

William Ke

AI for Room Assignment

Master's thesis in Cybernetics and Robotics

Supervisor: Ole Morten Aamo

June 2021

William Ke

AI for Room Assignment

Master's thesis in Cybernetics and Robotics

Supervisor: Ole Morten Aamo

June 2021

Norwegian University of Science and Technology

Faculty of Information Technology and Electrical Engineering

Department of Engineering Cybernetics



Norwegian University of
Science and Technology

Abstract

The objective of this Master's thesis is to design and implement an algorithm that is able to automatically assign workplaces to students while satisfying a set of constraints. The problem of assigning workplaces to students was able to be formulated as a quadratic assignment problem, with the set of constraints expressed an objective function. Using an implementation of the hill climbing method, we were able to solve for a locally optimal solution.

Sammendrag

Formålet med denne masteroppgaven er å designe og implementere en algoritme som automatisk kan tildele arbeidsplasser til studenter samtidig som et sett med begrensninger blir overholdt. Problemet med å tildele arbeidsplasser til studenter var i stand til å bli formulert som quadratic assignment problem, med settet av begrensninger uttrykt som en objekt funksjon. Ved en implementasjon av hill climbing metoden kunne vi løse for en lokalt optimal løsning.

Preface

This master's thesis marks the finalisation of my Master's degree in Cybernetics and Robotics at the Norwegian University of Science and Technology (NTNU). The project was written under the supervision and guidance of Professor Ole Morten Aamo, who did a great job assisting me throughout the thesis. Moreover, I would like to express my gratitude and appreciation to all my friends and family for their support throughout my years as a student at NTNU.

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, 15th June 2021*

Contents

Abstract	iii
Sammendrag	iv
Preface	v
Contents	vi
Glossary	vii
1 Introduction	1
1.1 Background and Motivation	1
1.2 Objective	1
1.3 Contributions	2
1.4 Thesis Structure	2
2 Theory	3
2.1 Combinatorial optimisation	3
2.2 Room assignment problem	4
2.2.1 Quadratic Assignment Problem (QAP)	4
2.3 Solution approaches/techniques	5
2.3.1 Local Search Based Techniques	5
2.3.2 Population Based Algorithms	8
3 Design	10
3.1 Problem description	10
3.1.1 Problem formulation	11
4 Implementation	13
4.1 Data model	13
4.2 Implementation of the algorithm	15
4.2.1 Assign those who applied for category 4 and 5	15
4.2.2 balancing the optimisation problem	16
4.2.3 pre-assign students	17
4.2.4 optimisation loop	17
5 Results and Discussion	20
5.1 Case 1	20
5.2 Case 2	21
6 Conclusion	25
6.1 Further Work	25
Bibliography	27

Glossary

approximate algorithm is algorithms that find approximate solutions to optimisation problems with provable guarantees on the distance of the returned solution to the global optimal one . vii

combinatorial optimisation is a subfield within mathematical optimisation that consists of finding an optimal object from a finite set of objects.. vii

exact algorithm is an algorithm that always solve an optimisation problem to global optimality. . vii

linear assignment problem(LAP) If the total cost of the assignment for all tasks is equal to the sum of the costs for each agent, then the problem is called the linear assignment problem. vii

Chapter 1

Introduction

1.1 Background and Motivation

Combinatorial optimisation is a subfield within mathematical optimisation. This field covers problems that consists of finding an optimal object from a finite set of object. [1]. Many examples of these kinds of problems typically have search spaces that are too large to enumerate exhaustively using brute force. Different techniques and methods have to therefore be employed to solve these problems in practice.

Notable examples for these kinds of problems include: The Travelling Salesman Problem [2], Bin-packing problem [3], and the knapsack problem [4]. Combinatorial optimisation has otherwise important applications in numerous distinct fields such as: artificial intelligence, software engineering, applied mathematics, operational research, and decision theory.

The motivation for this project is to automate the process assigning workplaces to fifth-year students at The Department of Engineering Cybernetics at NTNU. The process is currently done manually by having a administrator manually assigning each workplace to each student. About 170 students have to be assigned to a workplace each year according the students wished while keeping track of the necessary constraints. The constraint include not assigning a workplace to multiple students, and trying to seat students who wish to share an office into workplaces into the same room. 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 making mistakes.

1.2 Objective

The objective of this master's thesis is to design and implement an algorithm that is able to automatically assign workplaces for fifth grade students at the Department of Engineering

Cybernetics at NTNU. The workplaces have different categories, such as fixed space, with or without a computer, or drop-in. The students gets to state their wishes regarding category of workplace, as well as who they want to share an office with. The problems has some additional constraints that should be satisfied. These constraints will be given as well as a more complete description of the problem in Section 3.1

The above problem can be defined as a mathematical optimisation problem and the criteria can be expressed a objective function. The goal of the algorithm can then be reduced to solving for a solution that minimises the objective function.

1.3 Contributions

We propose a model for describing the room allocation optimisation problem. Furthermore, we propose an heuristic algorithm that produces a locally optimal solution for 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. Chapter 4 will be for presenting a data model that the algorithm will uses, as well as the implementation of the algorithm itself. Chapter 5 presents the results and discussion of the output of our implemented algorithm. Finally, Chapter 6 will give a conclusion of our work and results, as well as any potential further work.

Chapter 2

Theory

This chapter is for giving an overview of the necessary theory that is relevant for solving the problem as described in Section 1.2. Section 2.1 will be briefly about combinatorial optimisation. Section 2.2 will describe the room assignment problem. Section 2.3 will be about methods and techniques for solving the room assignment problem.

This master's thesis can be seen as a continuation of my specialisation project that I wrote in autumn 2020, about *AI for Exam Allocation* [5]. The methods and techniques for solving the exam allocation problem can also be used to solve the room assignment problem as both are in essence variants of the assignment problem. Section 2.3 about methods and techniques are therefore directly quoted from the specialisation project report *AI for Exam Allocation* [5].

2.1 Combinatorial optimisation

Combinatorial optimisation is an important subfield within mathematical optimisation. The subfield is concerned with problems where one wants to find an optimal object from a finite set of candidate objects. This is in opposition to regular optimisation where the decision variables may be continuous.

A common characteristic of problems in combinatorial optimisation is that the space of possible solutions is too large to search exhaustively using brute-force searching. Exact algorithms that are able to solve for a global optimum are only practical viable for small problem sizes. There are however several alternative approaches for solving these problems.

one approach is to use heuristics [6] or approximation algorithms [7] to solve for nearly optimal solutions in a more reasonable timeframe.

a second approach is to try to look for special cases of the problem where there may exist an exact algorithm with a reasonable runtime or where heuristic algorithms may give better

solutions.

