution will be altered in some way.

4. The algorithm will then compare the current and the altered solution and choose a new current solution based on the outcome of the acceptance function.

5. Lastly the temperature will decrease by the value of the specified cool down rate and a new iteration will commence.

Values

Depending on how high the initial temperature and cool down rate variables are set, the algorithm will behave differently. If the initial temperature is high the more likely the algorithm is to choose a worse solution. If the initial temperature is set close to zero the algorithm will likely only choose better solutions and therefore running the risk of only finding a local optimum.

If the cool down rate is high, less iterations will be made and the chance of finding the global optimum will be lowered. The frequency of the algorithm choosing worse solutions will also be drastically lowered each iteration since the temperature will be lowered more rapidly.

Chapter 3

Method

This chapter describes the methodology of the comparative study. It includes a brief description of the algorithm implementations used for the test, how the data set was chosen and which constraints were used.

3.1 Test approach

A test was performed by providing the data set to the algorithm as a file, and running the algorithm in the test environment. The time was measured in milliseconds by the test suite Java program, which can be found in appendix A. When finished, the test suite printed the total time elapsed by comparing the system time with the time stamp saved when starting the test. The algorithms were considered finished when the fitness level had reached 0. The fitness level function can be found in section 3.4.1.

A run time limit of 550 seconds was introduced since the genetic algorithm could not find a solution in a reasonable amount of time for the XL data set.

To compare the algorithms the problem was solved 20 times per data set and algorithm. The two algorithms runs were plotted on separate bar charts and the median run time was then plotted into separate diagrams together with the standard deviation.

3.1.1 Environment

The algorithms were run on an HP Envy 15 Notebook PC, with a 2.50 GHz AMD A10 processor with 8 GB RAM. The only program running at the time was the Command Prompt running the implemented Java files and the internet connection was disabled. The algorithms were run one at a time, with different data sets under equal conditions.

3.2 Algorithms

The source code for the test suite and the algorithms implemented in Java can be found in appendix A. The algorithms are here explained in pseudo code.

3.2.1 Data structures

The data structures used by Pertoft and Yamazaki [13] were reused for the simulated annealing algorithm. A solution consisted of a list of rooms and the timetable for each room. These room timetables were represented as matrices with columns for each day and rows for each time slot that day. The matrix contained integers representing the identification number of the event for that time slot.

When creating a new solution, all the room matrices were copied element by element. This can be done in a time, linear to the amount of time slots.

3.2.2 Genetic algorithm

The genetic algorithm was implemented by Pertoft and Yamazaki [13] and uses single point crossover and roulette-wheel selection. The algorithm was optimized by Pertoft and Yamazaki [13] with a weighted fitness function, since some of the hard constraints were found to be more often violated in the beginning than others. In general terms, it can be described with the pseudo code in algorithm 1, quoted from the study made by Pertoft and Yamazaki [13].

The genetic algorithm was run with the same start condition as Pertoft and Yamazaki [13] used, which was a population size of 100 timetables and a mutation rate of 6 %. These conditions were chosen since Pertoft and Yamazaki [13] did extensive testing and found out these were the best.

Algorithm 1 Genetic algorithm.

```
function FIND_BEST_GA
    use a randomized population and evaluate fitness of its chromosomes
    while most fit individual is not fit enough do
        while offspring population is not full do
            select two parent chromosomes with roulette selection
            perform single point crossover with the two parent chromosomes
            mutate offspring chromosome
            repair offspring chromosome
            evaluate fitness of offspring chromosome
            add offspring chromosome to offspring population
        end while
        merge the parent and offspring populations
        delete the rest of the chromosomes
    end while
    return most fit chromosome from population
end function
```

3.2.3 Simulated annealing

The concept of simulated annealing was followed strictly when constructing the algorithm shown in algorithm 2. A random solution was generated and start (T_{start}) and final (T_{final}) temperatures were given, to create an interval. The cooling rate k was also provided. The algorithm iterates over the temperatures in the interval, and cools it with a factor k each time. A modified solution is produced, and compared to the current.

The acceptance function shown in equation 3.1 was used. It was chosen due to it being commonly used [8]. E_{new} represents the fitness of the new solution, E_{old} the fitness of the old solution and T the temperature.

$$e^{\frac{E_{new} - Eold}{T}} \tag{3.1}$$

For this study the values in table 3.1 were used as this created an even spread and right amount of iterations while still being time efficient.

Parameter	**Value**
T_{start}	100
T_{final}	0.7
k	0.9995

Table 3.1. The values provided for the simulated annealing algorithm.

As each solution is its own Java object instance, the timetable will be copied each iteration. This can be done with a time consumption linear to the amount of time slots.

