Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Bakken, Edvard Gjessing

Does sequence affect grades?

A quantitative analysis of graded Python source code and their relative position in a sequence.

Master's thesis in Informatics
Supervisor: Sindre, Guttorm

June 2020

NTNU
Kunnskap for en bedre verden

Bakken, Edvard Gjessing

Does sequence affect grades?

A quantitative analysis of graded Python source code and their relative position in a sequence.

Master’s thesis in Informatics
Supervisor: Sindre, Guttorm
June 2020

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Abstract

The aim of this master thesis was to investigate the relationship between the sequence (i.e order of presentation) of computer science student's submissions (i.e Python source code) and their given grades in tertiary education. After students submit their exam answers, a professional rater grades these submissions in a specific sequence. This thesis investigates if different permutations of this sequence has any effect on the grades and if an *optimal* sequence (i.e optimal permutation) increase values such as *fairness*, *validity* and *reliability* in the form of *inter*-rater and *intra*-rater reliability. The second goal of this thesis was to provide a prototype to automate the generation of *optimal* sequences given a set of source codes.

Scientific literature on psychology and education shows that monotonic work over a significant period of time decreases a humans ability to perform and increases irrationality and invalidity. This study utilize quantitative data from digital exams in the computer science course TDT4127 at the Norwegian university of science and technology (NTNU) in the form of Python source code to conduct 3 different experiments (Experiment 1: N = 10, Experiment 2: N = 40, Experiment 3: N = 0).

The results from this thesis suggests that there was no significant relationship between the independent variable *sequence* and the dependent variable *grade* as no significant effects were observed. However, this does not directly imply that there is no relationship at all. Independently of these results, the suggested prototype is documented in detail and the results of the conducted experiments suggests that it is able to generate *optimal* (as defined in this thesis) sequences purely based on source code as input. This thesis also provide a strong suggestion that using Greedy-String-Tiling to calculate the similarity between source code is similar to human measurements of similarity.

Sammendrag

Målet med denne masteroppgaven var å undersøke forholdet mellom sekvensen (dvs. rekkefølgen på presentasjonen) av IT-studenters eksamensinnleveringer (dvs. Python-kildekode) og gitte karakterene i tertiær utdanning. Etter at studenter har levert besvarelsene til en eksamen, gir en profesjonell sensor besvarelsene en karakter i en gitt sekvens. Denne oppgaven undersøker om forskjellige permutasjoner av sekvenser har noen innvirkning på karakterene, og hvis en *optimal* sekvens (dvs. optimal permutasjon) øker verdier slik som *fairness*, *validity* og *reliabilitet* i form av *inter*-rater og *intra*-rater reliabilitet. Det andre målet med denne oppgaven var å skape en prototype som automatiserer genereringen av *optimale* sekvenser gitt et sett med kildekoder.

Vitenskapelig litteratur om psykologi og utdanning viser at monotont arbeid over en betydelig periode reduserer menneskers evne til å utføre arbeid og skaper irrasjonalitet og ugyldighet. Denne masteroppgaven bruker kvantitative data fra digitale eksamener i IT-faget TDT4127 ved Norges teknisk-naturvitenskapelige universitet (NTNU) i form av Python-kildekode for å utføre 3 forskjellige eksperimenter (Eksperiment 1: N = 10, Eksperiment 2: N = 40, Eksperiment 3: N = 0).

Resultatene fra denne oppgaven antyder at det ikke var noen signifikant sammenheng mellom den uavhengige variabelen *sekvens* og den avhengige variabelen *karakter* ettersom ingen signifikante effekter ble observert. Dette innebærer imidlertid ikke en direkte antydning om at det ikke eksisterer noe forhold i det hele tatt. Uavhengig av disse resultatene er den foreslåtte prototypen dokumentert i detalj, og resultatene fra de gjennomførte eksperimentene antyder at den er i stand til å generere *optimale* (som definert i denne oppgaven) -sekvenser rent basert på kildekode. Denne oppgaven presenterer også sterke antydninger til at bruken av Greedy-String-Tiling for å beregne likheten mellom kildekode ligner på menneskelige målinger av likhet.

Acknowledgements

This thesis was written the autumn of 2019 and the spring of 2020 at the Department of Computer Science (IDI) at the Norwegian University of Science and Technology (NTNU).

First of all, I give my thanks to my thesis advisor professor Guttorm Sindre of the Norwegian the Norwegian university of science and technology (NTNU). His advise, guidance and engagement in discussions allowed for this paper to take form. Secondly, I must express my very profound gratitude to my long time friend and mathematician Eirik Agnalt Østmo for his engagement in the thesis, advice and support throughout the process. Without him, the contributions of this thesis would definitely have been of lesser value.

I would also like to thank Associate Professor Magnus Lie Hetland for technical guidance, Andrea Bach and my colleges at UNIBOYZ for providing constant feedback, validation and emotional support.

Finally, I must express my very profound gratitude to my family and friends for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis.

Thank you.

Edvard Gjessing Bakken
Trondheim, June 10, 2020

Contents

List of Figures

List of Tables

Abbreviations

GST	Greedy String Tiling
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
OPTICS	Ordering Points To Identify the Clustering Structure
AST	Abstract Syntax Tree
EPS	ϵ, Epsilon
MinPts	Minimum amount of necessary neighbours
TSP	Traveling salesman problem

Part I

Introduction and Methodology

Chapter 1

Introduction

This chapter presents the motivation for writing this thesis, an overview of the scope, the research questions and a general presentation of the thesis approach. The chapter ends with a presentation of the thesis structure.

1.1 Motivation

Since 2013, Norwegian universities and academic institutions have implemented digital assistance tools to for conducting final exams. These tools have proven to decrease manual labor, increase security and streamline the overall process. The tools consists of a range of specialized computer software-products that contain functionality to create, complete and grade exams.

Digital tools are already implemented to reduce monotonous tasks such as grading multiple-choice questions and in some cases, scanning written essays and suggesting a grade, even when the content is highly advanced. However, while digital tools do remove a considerable amount of manual labor, they are not sophisticated enough to automatically grade all kinds of student submissions to completely substitute humans. Many open-ended questions with multiple levels of correctness and advanced theory are difficult to evaluate. Therefore, in many cases, humans still have to manually evaluate submissions even though it means evaluating hundreds of student submissions in a short amount of time. When comparing *digital* with *non-digital* exams, the difference does not lie in human intellectual insight and decision-making, but instead cover areas such as logistics, security and submitting digital documents instead of physical paper.

When confronted with monotonic tasks, humans are prone to make mistakes and become influenced by conscious and subconscious effects. A research paper titled "Sequential effects in Olympics synchronized diving scores" by Kramer (2017), presents different aspects of human traits when observing sequences. It states that if given the task of evaluation, such as a judge scoring Olympic divers or a professor grading open-ended student submissions, there is no (objectively) correct answer, merely our own opinion. Even when adhering to specific criteria or guidelines, scores can still differ significantly from each other. Subjective measures of objective input often output different results between individuals (Muckler and Seven, 1992). Kramer investigates if our current opinion is influenced by the surrounding context, as if our judgment of one diver is influenced by the previous diver, creating a *contrasting effect.* Kramer concludes that "Olympic judges show contrast effects when scoring synchronized divers. This bias causes a decrease in scores for athletes that follow a high-scoring pair and an increase in scores for those who follow a low-scoring pair. Such a bias represents an unfair (dis)advantage for divers".

Spear (1997) conducted an experiment where she gave 336 teachers the task of evaluating three different written works differing in quality and authorship. The experiment considered the received grade for each work based on its position in the sequence of the three. To measure the existence of contrast effects in grading, she measured the correlation between the given grade and the paper's order. The six possible permutations were seen the same number of times in order to measure differences. Spear concludes that work of high quality was favored when following work of lower quality, and work of low quality was assessed more harshly when following, rather than preceding, work of contrasting quality. Spear also shows that the number of preceding contrasting items amplifies this effect. Contrasting effects (previous observations inversely favours following units), has been observed in many other areas such as Human-Agent Interaction (Ramesh et al.), judging vocal competitions (Page and Page, 2010), speed dating (Bhargava and Fisman, 2014) and grading essays (Zhao et al., 2017).