2.2 Room assignment problem

2.2.1 Quadratic Assignment Problem (QAP)

The room assignment problem can be seen as a variant of the quadratic assignment problem. The quadratic assignment problem was first introduced by Koopmans and Beckmann (1957) [8]. The objective of the original problem is to assign a set of facilities to a set of locations with the goal of minimising the total assignment cost. The cost is a function of the distance and flow between the facilities, in addition to costs associated with a facility being placed at a certain location.

The problem can be formulated as the following [9]:

Let n be the number of facilities and the number of locations. $F = (f_{ij})$, $D = (d_{kl})$, and $B = (b_{ik})$ are three $n \times n$ matrices with real numbers. f_{ij} is the flow between facility i and facility j , d_{kl} is the distance between location k and location l , and b_{ik} is the cost of placing facility i at location k . N is the set $N = \{1, 2, \dots, n\}$. S_n is the set of all permutations $\phi : N \rightarrow N$

$$\min_{\phi \in S_n} \sum_{i=1}^n \sum_{j=1}^n f_{ij} D_{\phi(i)\phi(j)} + \sum_{i=1}^n b_{i\phi(i)} \quad (2.1)$$

When it come to facility location the matrices F and D are symmetric with zeroes on the diagonal. The matrices F , D , and B are all nonnegative. The product $f_{ij} D_{\phi(i)\phi(j)}$ is the cost of assigning facility i to location $\phi(i)$ and facility j to location $\phi(j)$, hence the "quadratic" word in the name of the problem. The product $b_{i\phi(i)}$ is the linear term that denotes the cost that occurs from placing facility i at location $\phi(i)$. An instance of a QAP with input matrices F , D , and B is denoted by $QAP(F, D, B)$. If there is no linear term, i.e. $B = 0$, then an instance is denoted by $QAP(F, D)$

If one were to remove first term $\min_{\phi \in S_n} \sum_{i=1}^n \sum_{j=1}^n f_{ij} D_{\phi(i)\phi(j)}$ leaving us only with the second term $\sum_{i=1}^n b_{i\phi(i)}$, then the problem becomes known as a linear assignment problem (LAP).

an assignment problem where the number of facilities and the number of locations is equal to each other is known as a *balanced* assignment problem. This might seem like a problematic restriction at first glance since this might not be the case for many real world problems. However, it is possible to make any unbalanced problem to a balanced one by adding "dummy" facilities or locations.

The formulation of the original QAP can also be used to represent problems that arise in different fields, not just for location planning. The quadratic term of the QAP can be used

to represent any kind of cost that arises from the relationship in between how two object are assigned. Some examples of how the QAP can be applied are given in the paragraphs below.

Steinberg(1961) [10] used QAP for solving the backboard wiring problem. Controls and displays were to be placed on a panel. And the problem was to place the devices so as to minimise the total wire length used to connect the devices together.

Dickey and Hopkins(1972) [11] used QAP for campus planning. The problem consisted of placing buildings on sites, with the objective of minimising the total walking distance between buildings given the traffic intensity between buildings.

The travelling salesman problem can likewise be represented as a QAP [12]. The TSP on n cities can be formulated as a $QAP(F, D)$, where F is contains the distances between cities, and D is the adjacency matrix of a Hamiltonian cycle on n vertices.

The quadratic assignment problem is in contrast to the linear assignment problem a very hard optimisation problem to solve. The time complexity to solve a QAP using brute force search is $O(n!)$, as the set of all possible solution candidates for a QAP of size n is $n!$. Sahni and Gonzalez (1976) [13] shows in fact that the QAP is a NP-hard problem, and even finding an approximate solution within some constant factor from the global optimal solution cannot be done in polynomial time unless $P=NP$.

As mentioned at the start of the section, our room assignment problem can be represented as a $QAP(F, D, B)$. The F matrix is used to describe the students who should sit together, either because of the students own wishes or because the students have the same supervisor. The D matrix is used to define the distance between workplaces. The B is used to denote the cost of assigning a student to a workplace, with higher cost for assigning a student to a workplace of a different category than the student applied for. A more complete description of the model for our room assignment problem will be given in Section 3.1. The next section describes some methods that is able to solve for a sufficiently good solution to the room assignment problem.

2.3 Solution approaches/techniques

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

2.3.1 Local Search Based Techniques

Local search based techniques [14], such as Hill Climbing, Tabu Search and simulated annealing are commonly seen as belonging to metaheuristics [15]. Metaheuristics are higher