Algorithm 2 Simulated annealing.

```
1: function FIND_BEST_SA($sol_{bad}$, $T_{start}$, $T_{final}$, $0 < k < 1$)
2:     $sol_{current} \leftarrow sol_{bad}$
3:     $sol_{best} \leftarrow sol_{bad}$
4:     $T \leftarrow T_{start}$
5:     while $T > T_{final}$ do
6:         $sol_{new} \leftarrow$ MODIFY($sol_{current}$)
7:         if accept(energy($sol_{current}$), energy($sol_{new}$), $T$) > rand(0,1) then
8:             $sol_{current} \leftarrow sol_{new}$
9:         end if
10:        if energy($sol_{new}$) > energy($sol_{best}$) then
11:            $sol_{best} \leftarrow sol_{new}$
12:        end if
```

```
13:         $T \leftarrow T * k$
14:     **end while**
15:     **return** $sol_{best}$
16: **end function**
17:
18: **function** ACCEPT($E_{old}$, $E_{new}$, $T$)
19:     **if** $E_{new} \geq E_{old}$ **then**
20:         **return** 1
21:     **else**
22:         $x \leftarrow \frac{E_{new} - E_{old}}{T}$
23:         **return** $e^x$
24:     **end if**
25: **end function**
26:
27: **function** MODIFY($solution$)
28:     **return** A randomly modified *solution*, where two time slots switched events
29: **end function**
```

3.3 Data sets

The algorithms were run on five different data sets. Four of them were the same as the ones used by Pertoft and Yamazaki [13]. The XL data set was created with inspiration from the real situation at the Royal Institute of Technology, KTH.

The data was formatted according to figure 3.1. Each section started with an octothorpe (#) followed by the name of the section. Each section then contained a number of entries for all the different properties.

```
# ROOMS
RoomName RoomCapacity RoomType
...
# COURSES
CourseName NumOfLectures NumOfLessons NumOfLabs
...
# LECTURERS
LecturerName CourseA CourseB
...
# student groupS
student groupName NumOfStudents CourseA CourseB
...
```

Figure 3.1. The format of the data set used by the algorithms.

The different data sets are summarized in Table 3.2 and can be found in whole in appendix B. The smaller data sets have less rooms, courses, lecturers and students. However, the event density, the ratio between the number of events and time slots, is kept between 0.41 and 0.54 due to the changing amount of time slots.

Input Data File	**XS**	**S**	**M**	**L**	**XL**
Lecture Rooms	1	2	2	3	4
Lesson Rooms	2	3	5	6	10
Lab Rooms	2	3	5	7	11
Total number of rooms	5	8	12	16	25
Courses	6	12	15	21	29
Lecturers	4	9	12	15	21
Student Groups	3	6	8	12	21
Total Events	41	70	115	159	293
Total Time slots	100	160	240	320	540
Event Density	0.41	0.44	0.48	0.50	0.54

Table 3.2. Summary of the different test data sets, inspired by the real scheduling problem at the Royal Institute of Technology, KTH.

3.4 Constraints

The problem is considered solved when the following criteria are satisfied:

- Every event in every course is assigned a time slot.
- All events are in the right kind of room.
- No student group has two events at the same time.
- No lecturer has two events at the same time.
- No two events are scheduled in the same room at the same time.
- No event is in a room with less capacity than the number of students at the event.

These constraints are referred to as hard constraints, which means that they are absolutely necessary for the solution to be valid. This is in contrast to soft constraints, which are not taken into consideration in this report.

3.4.1 Assessment

To grade the solution, a fitness level function, f, was used.

$$f(x_1, ..., x_4) = 2x_1 + x_2 + 4x_3 + 4x_4 \tag{3.2}$$

In equation 3.2, $x_1, ..., x_4$ are as in table 3.3.

variable	**meaning**
x_1	number of double booked student groups
x_2	number of double booked lecturers
x_3	number of room capacity breaches
x_4	number of room type breaches

Table 3.3. Description of the variables used in equation 3.2.

The reason why some constraints are being weighted more than others is because Pertoft and Yamazaki [13] noticed that their algorithm performed faster if these constraints were solved early. By increasing their weight, both algorithms will avoid these violations early on, thus making their fitness value increase faster.

Chapter 4

Results

The results from the genetic algorithm can be found in figure 4.1. Notice the logarithmically scaled y-axis. Each bar represents a test run, with run time on the y-axis. Each cluster represents a data set. The XS data set produced a valid solution in less than 0.5 seconds for all runs, while the S data set had two runs that took almost doubled time compared to the others.