Additional to contrasting effects, *distinction bias* is also suggested to influence our subjective judgment of objects in sequence. Distinction bias addresses the tendency to compare items close to each other differently than when evaluating them separately and within different time-periods. Therefore, equal items might be evaluated differently based on their position in a sequence. Aslett (2006) states that mental and physical fatigue either due to monotony, lack of interest, or lack of sleep can have severe implications with regards to task performance and accuracy. When evaluating sequences, all objects should be evaluated with equal measures to assure fairness and validity. This is of grave importance when evaluating performance such as when grading students in academia. However, research by Aslett (2006), Klein and El (2003) and Wolfe et al. (2001) suggests that this is not what is carried out. Evaluating large amounts of objects creates inconsistencies of grades based on their position in the sequence. One of the effects of this is that identical objects are evaluated differently and receive different grades purely based on their position in a sequence.

Contrast effect and *distinction bias* is a subset of the total set of biases affected by the sequence, hence *sequence effects.*

It is evident that internal and external forces influence our decisions. Cognitive biases affect our ability to perform, decreasing our mental functionality which thereby inflicting our reliability. In academia, reliability and validity are of grave importance to induce a fair system and by reducing the effects of cognitive biases, it directly increases fairness. This correlation has made the pursuit of reducing cognitive biases an important topic and is therefore the topic of this thesis.

Since the order of a sequence is often an available variable we can manipulate, this raises the question: Is it possible to manipulate the sequence of submissions before they are graded in order to reduce bias? In theory, if we decrease the amount of contrasting objects directly pursuing each other, smoothing out the contrast between the objects, it should result in a reduction of contrasting effects when they are graded. Similar to this, if we reorder the sequence in such a fashion that identical or similar objects directly follow each other, they will be evaluated with equal measures. Manipulating the order of the sequence to obtain an *optimal* order is the goal of this thesis. The optimal order is in this thesis based on two properties:

1. Equal answers receives equal grades
2. Contrasting effects are reduced

In this thesis, I will attempt to assist in the improvement of NTNU's digital exam solution in regard to reducing the effects of human biases affecting fairness in grading of student submission to final exams.

1.2 Scope of the Thesis

This thesis will focus mainly on discussing validity, fairness, and reliability considering grades given to submitted answers in computer science classes. The answers are in the form of source code in the programming language Python. The leading research will be conducted using students simulating professional raters in the task of grading real submitted final exam answers. The environment and participants of the experiment are concentrated at the Norwegian University of Science and Technology (NTNU) in Trondheim, Norway. I will conduct experiments regarding the sequence of submissions that are graded to extract patterns and gain insight into possible ways of improvement. I also discuss a strategy of automatically ordering source code into different orders and discuss the results. This thesis is not meant to replace the current system, but rather provide insight and suggest further research. The source code for the provided algorithm developed for this thesis can be found at the repository [1]. This thesis also provides insight into different biases affecting academia. It is essential to emphasize that this thesis does not cover all biases, but merely a small subset.

[1]Source code available at `https://bitbucket.org/EdvardGB/sequence_artefact/src/master/`

The following source code is an example of Python source code used in this thesis. This example is extracted from *source 1* presented in 2.2.1.

```
def find_const(strg):
    if strg[0] != "-":
        return strg[0]
    else:
        return strg[:2]
```

1.3 Research Goal, Research Questions, and Hypotheses

Goal When grading student submissions in computer science courses, research if an optimal sequence reduce sequential effects and develop a method to reduce sequential effects by automation

RQ 1 Does the sequence of submissions have any effect on the given grades?

- **RQ 1.1** Does an optimal sequence reduce the consequences of contrast effect?
- **RQ 1.2** Does an optimal sequence reduce the number of inconsistencies in equal grading of equal submissions?
- **RQ 1.3** Does an optimal sequence increase inter-rater or intra-rater reliability?

RQ 2 Given a set of Python source codes, can we automatically generate an optimal sequence?

- **RQ 2.1** Can GST be used to cluster equal submissions with similar grades?
- **RQ 2.2** Can we manipulate any sequence to become optimal by automation?

1.4 Ethical Considerations

As this thesis used human participants for experiments, ethical considerations is an important topic. All participants was fully informed and voluntarily participated in the experiments and agreed for the empirical data generated to be used in this thesis. No usage of offensive language of formulation were used before, under or after the experiments. All participants were anonymous and no data were collected that can be used to identify personal information about the participants.

The material used in the experiments was real student submitted programming assignments for exam answers. Permission was granted from the instructors involved to use the material. All submitted material used where anonymous, as in identification number, name or comments which can be used to identify a person was not available or used. All material is independent and only sub-parts of the total exam.

1.5 Contributions

The contributions of this thesis is mainly the exploration of the relationship between sequence of submissions and the grades given to them. It is explored by experiments and the design of an automatic method of generating *optimal* sequences. The results are described in chapter 9. Conclusions based on these results might lay the foundations for further research in the area.

- A better understanding of the relationship between grading sequence and evaluation of student submissions.
- A prototype to automatically generate a grading sequence reducing the influence of sequence effects in computer science courses.
- A suggested method for reducing inconsistencies when grading equal submissions with unequal grades.

1.6 Literature review

A background study of related literature and similar dissertations was performed prior to this research. The outcome of the literature review laid the foundations for the methodology and contribution. The sources used to retrieve the relevant literature where *scholar.google.com* and *oria.no*. When researching these sources, the following tags were used, either in combination or included in larger queries:

- Rater bias
- Sequence effects
- Reliability
- Clustering
- Source code similarity
- Grades

1.7 Thesis Structure

The remainder of this thesis is structured as follows:

Part II – Literature review and problem elaboration: This chapter presents previous literature reviewed prior to the completion of the developed algorithm and the writing of this thesis. It includes insight into different relevant aspects regarding the topic of this thesis and further elaboration of the challenges faced when developing the algorithm.

Part III – Background theory, related work and system design: This chapter provides technical details and insight into the development of the algorithm and the architecture of the algorithm itself. It also presents and overview of related work with similar solutions to similar problems.
Part IV – Experiments, Results, Discussion, and Conclusion: This chapter presents a detailed overview of the conducted experiments, the results and a discussion of the results. The chapter ends the thesis with a conclusion and further needed development.

Chapter 2

Research Methodology

This chapter presents the approached research process used in order to answer the research questions described in 1.3. The research process is defined by a strategy, a data generation method and an analysis approach. The Figure 2.1 illustrates an overview of different approaches to conduct the research process (Oates, 2005, chap. 3).

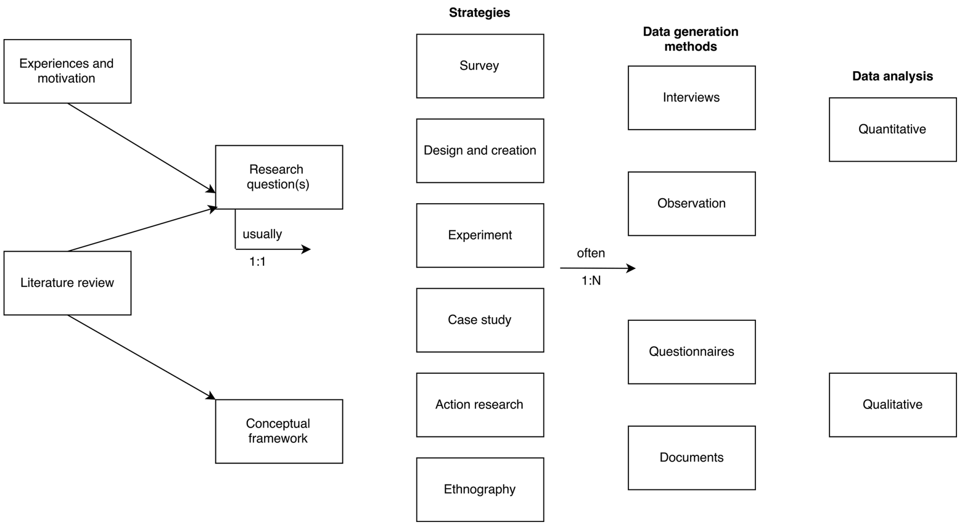


Figure 2.1: Model of the research process

2.1 Research Strategies

The choice of research strategy was based on how to achieve valid conclusions to the research questions. In order to achieve this, the combination of the strategies *Experiment* and *Design and Creation* was chosen as they provide methods to test various approaches and find optimal solutions iteratively. Experiments were used to identify the correlations between the variables, while Design & Creation were used to create a sequence-reordering-algorithm prototype. Other strategies such as a *case study*, *action research* or *ethnography* were considered, but given the limited access to case participators, time-restraints and given the exceptional circumstances caused by the COVID-19 virus pandemic occurring simultaneous with the creation of this thesis, they were not suitable.