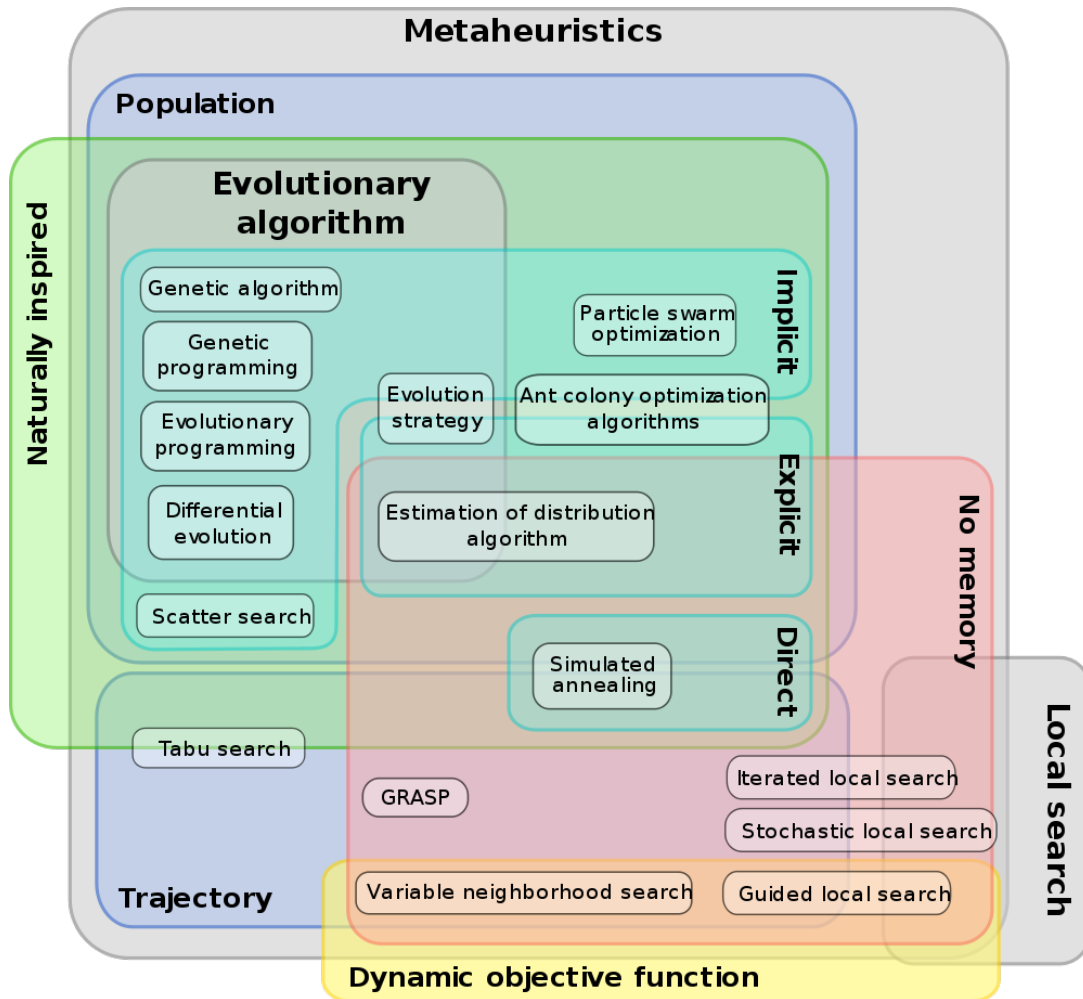


Figure 2.1: Metaheuristics classification

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

level heuristic methods that are designed to provide a sufficiently good solution to an optimisation problem. It is most commonly used in cases with incomplete information or limited computational power [16]. 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 until 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 [17] 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 [18] 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.

2.3.2 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 [19] simulated the natural evolutionary process by evolving and manipulating populations of solutions within the search space. Solutions are encoded as *chromosomes* (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 need to be properly defined. Consequently, the approach is usually more complicated to implement than local search based methods. The search strategy is fundamentally different from local search based methods in that multiple solutions are managed simultaneously, instead of just a single solution as seen in local search.

Memetic Algorithms

Memetic algorithms [20] aim 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 [21], 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

This chapter is for describing the room assignment problem, as well as a mathematical formulation of the problem. This includes the decision variables, as well as the objective function that is to be minimised.

3.1 Problem description

All fifth-year master's students in the Department of Engineering Cybernetics at NTNU can apply for a workplace on campus for working on their master thesis or specialisation project. The types of workplaces are divided into 5 categories: fixed workplace with a PC and monitor; fixed workplace with only a monitor; Fixed workplace with no equipment; drop-in workplace; No workplace. The students can apply for which category of workplace they want to be assigned to. Additionally, the students can state a maximum of one other student that they want to share office with. The definition for sharing an office for this problem is that the workplaces exist in the same room. The limitation of only being able to state one student is for reducing the complexity of the problem. This was done in consultation with my supervisor to make the problem easier to solve.

The algorithm should take the following criteria into consideration:

- Assign students to a workplace in the same category that they applied for
- Assign students who wish to share an office into workplaces in the same room.
- Assign students with the same supervisor to workplaces in the same room.
- Assign students in a fair way when there is not enough workplaces in a category for everybody.

If there are more workplaces available than there are students assigned, then it turns into a *unbalanced* problem. This is addressed by adding as many "dummy" students so as to make the problem balanced, and then solve for the balanced problem. The dummy students assigned to workplaces will in practice turn into empty workplaces.

It is more problematic when there are more student applying for a category of workplace than there are workplaces available. One cannot add "dummy" workplaces as they do not exist in real life. The question then becomes how to assign workplaces in a fair manner when the workplaces are indivisible? One answer to that question is to assign students to workplaces using *fair random assignment*, also known as a lottery. Assigning resources randomly via lottery is a surprisingly fair method [22]. Section 4.2.4 will go into more detail on how a lottery is combined with an optimisation algorithm in order to ensure fairness while still allowing room for optimisation.

3.1.1 Problem formulation

Our room assignment problem is formulated as the following:

Let n be the number of students and the number of workplaces. $F = (f_{ij})$, $D = (d_{kl})$, and $B = (b_{ik})$ are three $n \times n$ matrices with real numbers. f_{ij} is the flow between student i and student j , d_{kl} is the distance between workplaces k and workplaces l , and b_{ik} is the cost of placing student i at workplace k . N is the set $N = \{1, 2, \dots, n\}$. S_n is the set of all permutations $\phi : N \rightarrow N$

$$\min_{\phi \in S_n} \sum_{i=1}^n \sum_{j=1}^n f_{ij} D_{\phi(i)\phi(j)} + \sum_{i=1}^n b_{i\phi(i)} \quad (3.1)$$

The matrices F and D are symmetric with zeroes on the diagonal. The matrices F , D , and B are all nonnegative.

An element f_{ij} in the flow matrix F is a measure for how much it would cost to assign student i and student j in workplaces that are far from each other. In other words, a more positive number in f_{ij} would lead to a higher cost for the same distance. most elements in matrix F will have a zero value, except for those students that should be seated in the same room because of the criteria listed in the above section, Section 3.1. Our decision variable for the problem is a permutation ϕ from the set of all possible permutations S_n .

Several workplaces can exist in the same room. The workplaces that lie in the same room are defined to have zero distance from each other. All workplaces that do not exist in the same room will have a distance of one from each other. The reason for this is to reduce the complexity of our problem and therefore make it easier to solve. An additional reason for why this is done is because it would take a substantial amount of work to measure and categorise the distance between each workplace with every other workplace that exist on campus at NTNU.

The cost for assigning a student to a workplace in the same category that the student applied for is zero. However, if a student is placed in a different category then the cost becomes a constant C_b .

The simplifications above lead to having several permutations ϕ being equivalent in the sense that the cost from the objective function will be equal. The reason for this is that the

distance is zero for workplaces that exist in the same room, as well as only the category of workplace matter for computing cost. This results in making our problem easier to solve compared to the general QAP case described in Section 2.2.

Chapter 4

Implementation

The algorithm was chosen to be implemented in Matlab. One of the reasons for that is Matlab has many useful built-in functions for manipulating and accessing matrices, arrays, tables, and other data structures for containing large amounts of data. An additional reason for why Matlab was chosen is because the author of this report is most familiar with Matlab.

An alternative programming language that would have worked just as well would be Python with NumPy [23]. NumPy is a popular library for Python that adds support for n-dimensional arrays, as well as a collection of high-level mathematical functions to manipulate those arrays.

4.1 Data model

This section is for describing the data model to contain all the necessary data used during implementation of the algorithm in Matlab.

The input to our algorithm is two spreadsheets that is imported into Matlab as Matlab tables. The two Matlab tables is used to contain all of the necessary data that the algorithm uses. The "workplace" table contains data about every workplace on campus. The "student" table is used to contain data about every student, as well which room the student has been assigned to.

The data in the matrices F , D , and B as defined in Section 3.1.1 is moved into columns of the two Matlab tables to make implementation and management of the data easier. This is done in order to avoid keeping track of changes done to 3 different matrices and instead only keep track of 2 structured tables.