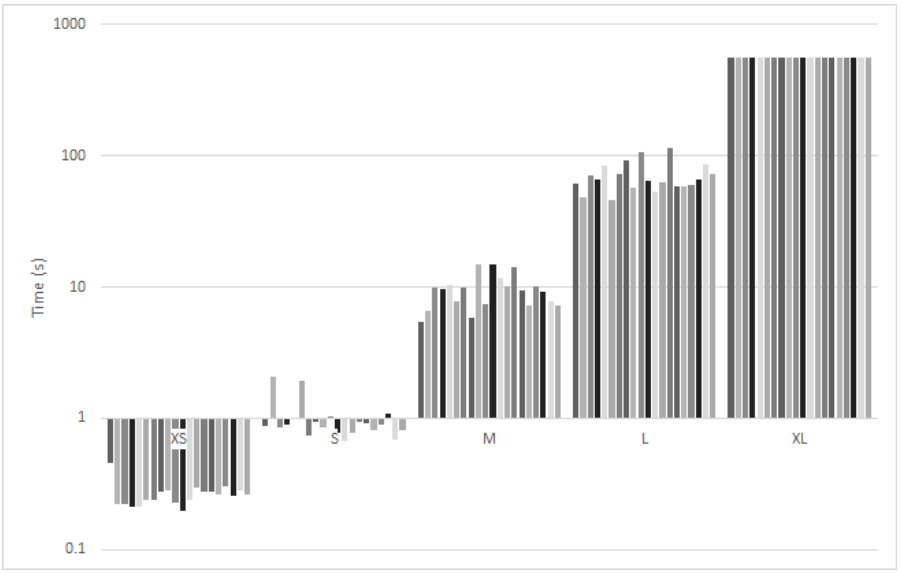


Figure 4.1. The results from the genetic algorithm on all timetables.

As can be seen in figure 4.1, the M, L and XL data sets need much more time to produce a solution.

When the XL data set was run with the genetic algorithm, no solution had been found after 550 seconds on any of the 20 runs. The fitness level had approximately been improved from −1076 to −69.

The simulated annealing results can be found in figure 4.2. This chart also has a logarithmically scaled y-axis, however the chart in figure 4.1 has a ten times higher value on the y-axis. The bars and clusters still represent run time and data sets.

As can be seen in figure 4.2, the spread between runs is noticeable. However, the time consumption is for all test runs smaller than the genetic algorithm's respective test runs.

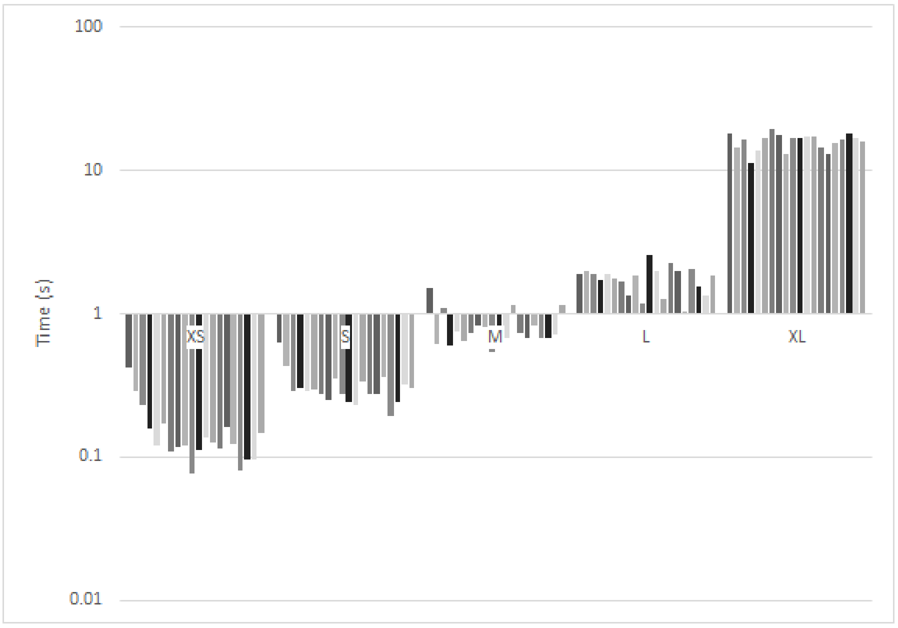
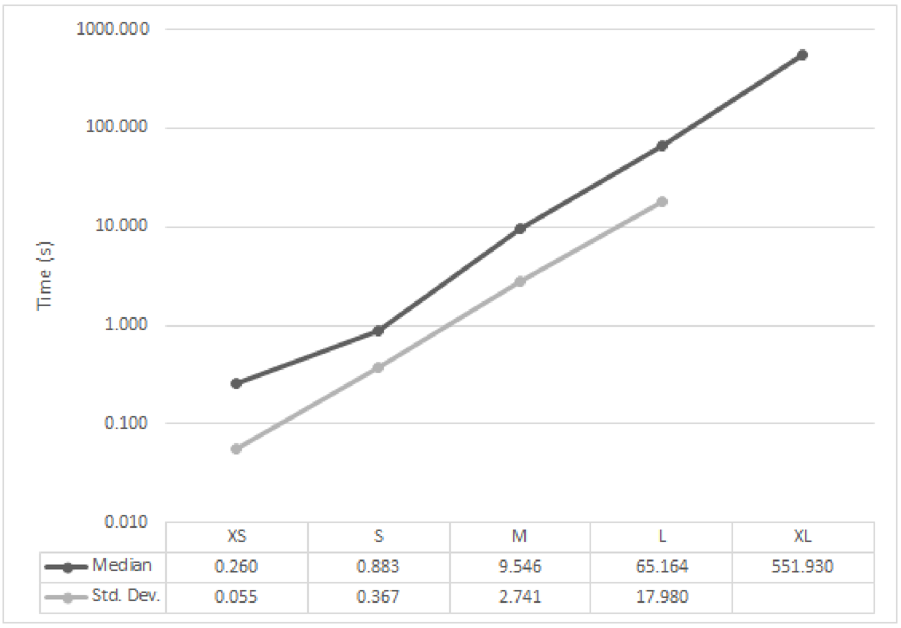