2.1.1 Experiments

The term *Experiment* is not any arbitrary piece of field research. It specifically refers to a research strategy to identify the relationship between cause and effect. This is done by manipulating *independent* variables and observe the changes of *dependent* variables to identifying the in-between correlation (Oates, 2005, chap. 9). The variables is defined by a *hypothesis* which is a sentence stating a predicted correlation in the form of:

- "Increasing variable X, causes variable Y to decrease"
- "Activity X causes the effects Y".
- "Removing the *wings* from a plane reduces *drag* and increases *flight speed*"

By changing the variable X, we observe the effects of variable Y. If variable Y changes in a predictable pattern we can be fairly confident that X and Y are correlated. However, if Y does not change or it changes in an unpredictable pattern, then Y does not correlate with X. To conclude with a strong causal effect, the repetition of an experiment under various conditions is necessary to isolate the variables. This is done by including a random element in choosing the participants, choosing anonymous participants, using control and test groups, and/or eliminating or reducing external variables that may interfere with the experiment.

In a scientific experiment, a **null hypothesis** ($H0_i$) is the hypothesis that there is no correlation between X and Y. To state that there is *no* correlation between the two is practical, because by disproving this directly implies that there actually *is* a correlation. If the null hypothesis is not disproven, it implies that any difference observed in the data would be due to errors is measurements, samples or purely due to random chance. However, if the null hypothesis *is* disproven, it is replaced by an alternative hypothesis ($H1_i$) which proposes the influence of a non-random factor and the *dependent* variable actually is dependent on the *independent* variable.

2.1.2 Design and Creation

Design and Creation is an iterative process to design and develop a product and applying it to a domain. It is typically a problem-solving approach, however it is rarely a fully implemented system that can be used immediately without further research. Therefore the design, also called *artefact*, often only serve as a prototype or a demonstration to illustrate functionalities or ideas. It can thereafter be further researched or implemented in full-scale projects (Oates, 2005, chap. 8).

In this thesis, I design and evaluate a specific combination of already developed algorithms combined into a single product similar to a sorting algorithm. This product/algorithm is therefore referred to as the *artefact* in the following chapters and serves as a part of the thesis contribution.

2.2 Data Generation Methods

Data generation is the activity of generating empirical data. To conduct a valid analysis and construct a grounded conclusion, relevant and valid data is necessary. For this thesis, data that can provide insight into the causes and effects in the process of grading student submissions are necessary. Therefore, I have chosen a quantitative approach which allows for the collected data to be easily translated into numerical values for later ease of measure.

In this thesis, student final exam submissions is hereby referred to as "submissions" and should not be confused with the data collected in the experiments even though the participants of all experiments were also students. For clarity, Experiment 1 *submissions* refer to the student submissions to the final exam extracted from the data sources described in 2.2.1, and not the empirical data collected in the experiment. The empirical data collected in the experiment are numerical values from 0 to 5, as the participant's task was to assign grades. I conducted two different experiments wherein both experiments participants were anonymous students from NTNU with knowledge of the programming language Python.

2.2.1 Data sources

The described experiments in 2.2.2 and 2.2.3 utilize previously collected *data material* in order to generate *empirical data* later analysed in this thesis. The data material is in the form of Python source code as described in 1.2 and collected from real student submissions to final exams. The exam was conducted digitally, and the submissions are therefore easily obtainable as individual digital documents. The actually received grade from the professional rater(s) grading the exams is also included the individual documents. The used material is extracted from the following sources:

1. 53 submissions, graded 1 to 5. Final exam, NTNU TDT4127, Nov. 2018, task 10.
2. 212 submissions, graded 0 to 5. Final exam, NTNU TDT4127, Nov. 2019, task 4.
3. 212 submissions, graded 0 to 5. Final exam, NTNU TDT4127, Nov. 2019, task 5.
4. 210 submissions, graded 0 to 7. Final exam, NTNU TDT4127, Nov. 2019, task 10.
5. 201 submissions, graded 0 to 7. Final exam, NTNU TDT4127, Nov. 2019, task 13.
6. 204 submissions, graded 0 to 6. Final exam, NTNU TDT4127, Nov. 2019, task 18.

Notice that each source is only one task from one exam and not the whole. The grades are of differing values due to the tasks weight towards the total grade and actually represents percentages and not grades. Nevertheless, in this theses these points are referred to as "grades". There are also no evidence identifying the number of professional raters responsible for these grades, in such there could have been only one single rater or multiple.

2.2.2 Experiment 1

The first experiment was conducted by using the data generation method *questionnaires*. Questionnaires are a pre-defined set of questions participants respond to in a determined order. Questionnaires are often used together with the *Survey* strategy as questionnaires can easily be answered simultaneously or asynchronously by a multitude of participants. Questionnaires is often utilizing *selected responses* (multiple choice) and **not** *constructed responses* (text), which in turn makes it easy to quantitatively analyse the collected data. Questionnaires can be *self-administrated* or guided by a researcher in a *research-administrated* session. The current experiment was conducted as a *research-administrated* questionnaire (Oates, 2005, chap. 15).

This experiment only serve as a proof of concept/pilot as the goal of the experiment were to observe how well participants (in this case 10 students) enter the role as a *rater* and assigns a numerical grade (0-5, were 5 is top score) to 20 different Python source code samples. The empirical data collected from the participants in this experiment is therefore secondary to the functionality of the experiment, however it still might give some interesting insights. The source code samples used in this experiment were taken from *data source 1*, and were chosen because they are constructed responses and not too complicated.

2.2.3 Experiment 2

The second experiment was intended to be a repeat of Experiment 1 with 20 new source code submissions, a constructed "optimal" sequence and a large number of participants. The experiment were also intended to be conducted in March 2020 but was cancelled due to the outbreak of the COVID-19 virus. Restrictions to physical participation demanded the experiment to be conducted with a self-administrated questionnaire instead of the research-administrated questionnaire. 40 randomly selected students participated in a digital questionnaire which contained two parts (none who also participated in experiment 1). The first part introduced the participants to the experiment with necessary information and instructed them; in equal fashion as in Experiment 1, to grade the 20 source code submissions allocated in the second part of the questionnaire. The source code was selected from *data source 2* as the material was similar to the material used in Experiment 1, only with small differences. The used questionnaire can be found appendix B.

2.2.4 Experiment 3

The third experiment was different from the other two as it generated empirical data using the developed artefact. A series of smaller experiments were conducted to record the performance and outputs of the artefact. The datasources used in this experiment where datasource 2 to 6.

2.3 Data analysis

Data analysis based on the empirical data generated from the experiments was done quantitatively (i.e Quantitative analysis). Quantitative analysis is the technique of extracting understanding from numerical values using mathematics, statistics, tables, charts, and graphs (Oates, 2005, chap. 17).

The quantitative analysis was conducted with the tools *Microsoft Excel*, Python (Scipy.stats), and the statistical software program *Minitab*. As the empirical data analysed was mostly ordinal (0 to 5 and 1 to 20), the nonparametric methods Wilcoxon–Mann–Whitney two-sample rank-sum test (*U*) (McKnight and Najab, 2010), Levene's test (Derrick et al.) and Spearman's rank correlation coefficient were used. As multiple independent comparisons were conducted, a global p-value was calculated with Fisher's combined probability test (Li et al., 2014) and corrected with a Bonferroni correction. Parametric tests such as T-test and ANOVA was used on data of normal distribution.

Part II

Literature review and problem elaboration

Chapter 3

Literature Review

This chapter presents an overview of the literature researched prior to the writing of this thesis. It provides insight into the most relevant literature, even though it is only a subset of the total literature reviewed. Section 3.1 presents the two predominant approaches of measuring a student's knowledge, section 3.2 presents insight into the process of grading student submissions, section 3.3 presents an overview of different biases related to this thesis and section 3.4 presents an quick overview of digital exams in Norway.

3.1 Constructed and selected responses

Constructed and *selected* responses are the predominant approaches used to measure a student's knowledge—each providing their own set of practical aspects suited for different types of contexts. Constructed and selected responses are modifiable tools used in order to construct an arena in which the student can operate. They differ in their ability to let the student demonstrate their level of knowledge and critical thinking. However, choosing the correct tool to display the correct knowledge is critical as they do have limitations in certain areas. They are therefore often used in combination complementing to each other.