Figure 4.1 show an image of a workplace table with example data. The table contains data about which room, workplace number and what category each workplace is. Column 1 and 2 is used as a unique identifier for each workplace on campus. Column number 4 and 5 are only for keeping track of internal states that the algorithm uses.

The room category has been changed into an enumerated data type to make conditional checks in the code less complex. The following lists what category of workplace each number corresponds to:

1. Fixed workplace with a PC and monitor.
2. Fixed workplace with only a monitor.
3. Fixed workplace with no equipment.
4. Drop-in workplace.
5. No workplace.

The above categories will be later referenced to respectively as category 1, 2, 3, 4, and 5. The numbers also signifies the "desirability" of each category, with category 1 being most desirable and category 5 being least desirable. This ordering of the categories will come into play when there are not enough workplaces of a category for every student that applied. This is further explored upon in Section 4.2.1 and Section 4.2.2

	1	2	3	4	5	6	7
	ROM	plasnrr	TypeArbeidsplass	allocatedtruefalse	countroom		
94	Hovedbygget	111	2	1	15		
95	Hovedbygget	112	2	1	15		
96	Hovedbygget	113	2	1	15		
97	Hovedbygget	114	2	1	15		
98	Hovedbygget	115	2	1	15		
99	Hovedbygget	116	2	1	15		
100	Hovedbygget	117	2	1	15		
101	Hovedbygget	118	2	1	15		
102	Hovedbygget	119	2	1	15		
103	Hovedbygget	120	2	1	15		
104	Hovedbygget	121	2	1	15		
105	Hovedbygget	122	2	1	15		
106	Hovedbygget	123	2	1	15		
107	Hovedbygget	126	2	1	15		
108	GF313	99	2	1	8		
109	GF313	100	2	1	8		
110	GF313	101	2	1	8		
111	GF313	102	2	1	8		
112	GF313	103	2	1	8		
113	GF313	104	2	1	8		
114	GF313	105	2	1	8		
115	GF313	106	2	1	8		
116	GF318	108	2	1	2		

Figure 4.1: a screenshot of the workplace table with example data

Figure 4.2 shows a screenshot of the students table before running the assignment algorithm. The student's email is used as a unique identifier. Column 1 shows which category of workplace the student applied for. Column 2 designates who the students want to share office with, if any. Column 3 shows which supervisor the student has. Columns 4 to 6 shows which room the student has been assigned to, which have not been done yet for this screenshot. Column 7 to 11 are used for keeping internal states of the algorithm.

	1	2	3	4	5	6	7	8	9	10	11
	BehovForArbeidsplass	viiSitteMed	Faglaerer	allocatedroomname	allocatedplacenummer	allocatedplacetype	lucky1	lucky2	lucky3	newplacetyetoallocate	countoccurrence
1 student1@ntnu.no		1'student105@...	Adil Rashee...	<undefined>	<missing>	0	0	0	0	0	0
2 student10@ntnu.no		1''	Adil Rashee...	<undefined>	<missing>	0	0	0	0	0	0
3 student100@ntnu.no		1''	Adil Rashee...	<undefined>	<missing>	0	0	0	0	0	0
4 student101@ntnu.no		1'student103@...	Adil Rashee...	<undefined>	<missing>	0	0	0	0	0	0
5 student102@ntnu.no		1''	Adil Rashee...	<undefined>	<missing>	0	0	0	0	0	0
6 student103@ntnu.no		1''	Adil Rashee...	<undefined>	<missing>	0	0	0	0	0	0
7 student104@ntnu.no		1''	Adil Rashee...	<undefined>	<missing>	0	0	0	0	0	0
8 student105@ntnu.no		1''	Adil Rashee...	<undefined>	<missing>	0	0	0	0	0	0
9 student106@ntnu.no		1''	Anastasios L...	<undefined>	<missing>	0	0	0	0	0	0
10 student107@ntnu.no		1''	Anastasios L...	<undefined>	<missing>	0	0	0	0	0	0
11 student108@ntnu.no		1'student107@...	Anastasios L...	<undefined>	<missing>	0	0	0	0	0	0
12 student109@ntnu.no		1''	Anastasios L...	<undefined>	<missing>	0	0	0	0	0	0

Figure 4.2: a screenshot of the students table before room assignment

Figure 4.3 shows a screenshot for how the student table looks like after running the assignment algorithms, where the Columns 4 to 6 have now been populated with values instead of being empty.

	1	2	3	4	5	6	7	8	9	10	11	
	BehovForArbeidsplass	vilSitteMed	Faglaerer	allocatedroomname	allocatedplacenumbr	allocatedplacetype	lucky1	lucky2	lucky3	newplacetypeallocate	countoccurrence	
1	student110@ntnu.no	1"	Annette Sta...	Hovedbygget	"54"		1	1	0	0	1	14
2	student111@ntnu.no	1"	Annette Sta...	Hovedbygget	"55"		1	1	0	0	1	14
3	student112@ntnu.no	1"	Annette Sta...	Hovedbygget	"56"		1	1	0	0	1	14
4	student113@ntnu.no	1"	Annette Sta...	Hovedbygget	"57"		1	1	0	0	1	14
5	student114@ntnu.no	1"	Annette Sta...	Hovedbygget	"58"		1	1	0	0	1	14
6	student115@ntnu.no	1"	Annette Sta...	Hovedbygget	"59"		1	1	0	0	1	14
7	student116@ntnu.no	1"	Annette Sta...	Hovedbygget	"60"		1	1	0	0	1	14
8	student117@ntnu.no	1"	Annette Sta...	Hovedbygget	"61"		1	1	0	0	1	14
9	student118@ntnu.no	1"	Annette Sta...	Hovedbygget	"62"		1	1	0	0	1	14
10	student119@ntnu.no	1"	Annette Sta...	Hovedbygget	"63"		1	1	0	0	1	14
11	student12@ntnu.no	1"	Annette Sta...	Hovedbygget	"64"		1	1	0	0	1	14
12	student120@ntnu.no	1	student122@...	Annette Sta...	Hovedbygget	"65"	1	1	0	0	1	14

Figure 4.3: a screenshot of the students table after room assignment

an advantage of using Matlab tables is that the output of the algorithm can be exported to a human readable format. The desired columns can be selected in Matlab and then exported to a spreadsheet.

4.2 Implementation of the algorithm

This section is for giving a step-by-step overview of the whole algorithm.

The steps of the algorithm can be roughly summarised as the following:

1. Assign those who applied for category 4 and 5 to their desired workplaces
2. balancing the optimisation problem
3. Do a lottery to choose "lucky students" if there are not enough workplaces
4. pre-assign students
5. Do optimisation loops until some stopping condition.

The subsequent subsection will go into more detail on each step shown above.