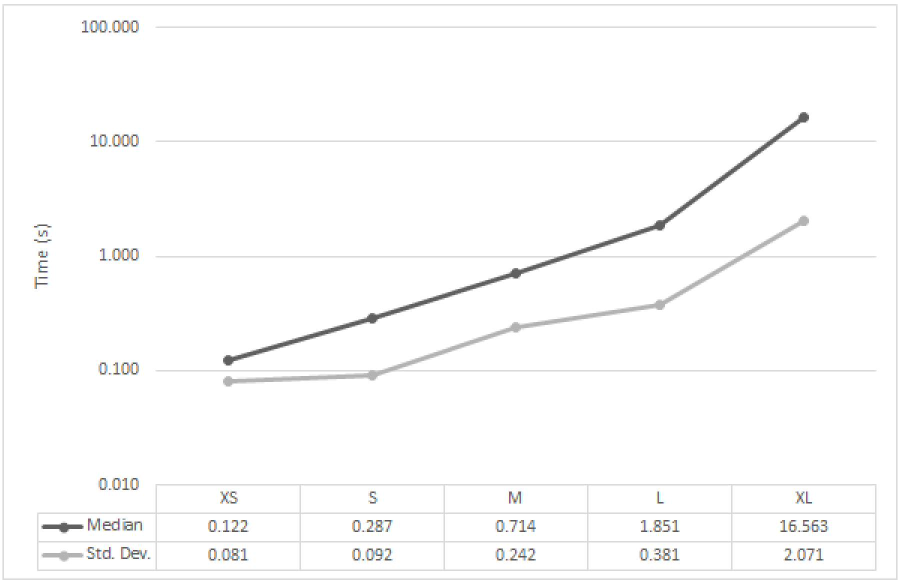
Figure 4.2. The results from the simulated annealing on all timetables.

The median time consumption and the standard deviation for the different timetables were plotted in figure 4.3 and figure 4.4. For comparative reasons, the XL runs were plotted in the genetic algorithm chart, even though they did not produce a valid solution. Both charts are drawn with a logarithmically scaled y-axis. The genetic algorithm overall performs worse than the simulated annealing algorithm with respect to time consumption.



	XS	S	M	L	XL
Median	0.260	0.883	9.546	65.164	551.930
Std. Dev.	0.055	0.367	2.741	17.980	

Figure 4.3. The median time consumption and standard deviation of the genetic algorithm on each of the timetables. A valid solution for the XL timetable was not produced.



	XS	S	M	L	XL
Median	0.122	0.287	0.714	1.851	16.563
Std. Dev.	0.081	0.092	0.242	0.381	2.071

Figure 4.4. The median time consumption and standard deviation of the simulated annealing on each of the timetables.

An overview over the fitness improvement of one test run on the L timetable can be found in figure 4.5 and figure 4.6, for each algorithm. The genetic algorithm improved the solution quickly in the early stage of the run, and then slowly completed the solution. The simulated annealing based algorithm improved the fitness more evenly over the run. Noticeable is also the scale of the x-axis.

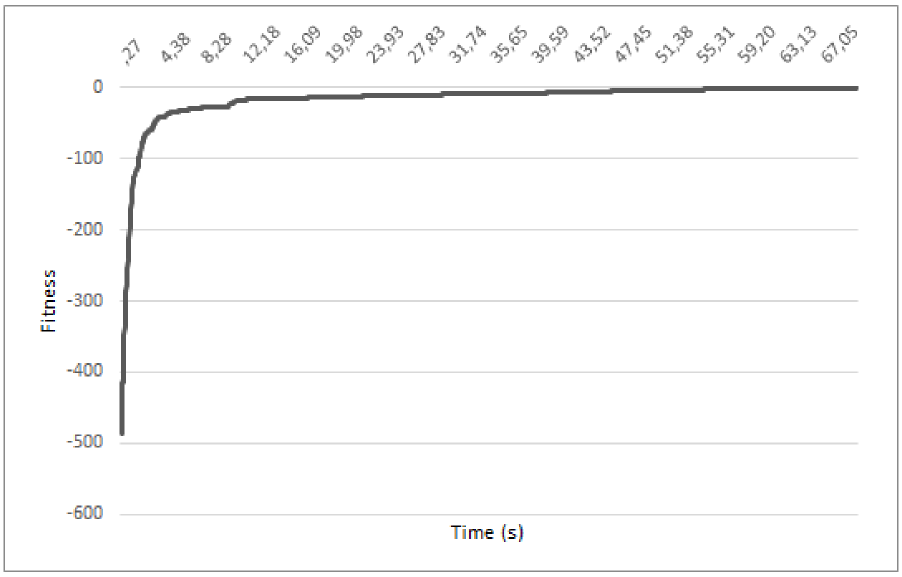


Figure 4.5. The genetic algorithm's fitness of the best solution found over time.