3.1.1 Selected response

A selected response is typically a **quantitative** approach which often provides a series of alternatives where usually only one answer is correct. It is not suited to provide an arena for the student to illustrate critical thinking and fully let the students demonstrate their complete knowledge of a subject. It is more likely used as an easy way of testing if the student knows the correct answers to a series of questions. A selected response is often chosen for its ability to be easily graded as it often provides a binary (True/False) aspect and is effortlessly translated into a numerical score (a machine often automates this process).

3.1.2 Constructed response

A constructed response is typically a **qualitative** approach. It demands the subject in question to demonstrate cognitive knowledge and reasoning. It is often called "open-ended" questions, as more than one answer are often accepted as correct. A constructed response provide an opportunity for the students to construct their answers within certain restrictions. Relative to a selected response, a constructed response provides a challenge when graded as more than one answer is accepted as correct, the degree of "correctness" is not binary, but multivariate. When the set of possibly correct and incorrect answers or the multivariate degree of correctness increases, the distinction of *correct* and *incorrect* becomes vague.

Even though a constructed response serve an important role in displaying a high level of knowledge and critical thinking, it also is hard to evaluate. Given a set of *rules* or *guidelines*, evaluation becomes a more manageable task. However, Rye (2014) provides claims of unequal grading of a constructed response due to differing factors in the grading community.

3.2 Grading

Grading is an essential process in academic environments in order to standardize the evaluation of the degree of individual achievement. Grades provide an objective truth, and the assignment of the level of achievement can be done with any ordinal scale. Academic institutions such as NTNU use the letters A through F, where A is top performance, and F is a failed attempt. The need to standardize the level of achievement in a field of study is to quickly convey the achieved level to any interested third party, such as an employer or another academic institution. The third part uses the grades (as well as many other factors) in essential decision-making processes such as to whom they should delegate positions to in a company, or as to whom universities will accept as students. The grade's role in the academic environment is therefore a very important part of the foundation of academia, and any form of involuntary risk which might damage its integrity is a problem for every partaker of the system.

3.2.1 Validity, fairness and reliability of grades

Preservation of academic standards such as validity, fairness and reliability is as stated in 1.1 and referring to Skedsmo and Huber (2018) and, The Norwegian Directorate for Education and Training (NDET) (6.S), of severe importance. The effect of invalidity or unfairness in the grading process can produce severe implications for the receiving parts. de Moira et al. (2002) states that "Marking should be at a common high standard and free from bias, otherwise some candidates are placed at an unfair advantage and others at an unfair disadvantage".

- **Fairness** refers to equal impartial treatment or behavior without favouritism or discrimination. A measure is considered fair if the result is not dependent on other factors than what is measured.

- **Validity** refers to the manner of evaluating accurately based on a set of sound policies. A measure is valid if it grants the correct level of degree of achievement.

- **Reliability** refers to the consistency of a measure. A measure is considered reliable if it produces equal results of the same conditions on multiple occasions.

These three aspects are independent of each other as grades can be reliable, fair, but invalid at the same time. As a valid grade is a grade which greatly reflects the level achievement, an invalid grade does not. An unfair and invalid system can be reliable.

3.2.2 Inter and intra-rater reliability

Inter-rater reliability refers to the level of agreement/disagreement between a group of raters. A high level of agreement is desired as it increases validity. We can measure inter-rater reliability by analyzing the standard deviation of grades given to one or more submissions from all parts of the group. As academia strives for equal measure for all, low inter-rater reliability is an issue in cases where two or more rater assigns different grades to identical submissions (Aslett, 2006).

Intra-rater reliability refers to the level of reliability of one rater. If a rater consistently assigns the same grade to identical assignments in different contexts, it is considered as reliable. However, that does not indicate that the grades are valid or even fair. Intra-rater reliability can consistently assign unfair grades. As mentioned in 1.1, mental and physical fatigue either due to monotony, lack of interest, or lack of sleep can have severe implications with regards to task performance and accuracy (Aslett, 2006). *Differential Rater Function over Time* (DRIFT) is a term introduced by Wolfe et al. (2001) to describe the different effects affecting a person undergoing repetitive and monotonic tasks. Mental attitude changes (often subconsciously) as we are influenced by fatigue and other factors, creating a shift in our evaluation and judgment. This shift might result in assigning more harsh grades to early positioned submissions in the grading process, and less harsh grades later in the process, or vice versa.

3.2.3 Subjective-influenced objective decisions

Since the degree of correctness is often not evident when dealing with a constructed response, subjective decisions are most likely prone to happen, even when following a set of rules or guidelines or even conducting a standardization-meeting prior to the exam (Raikes et al., 2009), (Midtbø et al., 2018). Objective decision-making refers to the elimination of subjective perspectives and a process that is purely based on hard facts. Eliminating subjective perspectives is not an easy task, especially when dealing with sub-optimal facts that do not cover the full specter of evaluation to assign the most valid grade. Subjective decisions fills the gaps of ambiguity and uncertainty in which the objective facts do not, creating an objective decision partly or heavily influenced by subjective factors such as mood and fatigue. The relationship between mood, fatigue, and other factors and how they influence memory and judgment has proven to be one of the most challenging problems in all of the social and cognitive psychology (Srull, 1984).

3.2.4 Grading computer science classes

Fitzgerald et al. (2013) reports insight into professional raters and their methods, grading scales, and agreement when grading computer programs. It concludes with the statement, "Clearly there is no single right way to grade programs.", but states that professional raters do agree with which programs are "Very good" and "Very bad". This agreement suggests that even if there is no consensus of the "correct" grade, professional raters are highly correlated with one another. Another research paper, Albluwi (2018) reports insight into different ways of grading a computer program, however it also report disagreement with what's "Very good" and "Very bad", disagreeing with Fitzgerald et al. (2013). Albluwi (2018) conducted an experiment where 60 computer science class instructors were presented with the task of grading four different source codes. The source code varied in quality by implementing different errors in all of them: *s1*(structural error), *s2*(syntax error), *s3*(logic error) and *s4*(both syntax and logic errors), respectively ordered from highest to lowest based on the average given grade.

s2 and *s3* were assigned different scores within the range [0 to 9] and [0 to 8] respectively where the total range possible was [0 to 10], suggesting a large disagreement between the participators.

3.3 Cognitive bias

In psychology, cognitive bias is "cases in which human cognition reliably produces representations that are systematically distorted compared to some aspect of objective reality" Haselton et al. (2015). In other words, our brain subconsciously tries to dictate our reason and behavior, which sometimes causes us to behave irrationally or experience errors in judgment. This phenomenon has been studied since 1971 when it was introduced by Tversky and Kahneman (1971) and further research has expanded our knowledge of how our brain works to a somewhat reliable model. There are many discussions and critiques in this field, but some predictable effects in specific areas have been documented and serve as a guide to understand how we are affected by internal and external factors.

In psychology, *heuristics* are mental "shortcuts" that allow people to quickly and efficiently solve problems and make judgments Shah and Oppenheimer (2008). They allow people to operate without the need to dedicate large amounts of time to evaluate their next move. In most cases, we do not need the optimal solution for every problem we face. “People rely on a limited number of heuristic principles which reduce the complex tasks of assessing probabilities and predicting values to simpler judgmental operations" Tversky and Kahneman (1974). Heuristics assist our ability to reach conclusions and make decisions by extracting sub-optimal solutions. If we had to find the optimal solution behind each decision, the amount of information to process and the processing-time required quickly extends towards infinity. Heuristics is documented to lead to cognitive bias, but all cases of cognitive bias is not an effect of heuristics Lockton (2012).

3.3.1 Sources of bias

There are many different kinds of cognitive biases. These cognitive biases are divided into groups based on their influences and effects, such as biases in a social context, decision-making, and biases affecting memory. A social bias is often viewed as a disproportional inclination for or against a person or a group, usually in a way that is interpreted as unfair or misguided.

The following tables 3.1 and 3.2 illustrate a small sample of different cognitive biases affecting not only academia, but various other relevant or interesting samples.