4.2.1 Assign those who applied for category 4 and 5

All students who applied for category 4 and 5 can be assigned to their desired categories. There is no limit to how many students who can be assigned to workplace category 4 and 5, in other words drop-in workplaces and no workplaces respectively. All students have access to every workplace that are designated for drop-in. The drop-in workplaces operate on a "first come, first serve" basis, and have therefore no limit. Likewise there is no limit for assigning students the students who did not want a workplace to "no workplace".

4.2.2 balancing the optimisation problem

The next step will be to check if the optimisation problem needs to be balanced. This is done by computing the number of students who applied for each category and the number of each category of workplace available to be assigned. This is accomplished by simply iteration over the student table and counting the amount of students that applied for each category. The same is done for the workplace table to count the amount of workplaces available for each category.

Dummy students are added if there are less students who applied for a category of workplace than there are workplaces available. Nothing has to be done if the number of student and workplaces are equal. The most complicated behaviour happens if the number of students exceed the number of workplaces available for that category.

let x be the number of students who applied for a category, and let y be the number of workplaces available to be assigned, where $x > y$. That means $(x - y)$ students have to be removed for the problem to be balanced. One way to solve this fairly will be to arrange lotteries in which y winners will be drawn, consequently there will be $(x - y)$ losers in this lottery. The winners will be allowed to be assigned workplaces that they applied for, while the losers are forced to be assigned a workplace that is one level less desirable than the one they applied for.

This act of balancing the problem will have to be done 3 times, for respectively workplace category 1, 2, and 3. Balancing does not have to be done for category 4 as there is no limit to the amount of students who can be assigned to drop-in workplaces. The balancing ensures that as many students as possible are assigned to their correct categories when there are more students than workplaces of that category.

A problem that arises from doing lotteries is that some students may play the lottery multiple times. Some student may try to cheat the algorithm by applying for a more desirable category of workplace rather than the category they really needed, in order to get a workplace that they want. There must therefore be an option to reduce the probability of winning for the student who have participated in multiple lotteries, in order to make it more fair for the students who only participate in one lottery.

This is done by defining two new integer variables $lottery_a$ and $lottery_b$. The variable $lottery_a$ is for how many numbers of lots should be added for each of the students that have not yet participated in a lottery. The variable $lottery_b$ is how many numbers of lots should be added for each of the students that have participated in a lottery before. $lottery_b/lottery_a$ then determines the ratio for how less likely the students who have participated before have of winning. Because there now is multiple lots for the same students, care must be taken to remove every duplicate lot of students who have already won that lottery. This done to avoid drawing the same student multiple times in the same lottery and to ensure we end up with the correct number of students who have won, which is the number y .

an additional problem that arises from the lottery is that the algorithm becomes non-

deterministic. This may make the result obtained in this thesis harder to reproduce. Luckily Matlab allows for setting a seed to their random number generator. The results shown in Chapter 5 will therefore have the seed supplied to aid in allowing our result to be reproducible.

4.2.3 pre-assign students

The algorithm tries to pre-assign students according to some "smart" rules. This is done in order to reduce the amount of optimisation loops that has to be done before arriving at a locally optimal solution.

This is done by first computing the number of workplaces each room has and correspondingly computing how many students each supervisor has. The "workplace" table is then first sorted by the category of workplace, then secondly on the room with most workplaces. The same is done for the "student" table by first sorting on the applied category, then secondly sort on the supervisor with most students.

After the sorting has completed, the workplace on row 1 can be assigned to the student on row 1, workplace on row 2 can be assigned to the student on row 2, and so on until all workplaces has been assigned to a student. This result in an assignment where mostly the largest groups students who share a supervisor have been placed in the largest rooms.

4.2.4 optimisation loop

This subsection is for describing how the optimisation algorithm works, after all the pre-processing has been done.

The following is pseudocode for the optimisation part of the algorithm:

```

1: Sol_Best ← s0
2: Candidate_best ← s0
3: neighbourindex ← 1
4: while (not stoppingcondition()) do
5:   Sol_neighbors ← getneighboursof(Sol_Best,neighbourindex)
6:   for Candidate ∈ Sol_neighbors do
7:     if ComputeCost(Candidate) < ComputeCost(Candidate_best) then
8:       Candidate_best ← Candidate
9:   if ComputeCost(Candidate_best) < ComputeCost(Sol_Best) then
10:    Sol_Best ← Candidate_best
11:    neighbourindex ← 1
12:   else
13:     neighbourindex ← neighbourindex + 1
14: output Sol_Best

```

The above code is based on the steepest ascent hill climbing algorithm as described in Sec-

tion 2.3. The reason for choosing the hill climbing algorithm is because it is relatively easy to implement, and it can be adapted to become the other algorithm described in Section 2.3 if the need arises. s_0 is the input for the optimisation algorithm and it is the student table. $Candidate_best$ is the current best candidate solution from the neighbourhood. Sol_best is the current best solution.

The hill climbing method requires a definition for the neighbourhood of a solution. It is not inherently apparent what a neighbourhood as this depends on both the specific problem and what the solution to the problem is. Focusing on the student that currently contributes most to the cost did seem like a good starting point. The neighbourhood of a solutions for this problem is therefore defined as the set of all permutations where the current most costly students has switched workplaces with another possible student. A switch is only possible if both students are assigned to the same category of workplace, this is to ensure that the solution is still feasible.

The $ComputeCost()$ is the function that computes the cost. The exact implementation is different from the objective function as defined in Section 3.1.1. This is because the information contained in the matrices has been moved into columns on the student and workplace tables. It works by counting the number of occurrences for each constraint then multiplying that with a weighting coefficient. The exact number for the weighting coefficients can be changed by the end user in order to change the prioritisation of the constraints. By setting a weighting coefficient to zero leads to that constraint being disregarded as it will no longer contribute to the cost. The cost from each constraint is then summed into a total cost and returned.

The algorithm iterates and computes the cost of each candidate solution $Candidate$ in the neighbourhood and compares with the current best candidate $Candidate_best$ in the neighbourhood. If the cost is lower, then that candidate becomes the new best candidate.

It then does a new comparison to check if the best candidate solution $Candidate_best$ has a lower cost than the the current best solution Sol_best .

If that is not the case the $neighbourindex$ gets incremented by one. This changes what the $getneighboursof()$ function returns. It instead returns the set of all possible switches with the **second** most costly student. $neighbourindex$ can be any value from 1 to the number of students in the problem. Summarised $getneighboursof(\phi, i)$ takes in a single permutation ϕ of students assigned to workplaces and $neighbourindex$ i , and return a set of all permutations where the i th most costly student is switched with every other possible student.