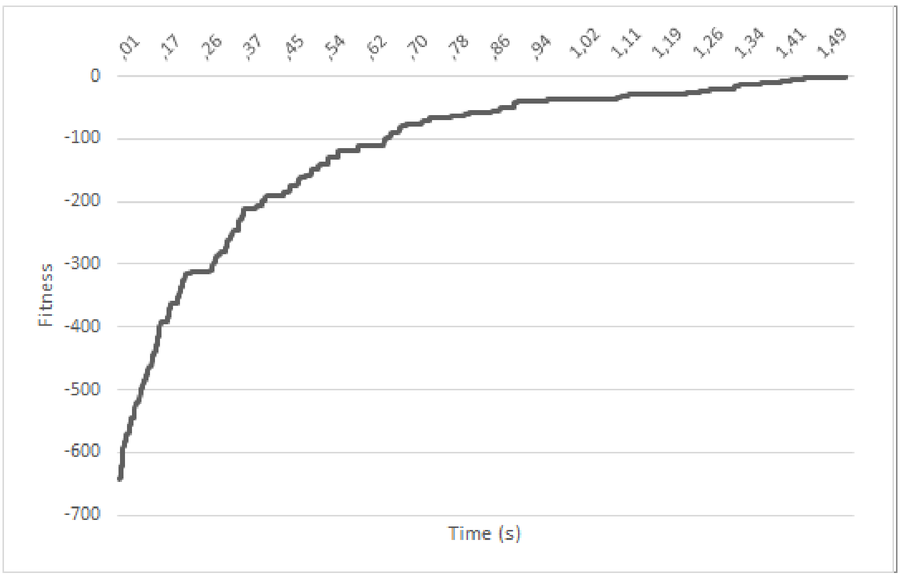


Figure 4.6. The simulated annealing's fitness of the best solution found over time.

The fitness value of the current solution for one test run on the L timetable with simulated annealing was plotted in figure 4.7. An increasing segment indicates that a better solution was accepted, and a decreasing segment indicates that a worse solution was accepted.

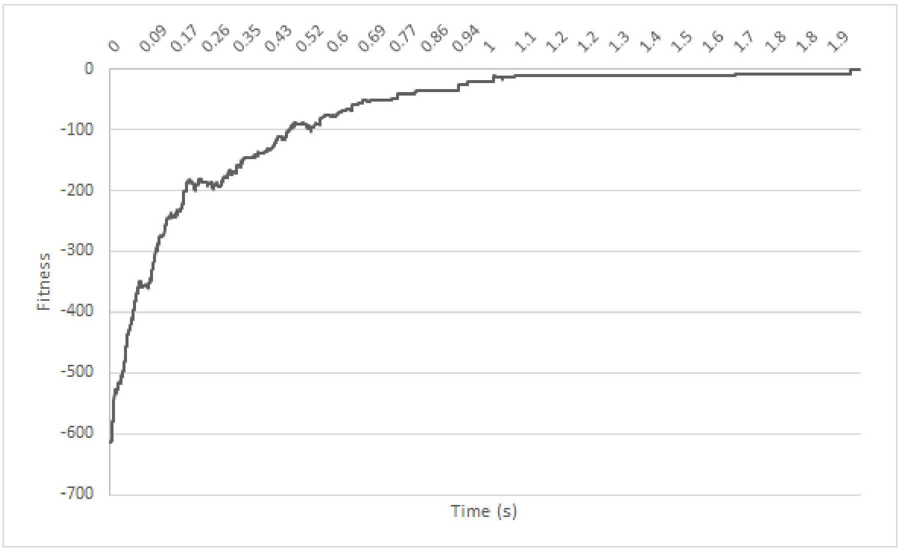


Figure 4.7. The simulated annealing's fitness of the current solution over time.

Chapter 5

Discussion

This chapter presents an analysis of the time consumed by the algorithms, followed by the differences between them. After that, the reliability of the study is commented, and some possible improvements are suggested.

5.1 Time complexity

The time consumed by the algorithms for solving the same problems increased more than linear when adding events and time slots to the data set. However, the genetic algorithm showed to grow faster when the input data increased.

What is interesting is the relatively rapid increase in fitness in the beginning of the genetic algorithm. The simulated annealing algorithm is however still better for every given time interval. In their report, Pertoft and Yamazaki [13] makes a few proposals to further improve their algorithm.

Given that these improvements would indeed result in a more optimized genetic algorithm, a hybrid of the two algorithms could turn out to perform better than each alone. This is due to their very different behaviors, where the genetic algorithms performs best in the early stages and the simulated annealing in the final stages. By integrating the two, the genetic algorithm would be used in the early stages to narrow down the search space and the simulated annealing algorithm would be used to find the best solution in that search space.

This hybrid will of course only work if it is possible to improve the genetic algorithm, but due to the scope of this study the suggestions given by Pertoft and Yamazaki [13] have not been implemented.

5.2 Main differences between the two algorithms

The reason for the difference in time complexity between the two algorithms are most probably due to their different implementations. The genetic algorithm creates a large amount of bad solutions, and crosses them with each other to finally get a good one. There is, however, no assurance that the solution will get acceptable fast, or even ever. This randomness is most likely the reason why there are two runs which took twice the amount of time in the S data set in figure 4.1.