Name	**Description**
Serial position effect	Observing a sequence, recalling the first and last occurring items is easier than recalling centering items.
Distinction bias	The tendency to evaluate two items as similar/dissimilar when evaluating them simultaneously than when evaluating them separately
Assimilation effect	Two items perceived as similar/dissimilar will be further assessed as similar even when proven dissimilar/similar.
Anchoring	Excessive weighting/dependency of certain items/information when making decisions.
Consistency bias	Incorrect recall of past attitudes and behaviour altering them to resemble present attitudes and behaviour.
Contrast effect	Experiencing opposing instances of a certain stimulus creating a distinct perceived contrast between the instances.
Source confusion	Confusing memories (often episodic) with different memories/information, creating distorted memories.
Decision / respondent fatigue	After a long session of decision making, the quality of an individuals decisions deteriorates.

Table 3.1: This table illustrate a sample off biases in which take effect when observing items in a sequence, also called *Sequence effects*.

Name	Description
The Weber–Fechner law	Difficulty in comparing small differences in large quantities
Empathy gap	The tendency to underestimate the influence or strength of feelings, in either oneself or others.
Halo effect	The tendency to evaluate a person/item positively or negatively in a different context based upon previous observations.
Automation bias	The tendency to excessively depend on automated systems in which can lead to incorrect or poor decisions.
Bias blind spot	The tendency to view oneself as less biased/less prone to make mistakes than others. Lack of self-awareness.
Framing effect	The tendency to conclude with different results from the same information, only by changing the presentation.

Table 3.2: This table illustrate a sample off relevant cognitive biases. There exists an almost never-ending list of different biases observed and documented.

3.4 Digital Exams at NTNU

Since 2013, NTNU and other higher educational institutions in Norway initiated pilot testing and preparation for digital assessment of final exams (DIG), (Brusch et al.). In 2018, UNIT (The Norwegian Directorate for ICT and joint services in higher education and research) acted to preserve the task of digitizing higher education and research and further investing in the transition to digital platforms (UNIT, 2019). This enactment covers the whole educational system, including examination. To conduct a final exam digitally does not suit all arbitrary subjects, however, subjects assessed on a textual basis such as computer science courses are greatly encouraged to transfer to a digital platform.

Digital Assessment software (DAS) is a service that focuses on national collaboration, standardization of work processes, and coordination of the development of examination systems in the framework agreements for digital exams. In 2016, UNIT signet an agreement with three different providers of digital assessment software companies. The products of the agreement was the software *WISEflow*, *Inspera Assessment* and *Flexite!Exam* (Dig). Inspera Assessment is chosen as the main software for use at NTNU.

3.4.1 Grading with Inspera Assessment

Inspera Assessment provides a platform that is used before, during, and after the exam. The creation of the exam is managed by an interactive user interface in which an instructor can design the exam with different types of question types (constructed, selected, or both). During the exam, all students use their personal username and password to log in to a similar interactive platform that displays the questions. After the exam, a third interactive platform is used by the raters to grade the submitted answers.

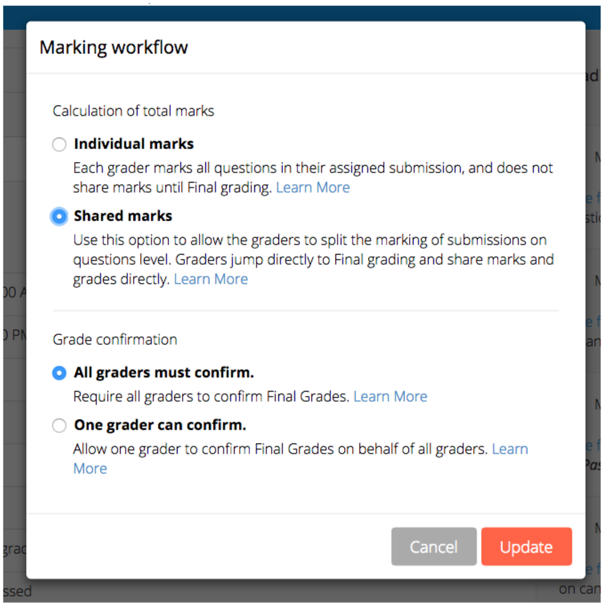
The submissions are presented to the rater in an ordered sequence based on the numerical studentID number. The rater does not know who the studenID number belongs to, making the whole process anonymous. The rater starts at the first submission and work their way thought to the last in a consecutive order. Limited functionality to navigate between the submissions are provided, but it does exist.

There is often more than one rater working together to grade all the submitted answers. Inspera provides two different methods of grading. One is the traditional method (*individual marks*), where each rater grade the complete submission, including all subtasks for one student and then move to the next. The other method is called *shared marks* and used when raters want to collaborate to create a single cumulative grade pr student. If the exam is divided into multiple independent questions, the raters can choose to divide the questions on the number of raters, spreading them out. Each exam is therefore not only graded by one single rater but multiple. In such, if the raters have low inter-rater reliability, it is spread out on all students equally, creating a more fair system. Figure 3.1 illustrate the option the original creator of the exam is presented with when selected the method of grading.

As the system is now, Inspera Assessment does not include a compiler as a resource students can utilize while conducting their exam (this is only relevant for computer science classes. A general compiler architecture is described in 5.3). This is explicitly not implemented since a compiler is a powerful tool one can use to validate if the written source code answer is correct or not. Most compilers also provide feedback on what is wrong or missing from the written source code if it does not compile correctly, providing the students with often unwanted or illegal information.

A compiler is a tool that makes the assessment of the student's knowledge harder and is therefore not implemented.

After the grading is complete, analysis can be done by exporting all the grades to a Microsoft Excel file (.xls) and analyze the grades to detect mistakes or to get general insight to increase validity.



Marking workflow

Calculation of total marks

○ **Individual marks**

Each grader marks all questions in their assigned submission, and does not share marks until Final grading. Learn More

● **Shared marks**

Use this option to allow the graders to split the marking of submissions on questions level. Graders jump directly to Final grading and share marks and grades directly. Learn More

Grade confirmation

● **All graders must confirm.**

Require all graders to confirm Final Grades. Learn More

○ **One grader can confirm.**

Allow one grader to confirm Final Grades on behalf of all graders. Learn More

Figure 3.1: Marking workflow in Inspera Assessment. Select between Individual marks and Shared marks (Sha).

Chapter 4

Problem Elaboration

This chapter expands on the defined problem introduced in the Introduction and the challenges faced when reviewing and developing the artefact.

4.1 Problem introduction

The topic of reordering submissions to improve fairness is not an easy task. The idea of reordering is simple, but how can we tell if one sequence is superior to another? When a human evaluates an object, subjective measures that are not easily quantified are used. Since the objective of this thesis is to *automatically* reorder identical or similar objects to follow each other directly, we need to measure the similarity between the objects as close to as a human would.

4.2 Generating a sequence

A sequence is described as a set of items and their position in the sequence. If all items where given the identifier based upon their position in the original sequence (the original sequence is the order the submissions were presented in from the data source, and is probably the *original* order a professional rater graded them in) such that the first item is identified by the number "1", the second number "2", etc. This creates a list of identifiers which represents the ordering of the items. A list of n items can therefore be presented as such: [1, 2, 3, 4, 5, ..., n].

The main goal of this thesis is to develop a method to reduce sequential effects when grading student submissions in computer science courses by automation. This is done by automatically generating an optimal sequence before it is graded. Therefore, all we want is just for the output of the algorithm to be a permutation of the original list (example output: [n, 5, 2, ..., 1, 4, 3]). The problem does not lie in generating a permutation, a random shuffled list is easy to make. The problem is to generate an optimal sequence. How do we do that? And are there such a thing as an optimal sequence at all? There are no prior research I have found that states exactly what the *optimal* order of submissions should be.

4.3 Evaluation of sequence

To find the optimal sequence, we need to prove it is optimal or at least present evidence that a sequence *might* be optimal. To do that we need a method of evaluating sequences, compare them against one another and extract the most optimal one. To evaluate something, you measure/observe its aspects or how well it performs in a certain area. But when we want to measure a sequence, which aspects do we choose to observe? How do we know if a sequence "performs well"? If we choose to measure the wrong aspects or area, we have no ground to find and extract the optimal one for our usecase.