The optimisation part of the algorithm takes up a majority of the runtime for the whole algorithm. Figure 4.4 shows how much time it took to run the functions in the algorithm, where the function $computecost()$ is notable for taking up 91% of the runtime. This is explained by quadratic term of the objective function, as the double summation is implemented as a nested for loop. This leads to the body of the for-loop in worst case being called n^2 times where the n is the number of students.

The $stoppingcondition()$ is implemented to return true after reaching a self defined max-

Children (called functions)







Function Name	Function Type	Calls	Total Time	% Time	Time Plot
computecost	function	377	66.469 s	91.0%	
tabular.subsref	function	37184	2.830 s	3.9%	
getrowmostcost	function	4	0.730 s	1.0%	
tabular.subsasgn	function	933	0.516 s	0.7%	
readtable	function	2	0.481 s	0.7%	
getrowcost	function	2	0.380 s	0.5%	
computenumber	function	2	0.363 s	0.5%	
getneighborsof	function	2	0.225 s	0.3%	
tabular.numArgumentsFromSubscript	function	36234	0.066 s	0.1%	
tabular.sortrows	function	3	0.042 s	0.1%	
tabular.horzcat	function	2	0.034 s	0.0%	
table.table>table.table	class method	8	0.033 s	0.0%	
...ategorical>ategorical.ategorical	class method	3	0.023 s	0.0%	
ategorical.cellstr	function	278	0.006 s	0.0%	
tabular.dotParenReference	function	6	0.003 s	0.0%	

Figure 4.4: a screenshot of the Matlab profiler after running the program

imum iteration or if the best solution has not changed after another self defined number of iterations. The function then finally return the current best solution Sol_{best} .

Chapter 5

Results and Discussion

This chapter is for presenting the results and discussion that were obtained by using the algorithm that was described in Chapter 4. The first case in Section 5.1 will be where we have a balanced problem. The second case in Section 5.2 is where we have an initially unbalanced problem and where we have to balance it by doing a lottery. The weighting coefficients wc_1 for students being assigned to the wrong category of workplace is 200, the weighting coefficients wc_2 for not assigning students who want to share an office is also 200, the weighting coefficients wc_3 for not assigning students who share a supervisor to the same office is 10. The reason the low value of the last weighting coefficients is that it has relatively low importance compared to the students who have explicitly stated they want to share an office.

5.1 Case 1

For the balanced case, we have 180 students in total. 95 students want a workplace of category 1, 31 students want a workplace of category 2, and 20 students want a workplace of category 3. 34 students want either category 4 or 5. The students who want categories 4 and 5 are easily handled as there are no restriction for assigning them their wanted type of category. 15 students have designated that they want to share an office with another student.

The number of workplaces available for category 1 is 95, category 2 is 32, and category 3 is 20. This is the ideal case as there are very few practical instances where the number of workplaces perfectly match the number of students.

Figure 5.1 show the total cost from the objective function as well as the cost from each term. $cost_1$ corresponds to the cost computed with wc_1 , $cost_2$ corresponds to the cost computed with wc_2 , and $cost_3$ corresponds to the cost computed with wc_3 . The total cost of the table before running the optimisation loop was 1320, and 520 after running the loop. $cost_2$ sharply decreases, while $cost_3$ decreases more slowly. The cause for that is there are few students who contribute greatly to the cost of $cost_2$, this contrasts with $cost_3$ where many

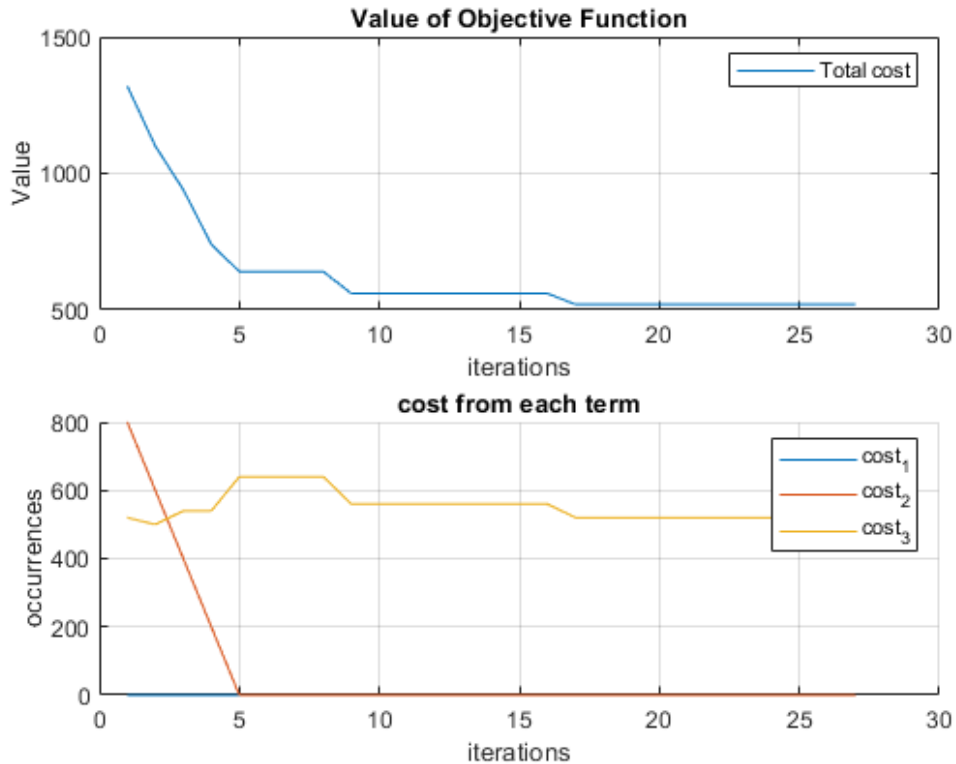


Figure 5.1: Plot of case 1 showing total and cost of each constraint as a function of iterations

students may share a supervisor, but each contributes only a little bit to the cost.

The result after running the optimisation algorithm is that $cost_1$ and $cost_2$ were able to be driven to zero, but $cost_3$ was not.

5.2 Case 2

The second case is where we have an unbalanced problem where the number of students is more than the number of workplaces. The algorithm now becomes non-deterministic as lotteries have to be arranged to balance the problem. The algorithm will be run three times with different seeds on the same input data. This is in order to illustrate how the lottery impact the cost from the objective function for both before and after the optimisation loop.

For the case 2 we still have the same students as in case 1. That is 180 students in total. 95 students want a workplace of category 1, 31 students want a workplace of category 2, and 20 students want a workplace of category 3. 34 students want either category 4 or 5. The students who want categories 4 and 5 are easily handled as there are no restriction for assigning them their wanted type of category. 15 students have designated that they want to share an office with another student.