The simulated annealing on the other hand, is steering its way to the final solution by forcing only better solutions to be produced when it approaches the end. The partial solutions are randomized over the whole search space, which the genetic algorithm only does in special cases.

The genetic algorithm written by Pertoft and Yamazaki [13] clearly performs worse than the simulated annealing method in these five test cases.

5.2.1 Basic local search may be sufficient

The study found the concept of these advanced meta-heuristics quite excessive. When running the simulated annealing, a random neighboring time slot is picked and substituted. Figure 4.7 does however show that almost every substitution of neighbors to a free time slot is improving the fitness. Therefore, a local search over these time slots would not get stuck into too many local optima. When the event density is around and below 50 %, as in these tests, these two algorithms may be too sophisticated. The computational power needed to produce and maintain all the instances of solutions quickly becomes unmanageable.

5.3 Reliability

The tests are not very lifelike, due to the vast number of free time slots available when the problem is solved. The closest data set to the situation is the XL one, which the simulated annealing algorithm handles quite well. This shows that the most benefit is obtained when there are few possible moves that do not increase fitness, where there are many local optima.

5.4 Improvements

There are some different improvements that can be made, both to the simulated annealing algorithm specifically and the university timetabling problem solving in general, in order to make them better. These are presented in this section.

5.4.1 Refined fitness function

The fitness function presented by Pertoft and Yamazaki [13] is optimized in order to make the genetic algorithm perform better. This makes the algorithm prioritize some properties before others. Since the same fitness function is used for the simulated annealing, these properties will also be prioritized by it. These are however different algorithms that are working differently. Therefore, a more detailed study of the impact of the fitness function can be done to improve the overall results.

5.4.2 Soft constraints

In the real world an algorithm which only returns a fair schedule would not be used. To be relevant it would also need to evaluate constraints which are not vital, but which should still be fulfilled if possible. These are the soft constraints as mentioned in chapter 2. To fulfill these objectives our simulated annealing algorithm would need an implementation of soft constraints.

These constraints could be implemented in the two ways described in chapter 2.2.1: either as a one stage or a two stage algorithm. If implemented as a two stage algorithm a hard constraint solution would first need to be found by the algorithm. By then using a fitness function which gives positive values if a soft constraint is fulfilled, the hard constraint solution could be optimized further.

If the algorithm was going to be designed as a one stage algorithm, both of the soft and hard constraint would need to be evaluated at the same time. To avoid getting a solution which fulfills soft constraints but violates hard constraints, the hard constraint fitness function would have to return higher penalty values than the soft constraint fitness function. This would result in that a breaking of a hard constraint would be such a high negative value that a fulfillment of several soft constraints would not nearly balance the end value. For future work, inspiration can be taken from Bogdanov [3], who presents a solution using this method.

Chapter 6

Conclusions

The implemented simulated annealing algorithm performs much better than the genetic algorithm by Pertoft and Yamazaki [13], with respect to time consumption in the context of this problem. The genetic algorithm performs relatively better than the simulated annealing in the early stages, whereas the latter performs better in the final stages. Increasing the event density and number of time slots results in a seemingly exponential time increase for both of the algorithms, which was to be expected since the university timetabling problem in general is NP-complete. Possible future work would be to improve the genetic algorithm in accordance to Pertoft and Yamazaki [13], to complement the simulated annealing thus creating a more optimized, hybrid algorithm.

Bibliography

[1] H. Andersson. “School Timetabling in Theory and Practice”. Bachelor’s thesis. Umeå: Umeå Universitet, 2015.

[2] Y. Awad, A. Badr, and A. Dawood. “An evolutionary immune approach for university course timetabling”. In: *IJCSNS International Journal of Computer Science and Network Security* 11.2 (2011), pp. 127–135.

[3] D. Bogdanov. “A Comparative Evaluation of Metaheuristic Approaches to the Problem of Curriculum-Based Course Timetabling”. Bachelor’s thesis. Stockholm: Royal Instititue of Technology, KTH, 2015.

[4] E. Burke et al. “Automatic University Timetabling: The State of the Art”. In: *The computer journal* 40.9 (1997), pp. 565–571.

[5] R. Chen and H. Shih. “Solving University Course Timetabling Problems Using Constriction Particle Swarm Optimization with Local Search”. In: *Algorithms* 2013.6 (2013), pp. 227–244.

[6] N. Dulay. *Genetic Algorithms.* Visited 2016-03-27. 1999. URL: `http://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol1/hmw/article1.html`.

[7] M. Fesanghary et al. “Hybridizing harmony search algorithm with sequential quadratic programming for engineering optimization problems”. In: *Comput. Methods Appl. Mech. Engrg* doi:10.1016/j.cma.2008.02.006 (2008).

[8] L. Jacobson. *Simulated Annealing for Beginners.* Visited 2016-03-17. 2013. URL: `http://www.theprojectspot.com/tutorial-post/simulated-annealing-algorithm-for-beginners/6`.