4.4 Subjectivity of raters

Since all human raters are influenced by their subjective persona as described in 3.2, no rater are the same. Therefore, it exists an element of randomness in the given grades to constructed responses. As long as submissions of the type constructed response are rated by humans, the element of randomness is consistent. The empirical data from the experiments and other measures is therefore not only affected by the sequence, but also by this random element. So, even if a perfect optimal sequence were generated and presented to a rater before grading, the element of randomness will consistently have an effect on the given grades. This causes even further difficulty when evaluating sequences.

4.5 Calculating similarity

Let us presume that we found some aspects we want to measure (examples are: "the length of the code" or "how many times the word 'if' appears"). When comparing items, we need a method of calculating similarity. There exist a range of mathematical formulas and different algorithms, also called *metrics*, which inputs a set of measures, and outputs a numerical value describing the similarity between two objects. This numerical value is often in the range between 0 and 1. The challenge is to choose the metric who reflects the objective the most. Choosing the wrong metric might result in calculating incorrect distances.

4.6 Compiling of code

When grading computer science classes, source code is often submitted as answers. One great idea is to base the given grade upon how well the source code performs. The rater simply compiles and runs the program and bases a grade upon the results. As described in 3.4.1, the software system Inspera Assesment (and many other DAS) does not include a compiler. Students writing and submitting source code do not know if their code compiles without problems or not. They have no way of validating if the code is correct and as a result of submitting uncompiled code, it has a high chance of crashing if compiled. If the rater gives zero points to all programs that crashes, it raises the difficulty level by a great magnitude and harshly punishes those who submitted a near correct answer containing small mistakes. All features used to compare similarity between source code based upon compiled code is therefore hard or impossible to use. Generating tokens is still possible using alternative ways and is explained in 5.3.

Part III

Background theory, related work and system design

Chapter 5

Background Theory

This chapter presents the relevant theory used in the development of the artefact. Section 5.1 presents a detailed insight into clustering, section 5.2 presents an overview of how features are extracted and selected, section 5.3 presents an overview of a general compiler architecture, and section 5.4 presents a quick introduction to the Traveling Salesman Problem.

5.1 Cluster analysis

Cluster analysis is a technique used in many different subjects such as machine learning, data mining, statistical analysis, information retrieval, image analysis, and pattern recognition. It is a technique used to analyze current knowledge in order of extracting new knowledge. *Cluster analysis* is the problem of knowledge exploration by cluster assignment, not a specific algorithm in itself. Cluster analysis uses data to create groups of similar objects, also called clusters. A cluster in itself is simply a multitude of assigned data points where the *rules* of assignment can vary. The *rules* of cluster assignment is defined by a multitude of different algorithms with different objectives such as classification or grouping (clustering). Algorithms such as Hierarchical clustering, k-means and DBSCAN are famous cluster analysis algorithms (Pang-Ning et al., 2005).

5.1.1 Data points

A data point or object is a discrete representation of an observation or a measurement. It is described by **attributes**, which are properties or characteristics of an object. Data points are easily described by a table such as 5.1, where rows are data points and columns attributes.

ID	Name	Sex
1	John Smith	Male
2	Lisa Simpson	Female
3	Chris Thompson	Male

Table 5.1: Example of representing data by using a table

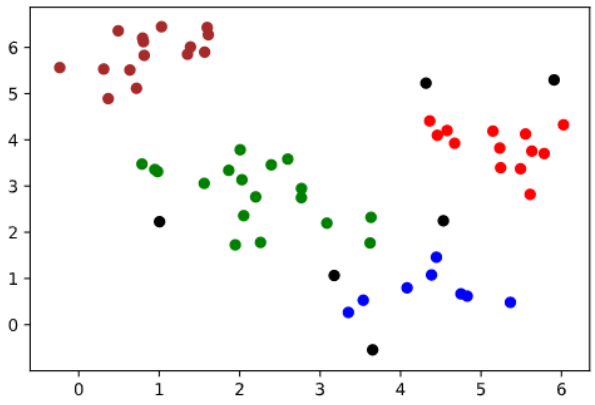


Figure 5.1: Example of clusters generated with the DBSCAN algorithm. The points are assigned to clusters, where the color represent their cluster assignment.

5.1.2 Noise

Noise, also called *outliers*, are distorted values often caused by errors in measurements or other forms of irregularities. Noise can be missing values, incorrect values, or drastically different values relative to the general norm in a set (i.e outlier). In Figure 5.1, noise are shown as black dots. When represented as a numerical number, noise is often written as the negative number "-1".

5.1.3 Data quality

The term "data quality" can best be defined as "fitness for use", (Tayi and Ballou). Data used in cluster analysis and other data mining techniques prefer data without noise, which adds irregularities and increases the difficulty of knowledge extraction. Reducing noise increases quality. A reduction of noise can be made using different methods, such as eliminating noisy data or estimation of missing values. Methods of preprocessing the data to make the data more suitable for its purpose can be applied. Methods such as *aggregation*, *sampling*, *dimensionality reduction* and *feature subset selection* is often used (Pang-Ning et al., 2005).

5.1.4 Features

Features in the sense of cluster analysis, are individual measurable characteristics of an object, which in total *describes* the object. A feature can be binary, categorical, or continuous and is synonymous with the term *attribute*. Many algorithms require features of numerical values since numerical values are more convenient for processing and statistical analysis. The importance of a feature is measured by its ability to convey useful information relative to a context. The nature of the context directly controls the usefulness of a feature. In the context of a job interview where the interviewee is describing why he/she should be hired, a *bad/ less important* feature might be the number of liters of blood the interviewee contains at the current moment. However, a *good/ more important* feature might be the interviewees' ability to communicate with other coworkers. In the context of a medical situation, a doctor might need to know the amount of blood contained in a patient and does not care much about the patient's coworker communication skills. A *feature vector* is an n-dimensional vector of features that represent an object. Extracting and selecting features is described in 5.2.

5.1.5 Distance Function

Having a method of computing similarity is necessary to differentiate the objects and assign them into clusters. Similarity, also called **distance**, is calculated in numerical values. The term *dimension* represents the number of features selected for a certain task, in this case, a calculation of difference/distance. A popular way of calculating the distance, d, between two objects in data analysis is Euclidian distance (Pang-Ning et al., 2005) given by the following formula:

$$d(p,q) = \sqrt{\sum_{i=1}^{n} \left(q_i - p_i\right)^2} \tag{5.1}$$

Where p & q are data points, n is dimensions and q_i and p_i are, respectively, the *ith* feature of p and q. A distance matrix represents the distances between all objects in a set. As an example, the distance matrix 5.2 represents the distances between the points $p_1 = (0,0)$, $p_2 = (2,0)$ and $p_3 = (3,2)$.

	p_1	p_2	p_3
p_1	0	2	3.6
p_2	2	0	2.23
p_3	3.6	2.23	0

Table 5.2: Example of a distance matrix containing the calculated distances between the three points $p1, p2, p3$

5.1.5.1 Levenshtein Distance

Levenshtein Distance, also called the *edit distance*, is a metric to calculate the difference between two strings by the number of changes (insertion, deletion, or substitution) needed to transform one into the other (Haldar and Mukhopadhyay, 2011). The higher the Levenshtein distance, the more different the strings are. Given two strings: (A) "hello" and (B) "world", the pseudocode for the Levenshtein algorithm is illustrated below. Table 5.3 illustrates how the algorithm calculates the distance by using a table.

```
Levenshtein_Distance(A,B)
    //Initiation
    distance = 0
    matrix = A.length*B.length matrix. First column and row range from 0 to (A.length or B.length)

    //Calculation
    for achar and ai in A  // achar = charater in a; ai = counter-object from 1 to A.length
        for bchar and bi in B
            if achar equals bchar
                matrix[ai][bi] = matrix[ai-1][bi-1]
            else
                matrix[ai][bi] = min(
                    matrix[ai-1][bi],
                    matrix[ai-1][bi-1],
                    matrix[ai][bi-1]) +1
    return matrix[-1][-1]
```

(1)

		h	e	l	l	o
	0	1	2	3	4	5
w	1					
o	2					
r	3					
l	4					
d	5					

(2)

		h	e	l	l	o
	0	1	2	3	4	5
w	1	1	2	3	4	5
o	2	2	2	3	4	4
r	3	3	3	3	4	5
l	4	4	4	3	3	4
d	5	5	5	4	4	4

Table 5.3: Initiation of Levenshtein Distance matrix is shown in table (1). Calculation of the Levenshtein Distance matrix is shown in table (2), where the bottom most right corner is the total distance.