What has changed is that the number of workplaces has been reduced. The number of

workplaces available now for category 1 is 92, category 2 is 29, and category 3 is 18. This means that there will be 3 losers in the first lottery, 5 losers in the second lottery, and 7 losers in the in the final and third lottery. The people that lost the first lottery will be added as participants in the second lottery, and the losers in the second lottery will be added as participants in the third lottery.

The seed for the random number generator in Matlab can be set with the command `rng(n)`, where n is a positive integer.

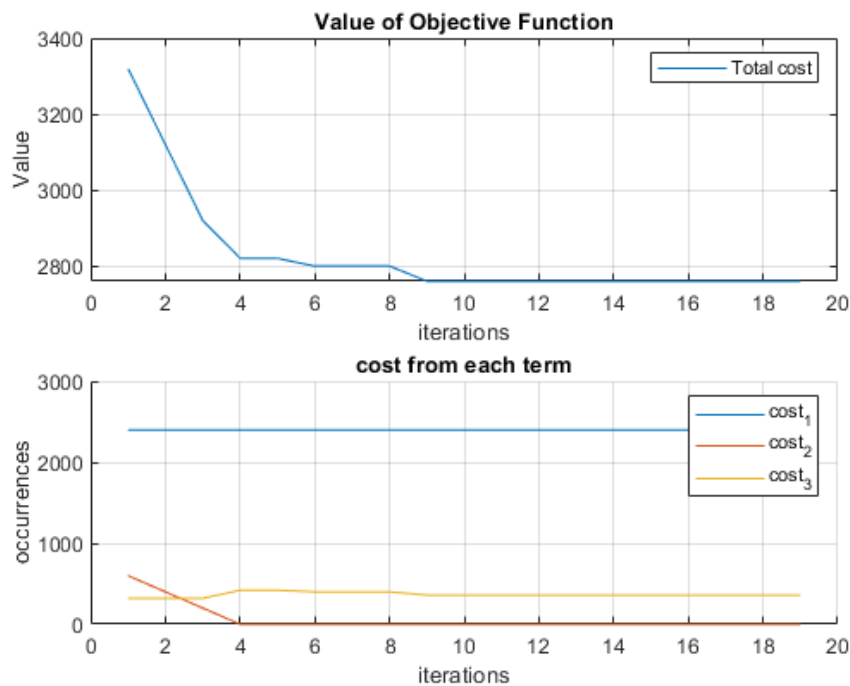


Figure 5.2: Plot of case 2, with seed 1 showing total and cost of each constraint as a function of iterations

Figure 5.2 shows the plot cost for case 2 when the seed is set to 1. The cost before the optimisation loop is 3320, and 2760 after the optimisation. The number of loops before the cost stabilised is 9

Figure 5.3 shows the plot cost for case 2 when the seed is set to 7. The cost before the optimisation loop is 3620, and 2620 after the optimisation. The number of loops before the cost stabilised is 15

Figure 5.4 shows the plot cost for case 2 when the seed is set to 11. The cost before the optimisation loop is 3020, and 2200 after the optimisation. The number of loops before the cost stabilised is 20

The start and end cost, as well as the number of iterations for each run is different. The lowest end cost across the three runs is 2200 for the third run.

The $cost_1$ for all three runs in the second case is constant and has a non-zero value. This is because the original problem was unbalanced. There were simply not enough workplaces

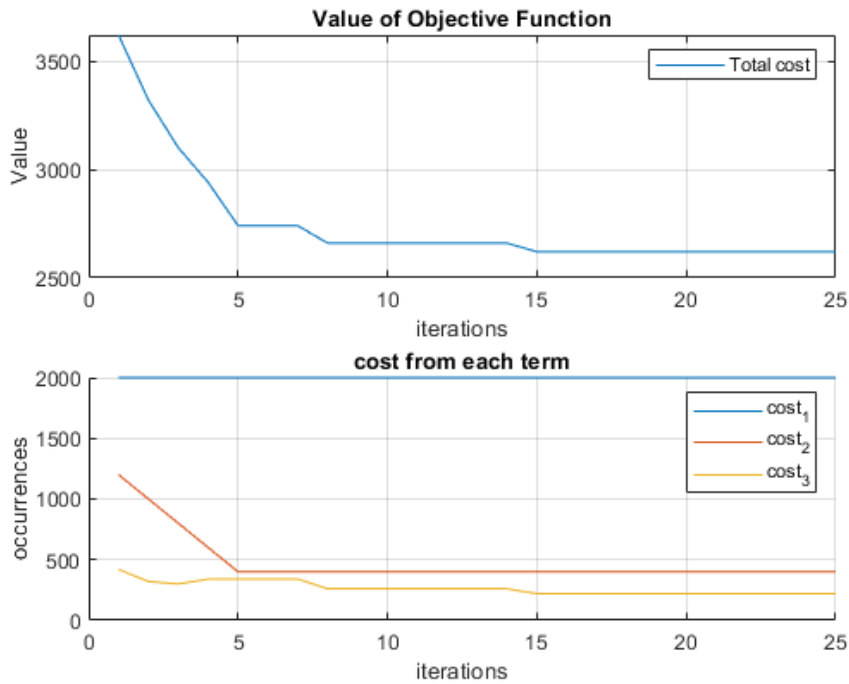


Figure 5.3: Plot of case 2, with seed 7 showing total and cost of each constraint as a function of iterations

for every student.

The non-deterministic nature of the algorithm makes it hard to draw any clear conclusion for how good the algorithm is as the results can change for every run even with the same input data. Still the optimisation algorithm was able to lower the starting cost for each run before stopping at a local optimum.

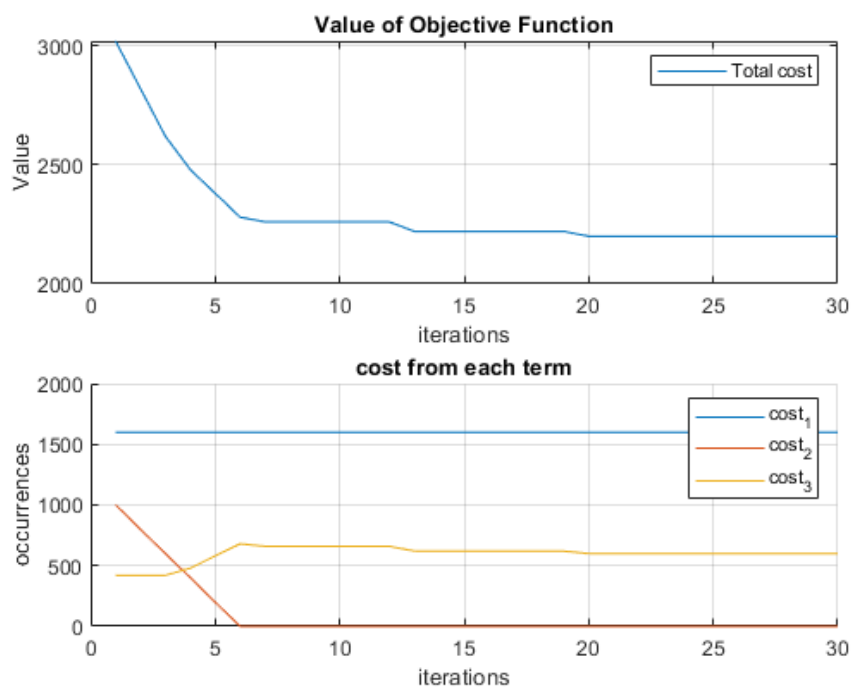


Figure 5.4: Plot of case 2, with seed 11 showing total and cost of each constraint as a function of iterations