[9] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. “Optimization by Simulated Annealing”. In: *Science* 220.4598 (1983), pp. 671–680.

[10] R. Lewis. “A survey of metaheuristic-based techniques for university timetabling problems”. In: *OR Spectrum* 30 (2007), pp. 167–190.

[11] Z. Lü and J. Hao. “Solving the Course Timetabling Problem with a Hybrid Heuristic Algorithm”. In: *AJMSA* LNAI 5253 (2008), pp. 262–273.

[12] M. Mitchell. *An Introduction to Genetic Algorithms.* Cambridge, MA: MIT Press, 1996. ISBN: 9780585030944.

[13] J. Pertoft and H. V. Yamazaki. “Scalability of a Genetic Algorithm that solves a University Course Scheduling Problem Inspired by KTH”. Bachelor’s thesis. Stockholm: Royal Instititue of Technology, KTH, 2014.

[14] C. Renman and H. Fristedt. “A comparative analysis of a Tabu Search and a Genetic Algorithm for solving a University Course Timetabling Problem”. Bachelor’s thesis. Stockholm: Royal Institute of Technology, KTH, 2014.

Appendix A

Source code

The Java source code for the implemented algorithms can be found in the public GitHub repository found at `https://github.com/jonasdahl/algorithm-comparison`.

Appendix B

Data sets

B.1 XL - Extra large

```
# ROOMS
Q1 250 0
D1 200 0
D2 50 1
D3 50 1
D45 40 1
D46 40 1
D31 40 1
D32 40 1
E1 350 0
E35 40 1
E36 40 1
E51 40 1
E52 40 1
F1 300 0
Q1 300 0
ALBA 400 0
TEXC 60 2
SPEL 40 2
SPOR 30 2
MUSI 40 2
ROD 30 2
ORA 30 2
VIO 40 2
GRA 30 2
KAR 30 2
MAG 30 2
BRU 30 2
# COURSES
CALC 2 1 0
JAVA 1 0 1
```

```
MULT 2 0 1
CTEC 1 2 0
CSEC 0 1 1
SCON 1 1 1
DIGI 1 0 1
ENGM 1 0 1
ALGD 1 1 0
ELEC 1 0 0
PROB 1 0 1
OPER 1 1 0
TERM 2 0 1
DIFF 2 1 0
MECH 0 1 2
QUAN 1 1 0
OOPC 1 1 1
TCHE 2 1 0
PERS 1 0 0
REAC 1 0 2
POLY 1 1 0
MAGN 2 2 0
POLT 3 2 1
NUMD 2 2 3
TERT 2 0 0
DDED 3 2 0
MAGA 3 1 0
NUMA 3 1 3
TERA 3 1 0
# LECTURERS
SVEN CALC MULT
BERT JAVA SCON OOPC
KARL CSEC
GUNN CTEC
BERI DIGI
ERIK DIFF POLT
SARA OPER
OLLE ENGM ELEC
BENG ALGD
JUDI TERM REAC
MANS MECH MAGN
MICH QUAN
PELL PROB
DARI TCHE POLY
MORT PERS
LEFT TERA
PATR TERT
MIHA DDED
DILI NUMA
CGRI NUMD
STEF MAGA
```

```
# STUDENTGROUPS
COMP_1 200 CALC JAVA
COMP_2 120 MULT CTEC
COMP_3 70 CSEC SCON
INFO_1 200 DIGI ENGM
INFO_2 100 ALGD ELEC
INFO_3 50 PROB OPER
PHYS_1 200 CALC TERM
PHYS_2 180 DIFF MECH
PHYS_3 100 QUAN OOPC
CHEM_1 150 CALC TCHE
CHEM_2 130 PERS DIFF
CHEM_3 100 REAC MAGN
DDOS_1 150 POLY POLT
DDOS_2 140 NUMD TERT
DDOS_3 120 MAGA DDED
BIZZ_1 150 POLY POLT
BIZZ_2 140 TERA
BIZZ_3 120 NUMA
MIZZ_1 50 POLY MECH
MIZZ_2 40 CALC
MIZZ_3 20 ELEC
```

B.2 L - Large

```
# ROOMS
D1 200 0
D2 50 1
D3 50 1
D45 40 1
D46 40 1
E1 350 0
E35 40 1
E36 40 1
F1 300 0
SPEL 40 2
SPOR 30 2
MUSI 40 2
ROD 30 2
ORA 30 2
VIO 40 2
GRA 30 2
# COURSES
CALC 2 1 0
JAVA 1 0 1
MULT 2 0 1
CTEC 1 2 0
```

```
CSEC 0 1 1
SCON 1 1 1
DIGI 1 0 1
ENGM 1 0 1
ALGD 1 1 0
ELEC 1 0 0
PROB 1 0 1
OPER 1 1 0
TERM 2 0 1
DIFF 2 1 0
MECH 0 1 2
QUAN 1 1 0
OOPC 1 1 1
TCHE 2 1 0
PERS 1 0 0
REAC 1 0 2
POLY 1 1 0
# LECTURERS
SVEN CALC MULT
BERT JAVA SCON OOPC
KARL CSEC
GUNN CTEC
BERI DIGI
ERIK DIFF
SARA OPER
OLLE ENGM ELEC
BENG ALGD
JUDI TERM REAC
MANS MECH
MICH QUAN
PELL PROB
DARI TCHE POLY
MORT PERS
# STUDENTGROUPS
COMP_1 200 CALC JAVA
COMP_2 120 MULT CTEC
COMP_3 70 CSEC SCON
INFO_1 200 DIGI ENGM
INFO_2 100 ALGD ELEC
INFO_3 50 PROB OPER
PHYS_1 200 CALC TERM
PHYS_2 180 DIFF MECH
PHYS_3 100 QUAN OOPC
CHEM_1 150 CALC TCHE
CHEM_2 130 PERS DIFF
CHEM_3 100 REAC POLY
```