5.1.6 DBSCAN

Density-based spatial clustering of applications with noise (DBSCAN) is a simple and effective density-based clustering algorithm. It was first proposed by Ester et al. (1996) and was awarded the SIGKDD Test of Time Award in 2014 (SIG), which is awarded to papers that have had an important impact on the data mining research community. With a runtime of O(logn) it discovers clusters of any amount and any arbitrary shape and size. As in the name, it is resistant to noise which is categorized by the algorithm as outliers and not assigned any cluster.

DBSCAN is a hard clustering algorithm which either assign a point to a cluster or assigns the point as an outlier. It uses a center-based approach to density, which allows us to classify a point either as a (1) core point,(2) border point, or (3) outlier. The concepts of core, border, and outlier are illustrated in Figure 5.2.

- **Core points**: A core point is the center of a cluster. There can be many core points in a cluster as they together form a tight mutually, joint linkage. A point is assigned the class *core* determined by a distance function, a distance/radius variable *EPS* (epsilon, ϵ) and a a specific minimum number of neighbouring points *MinPts* with a distance smaller than *EPS*.

- **Border points**: A border point is similar to a core point, but does not meet the requirements of the *MinPts*. A border point must have a core point as its neighbor.

- **Outlier points**: An outlier point is like a border point which does not meet the requirements of the *MinPts*. However, it does **not** have a core point as its neighbor.

DBSCAN works by iterating all the points in a set, identifying and assigning the points to a class. A separate group of core points identifies each cluster. The selection of the variables *EPS* and *MinPts* determines how the algorithm performs. DBSCAN is simple and robust but performs poorly within environments with various densities. *EPS* and *MinPts* are user-defined variables and often difficult to be assigned their optimal values as it can drastically differ amongst different densities.

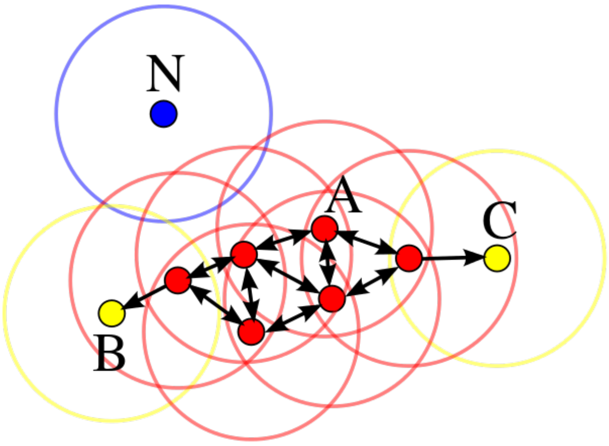


Figure 5.2: DBSCAN. Point A and the other red points are core points, B and C are border while the blue point is Noise/outlier. (Chire, 2011a)

5.1.7 OPTICS algorithm

As an improvement to DBSCAN, Ordering Points To Identify the Clustering Structure (OPTICS) was introduced in the paper Ankerst et al. (1999). OPTICS specifically targets the flaw DBSCAN possesses, its disability to not perform within environments with various densities at an optimal rate. OPTICS, as DBSCAN, require the variables *EPS* (ϵ) and *MinPts*, and uses the same classes as DBSCAN (core, border and outlier). However, the difference lies in the assignment of the classes. OPTICS assigns the class *core* to all points who has a defined *core-distance*. All other points which receive an undefined *core-distance* are either assigned as a border or outlier. The *core-distance* of an object is defined by the smallest ϵ possible while still maintaining *MinPts*. It is defined by the formula:

$$core-dist_{\epsilon,MinPts}(p) = \begin{cases} \text{UNDEFINED}, & |N_\epsilon(p)| < \text{MinPts} \\ \text{MinPts-distance}(p), & \text{otherwise} \end{cases} \tag{5.2}$$

Where p and q are objects, $N_\epsilon(p)$ is the set of objects inside p_ϵ radius (neighbours) and *MinPts-distance(p)* is the distance from p to its MinPts' neighbor. All *core points* also have a *reachability-distance*. All points contained inside this reachability-distance who is not already assigned as *core*, is assigned as a *border*. The *reachability-distance* can not be smaller than the *core-distance*. The *reachability-distance* (i.e *reach-dist*) of an object is defined by the formula:

$$\text{reach-dist}_{\epsilon,MinPts}(p,q) = \begin{cases} \text{UNDEFINED}, & |N_\epsilon(p)| < \text{MinPts} \\ \text{max(core-dist(p), dist(p,q))}, & \text{otherwise} \end{cases} \tag{5.3}$$

Where *dist(p,q)* is the distance function between two objects and can not be larger than ϵ. All core and border points assigned as neighbors defines a cluster. Points not reachable by a core point's *reachability-distance*, is an outlier and not assigned any cluster. Illustrating the results after running the OPTICS algorithm on a set of objects can be done with a *reachability plot*. It displays the *reachability-distance* for each point and illustrates the different clusters as "valleys" and noise and border points as "mountains". Figure 5.3 illustrates a *reachability plot*.

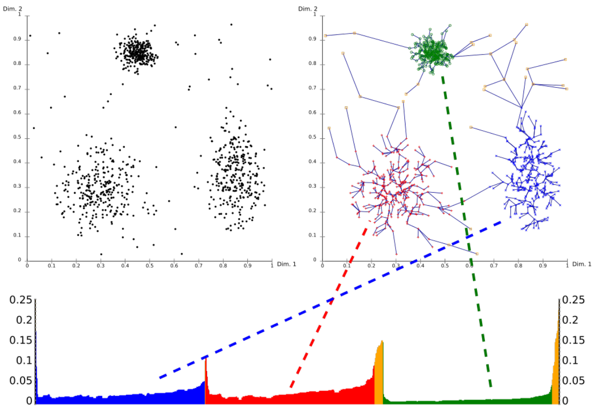


Figure 5.3: This figure illustrate a reachability plot generated after the OPTICS has been executed and calculated all the reachability-distances between the points. (Chire, 2011b)

5.2 Feature extraction and selection

In machine learning and statistics, extracting and selecting the correct features serve a critical role in achieving the optimal results. As discussed in 5.1.4, features carry information, and its importance is relative to the context. When a context changes, so do the required features to capture the essential information. Extracting available features and selecting the most important features is a challenge.

5.2.1 Feature extraction

Feature extraction is the process of extracting usable features from raw data into a *feature vector*. Raw data is data without organization or structure, often collected from a source such as text, measurements, images, and sound. Due to the nature of disorder and variations in raw data, choosing the correct feature extraction method is difficult, though it is the key to success. Some sources are less variable (it has fewer dimensions) and more structure than others, thereby allowing features to be more easily extracted while other sources are the opposite. The amount of dimensions a domain contains defines the method of extraction. Using methods of extraction designed for domains with a low amount of dimension ($D < 10$) on a domain with a high amount of dimensions ($D > 50$) yields poor results (Lewis, 1992). It is therefore important to recognize the operational domain, its dimensions and thereby choose the appropriate extraction method.

Feature extraction from textual domains is the feature extraction category *Structural description*, containing areas such as formal languages and their grammars, parsing techniques, and string matching techniques (Lewis, 1992). In this category, it exists two different kinds of features, *structural* and *statistical* features. Various methods are used to extract the structural and statistical features. *Structural methods* identify the structural features of a character where structure refers to how the information within a written text is organized. *Statistical methods* identify the statistical features of a character were statistical refers to "features of a data set, which can be defined and calculated through statistical analysis" (Wha), (Vithlani et al., 2015). Typical methods of feature extraction of textual sources is "Bag of words", "TF-IDF", and "Word2Vec".

Source code lies in the textual domain, which allows all textual feature extraction methods to generate a feature vector from the source code. Source code is heavily dependent on **keywords** and **syntax**. Therefore both recognizing the structure and statistics of characters is of great value. Examples of differnet methods of structural feature extraction of source code are the following:

Abstract Syntax Tree (AST): AST is an abstract syntactic model of the source code represented by a tree structure. The source code is parsed and modeled into an AST where nodes are program elements, and paths are the relative movement between the nodes (Alon et al., 2018). AST is a widely used model to compute the similarity between source code by comparing the structures of the program. Comparing structures allows for ambiguity in keywords and reduction of noise. However, comparing the structural tree graph with N-nodes is complex in computation and a worst case of O(n^3) (Ragkhitwetsagul et al., 2018). Solving a comparison problem between graph-based models are often NP-complete, however simplifications and reduction in complexity can be made to lower the computation time. AST also lacks the ability to be generated if obstructions or unintended faults in the syntax occurs. The source code needs to be correctly implemented before the AST can be created. Other graph methods are program dependence graphs (PDG), Compile tree, or control flow graphs (CFG) (Ragkhitwetsagul et al., 2018).

```
while (!d) {
  if (someCondition()) {
    d = true;
  }
}
```

(a) A simple JavaScript program.