Chapter 6

Conclusion

An overview of the theory and solution approaches for the quadratic assignment problem was given in this thesis. We managed to formulate our room assignment problem as a quadratic assignment problem. In addition, the Steepest-Ascent hill climbing algorithm was designed to fit our problem, as well as implemented and tested.

The results showed that our algorithm was able to minimise the objective function until it stopped at a local optimum. The QAP problem in general is a NP-hard and very hard to solve for a global solution. It is therefore hard to compare how close our local optimal solution is to the global solution, as that requires that we know the global solution.

The lottery ensures fairness by giving every student a chance to get the category of workplace they applied for. This however brings the disadvantage of making the algorithm non-deterministic. The results from Section 5.2 shows that the lottery may significantly change the final cost of the solution, even when the input data is constant.

The objective function by itself is a handy way of measuring the quality of a solution, even for human made solutions. An example usage can then be to compare hand-made solutions with solutions that the algorithm outputs to compare if the algorithm is better or worse than hand-made solutions at making assignments.

The objective of this thesis was to design and implement an algorithm that was able to assign students to workplaces automatically. This was achieved in the sense that the algorithm was able to solve for a local optimum.

6.1 Further Work

There is still some room for further improving the result obtained in this thesis. Some of the more advanced methods described in Chapter 2 could be implemented and tested to see if they yield better results for our room assignment problem.

The limitation that each student could only designate one other student they want to share an office with could be removed. The functionality could instead be expanded to be able to designate any number of students to share an office with. This would most likely however make the problem harder to solve for.

There will likely be a lot of work to integrate the algorithm with the Department of Engineering Cybernetics current existing IT-solutions before the algorithm can be put into operational use. This is however outside the scope of this master's thesis, as the goal was only to design and implement the algorithm.

Bibliography

- [1] A. Schrijver, *Combinatorial Optimization: Polyhedra and Efficiency, Volum 1*. Berlin, Germany: Springer, Feb. 2003, ISBN: 978-3-54044389-6. [Online]. Available: https://books.google.no/books?id=mqGeSQ6dJycC&redir_esc=y.
- [2] B. Gavish and S. C. Graves, ‘The Travelling Salesman Problem and Related Problems,’ *Massachusetts Institute of Technology, Operations Research Center*, Jul. 1978. [Online]. Available: <https://dspace.mit.edu/handle/1721.1/5363>.
- [3] A. C.-C. Yao, ‘New Algorithms for Bin Packing,’ *J. ACM*, vol. 27, no. 2, pp. 207–227, Apr. 1980, ISSN: 0004-5411. DOI: 10.1145/322186.322187.
- [4] H. M. Salkin and C. A. De Kluyver, ‘The knapsack problem: A survey,’ *Naval Research Logistics*, vol. 22, no. 1, pp. 127–144, Mar. 1975, ISSN: 0028-1441. DOI: 10.1002/nav.3800220110.
- [5] W. Ke, *AI for Exam Allocation*, Dec. 2020.
- [6] X.-S. Yang, *Engineering Optimization: An Introduction with Metaheuristic Applications*. Chichester, England, UK: Wiley, Jul. 2010, ISBN: 978-0-47064041-8. [Online]. Available: https://books.google.no/books?id=kTi6ul2g9VYC&dq=metaheuristic&lr=&hl=no&source=gbs_navlinks_s.
- [7] D. S. Hochba, ‘Approximation Algorithms for NP-Hard Problems,’ *SIGACT News*, vol. 28, no. 2, pp. 40–52, Jun. 1997, ISSN: 0163-5700. DOI: 10.1145/261342.571216.
- [8] T. C. Koopmans and M. Beckmann, ‘Assignment Problems and the Location of Economic Activities,’ *Econometrica*, vol. 25, no. 1, pp. 53–76, Jan. 1957, ISSN: 0012-9682. [Online]. Available: <http://www.jstor.org/stable/1907742>.
- [9] D. Optimierung, R. E. B. Er, A. C. Ela, R. E. Burkard and L. S. Pitsoulis, ‘The Quadratic Assignment Problem,’ *ResearchGate*, Jul. 1998. DOI: 10.1007/978-1-4613-0303-9_27.
- [10] L. Steinberg, ‘The Backboard Wiring Problem: A Placement Algorithm,’ *SIAM Rev.*, vol. 3, no. 1, pp. 37–50, Jan. 1961, ISSN: 0036-1445. [Online]. Available: <http://www.jstor.org/stable/2027247>.
- [11] J. W. Dickey and J. W. Hopkins, ‘Campus building arrangement using topaz,’ *Transp. Res.*, vol. 6, no. 1, pp. 59–68, Mar. 1972, ISSN: 0041-1647. DOI: 10.1016/0041-1647(72)90111-6.
- [12] R. E. Burkard, ‘Quadratic Assignment Problems,’ in *Handbook of Combinatorial Optimization*, New York, NY, USA: Springer, New York, NY, Jul. 2013, pp. 2741–2814, ISBN: 978-1-4419-7996-4. DOI: 10.1007/978-1-4419-7997-1_22.

- [13] S. Sahni and T. F. Gonzalez, 'P-complete approximation problems. J ACM 23(3): 555-565,' *J. ACM*, vol. 23, no. 3, pp. 555–565, Jul. 1976, ISSN: 0004-5411. DOI: 10.1145/321958.321975.
- [14] 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.
- [15] 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.
- [16] 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.
- [17] 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.
- [18] 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.
- [19] 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.
- [20] 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.
- [21] 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.
- [22] E. Budish, Y.-K. Che, F. Kojima and P. Milgrom, 'Designing Random Allocation Mechanisms: Theory and Applications,' *Am. Econ. Rev.*, vol. 103, no. 2, pp. 585–623, Apr. 2013, ISSN: 0002-8282. DOI: 10.1257/aer.103.2.585.
- [23] *NumPy Reference — NumPy v1.20 Manual*, [Online; accessed 14. Jun. 2021], Mar. 2021. [Online]. Available: <https://numpy.org/doc/stable/reference>.