B.3 M - Medium

```
# ROOMS
D1 200 0
D2 50 1
D3 50 1
D45 40 1
D46 40 1
E1 350 0
E35 40 1
SPEL 40 2
SPOR 30 2
MUSI 40 2
ROD 30 2
ORA 30 2
# COURSES
CALC 2 1 0
JAVA 1 0 1
MULT 2 0 1
CTEC 1 2 0
CSEC 0 1 1
SCON 1 1 1
DIGI 1 0 1
ENGM 1 0 1
ALGD 1 1 0
ELEC 1 0 0
PROB 1 0 1
OPER 1 1 0
TERM 2 0 1
DIFF 2 1 0
MECH 0 1 2
QUAN 1 1 0
OOPC 1 1 1
TCHE 2 1 0
PERS 1 0 0
REAC 1 0 2
POLY 1 1 0
# LECTURERS
SVEN CALC MULT
BERT JAVA SCON OOPC
KARL CSEC
GUNN CTEC
BERI DIGI
ERIK DIFF
SARA OPER
OLLE ENGM ELEC
BENG ALGD
JUDI TERM REAC
```

```
MANS MECH
MICH QUAN
PELL PROB
DARI TCHE POLY
MORT PERS
# STUDENTGROUPS
COMP_1 200 CALC JAVA
COMP_2 120 MULT CTEC
COMP_3 70 CSEC SCON
INFO_1 200 DIGI ENGM
INFO_2 100 ALGD ELEC
INFO_3 50 PROB OPER
PHYS_1 200 CALC TERM
PHYS_2 180 DIFF MECH
```

B.4 S - Small

```
# ROOMS
D1 200 0
D3 50 1
D45 40 1
E1 350 0
E35 40 1
SPEL 40 2
SPOR 30 2
MUSI 40 2
# COURSES
CALC 2 1 0
JAVA 1 0 1
MULT 2 0 1
CTEC 1 2 0
CSEC 0 1 1
SCON 1 1 1
DIGI 1 0 1
ENGM 1 0 1
ALGD 1 1 0
ELEC 1 0 0
PROB 1 0 1
OPER 1 1 0
TERM 2 0 1
DIFF 2 1 0
MECH 0 1 2
QUAN 1 1 0
OOPC 1 1 1
TCHE 2 1 0
PERS 1 0 0
REAC 1 0 2
```

```
POLY 1 1 0
# LECTURERS
SVEN CALC MULT
BERT JAVA SCON OOPC
KARL CSEC
GUNN CTEC
BERI DIGI
ERIK DIFF
SARA OPER
OLLE ENGM ELEC
BENG ALGD
JUDI TERM REAC
MANS MECH
MICH QUAN
PELL PROB
DARI TCHE POLY
MORT PERS
# STUDENTGROUPS
COMP_1 200 CALC JAVA
COMP_2 120 MULT CTEC
COMP_3 70 CSEC SCON
INFO_1 200 DIGI ENGM
INFO_2 100 ALGD ELEC
INFO_3 50 PROB OPER
```

B.5 XS - Extra small

```
# ROOMS
D1 200 0
D45 40 1
E35 40 1
SPEL 40 2
SPOR 30 2
# COURSES
CALC 2 1 0
JAVA 1 0 1
MULT 2 0 1
CTEC 1 2 0
CSEC 0 1 1
SCON 1 1 1
DIGI 1 0 1
ENGM 1 0 1
ALGD 1 1 0
ELEC 1 0 0
PROB 1 0 1
OPER 1 1 0
TERM 2 0 1
```

```
DIFF 2 1 0
MECH 0 1 2
QUAN 1 1 0
OOPC 1 1 1
TCHE 2 1 0
PERS 1 0 0
REAC 1 0 2
POLY 1 1 0
# LECTURERS
SVEN CALC MULT
BERT JAVA SCON OOPC
KARL CSEC
GUNN CTEC
BERI DIGI
ERIK DIFF
SARA OPER
OLLE ENGM ELEC
BENG ALGD
JUDI TERM REAC
MANS MECH
MICH QUAN
PELL PROB
DARI TCHE POLY
MORT PERS
# STUDENTGROUPS
COMP_1 200 CALC JAVA
COMP_2 120 MULT CTEC
COMP_3 70 CSEC SCON
```