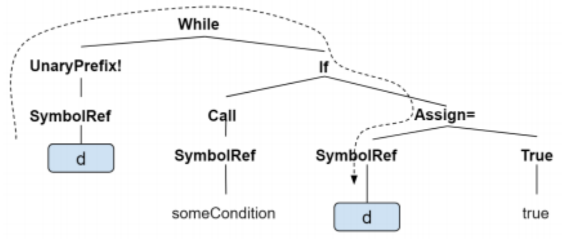


Figure 5.4: A JavaScript program and its AST, along with an example of one of the paths. (Alon et al., 2018)

N-gram analysis: N-gram analysis is a predictive model often used in Natural Language Processing (NLP) and used to detect and predict structures in sequences. It creates overlapping subsets of length n containing words from the text. Based on the N-1 previous words, it computes the probability of the next word in a sequence.

$$p(s) = p(a_1)p(a_2|a_1)p(a_3|a_1a_2)...p(a_n|a_1...a_{n-1}) \tag{5.4}$$

Where p is the probability function and *a1,a2...an* is objects in a sequence of length n (Hindle et al.). A similar usage of n-grams is the *syntactic n-grams*. It uses a tree-based structure and generates syntactic relations between words which reduces arbitrariness (Sidorov et al., 2014).

latent semantic indexing Latent semantic analysis (LSA) is a technique to aggregate all the words in a corpus by analyzing the context of relative appearance. This is done with the intent to provide sets of mutual words with similar context appearance, and therefore similar meaning (Landauer et al., 1998).

5.2.2 Feature selection

Feature selection is the process of selecting the relevant and discard irrelevant features to improve performance (Brank et al., 2011). Consider a set of independent features (f) extracted from a domain as $F = \{f_1, f_2, ..., f_n\}$. Using these extracted features, we wish to select a subset of relevant features and remove irrelevant or redundant features. The reason to only select a subset of features instead of using them all has to do with the phenomenon called "the curse of dimensionality". When the number of features increases, the time required and the complexity for an algorithm to compute increases, sometimes exponentially. It is also worth noting that additional features do not always increase the total descriptive power (Trier et al., 1996). Often, there exists a finite small subset of features with equal (or near equal) descriptive power as the whole. The task of selecting the optimal subset of relevant features is hard, as the term "relevant" is often somewhat obscure.

5.3 Compiling

A compiler is a translator. It is computer software who translates code written in a programming language such as Python or JavaScript (source), into another language (target). Compilers are necessary to translate human-written source code into generic machine code.

Now, why do we do this? In short, humans and computers like to organize things differently. Humans can effectively understand high level languages (such as Norwegian, English, or Python) containing aspects such as ambiguity and context. Humans can extract information from poorly structured data such as an image or a written Shakespeare poem without much trouble. Computers on the other hand, lack this ability. For a computer to know what to do, it needs to know exactly what needs to be done in the correct order, or else it will crash. It is very sensitive to miss-inputs and structural changes. Therefore, source code is written to be easily understood and correctly built by humans, then compiled into a language easily understood by the computer (i.e machine code). This is what the compiler does. It takes what is called "high level" human-written code and transforms it to an understandable "low level" code for a computer.

As there are many different languages, there are many different compilers. One can not use a Swedish-to-English translation book to translate Chinese into English. It is the same with computer programs. One can not use the same compiler designed to compile Python into machine code to compile JavaScript into machine code (unless it's designed to do that specifically, but this is never the case). Therefore there are as many (or more) compilers as there are computer languages, and none of them are exactly equal. However, they all share some common traits.

The common compiler is made up of many steps of code transformation. **Lexical Analyzer**, **Syntax Analyser**, **Semantic Analyzer**, Intermediate Code Generator, Machine Independent Code Optimizer, Code Generator and Machine Dependent Code Optimizer. In this thesis's scope, it is only necessary to understand the lexical, syntax, and semantic analysis processes.

5.3.1 Lexical Analysis

Lexical analysis is the first phase of a compiler. It takes source code as input and outputs the code as a series of tokens. Lexical analysis is generally made up by two steps. *Preprocessing* and *Tokenization*.

5.3.1.1 Preprocessing

Preprocessing is the first process of lexical analysis. It "cleans" the raw input to make it more easily read by the following processes. It removes white-space and following a set of *rules*, it splits the raw code-words into individual characters called *lexemes* that are more easily interpreted by the Tokenizer. A **lexeme** is a character or a combination of characters that serve as individual element. A *lexeme* is often simply a word such as "if", "while", or "return", but can also be numbers or separators such as ":" or ";". The identification of lexemes is usually made with a Final State Machine (FSM) (Figure 5.5 is an example of an FSM) in a process called **Scanning**. The Scanning process iterates the input character stream and identifies characteristics such as numerical, alphanumerical, or a combination of symbols (!==, <=). The Scanner maps the characters towards a corpus of words to identify *legal* and *illegal* lexemes. The process of creating lexemes can sometimes be very complicated, especially if the total set of allowed lexemes is very large. In cases where the Scanner identifies an *illegal* lexeme as a result of a input-error or illegal syntax, the Scanner stops the whole process and returns a **compilation error**. Figure 5.6 illustrates the inputs and outputs of the Preprocessor. It inputs source code and outputs a list of tokens.

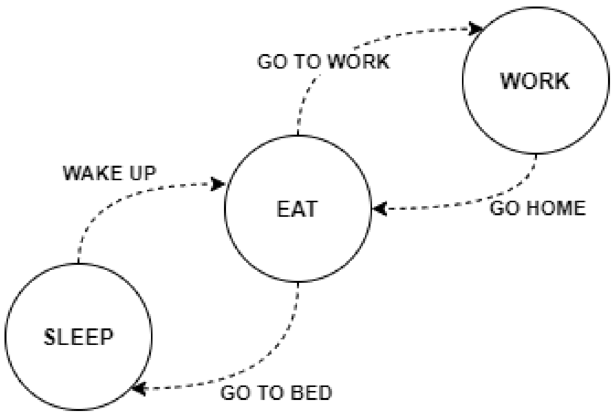


Figure 5.5: Example of a Final State Machine where any node is the starting node or the end node.

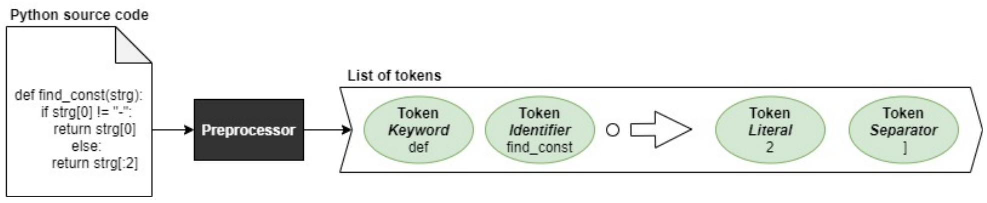


Figure 5.6: Illustration of inputs and outputs of the Preprocessor. Inputs raw text in the form of source code and outputs a list of generated tokens.

5.3.1.2 Tokenization

The tokenizing process transforms the generated lexemes into tokens. Tokens are wrappers with additional information of a fixed uniform structure to a lexeme. It tells the system what kind of meaning the lexeme has as it adds semantics to the lexeme. The tokenizer transforms lexemes into tokens by mapping the lexeme to a set of predefined grammatical rules. The tokenizer uses an evaluation function to map the lexemes into different categories such as *keywords, constants, identifiers, literals, operators* or *separators*. Table 5.4 shows an example of different types each lexeme is mapped to.

Type	**Value**
keyword	for, while, if, else, elif
separator	(,), [,], {, }, ;, :
literal	True, False, numbers, lists, tuples, dictionaries, sets
operator	+, -, *, /, and, or

Table 5.4: Example of tokens and their values.

All lexemes are assigned a category and converting into tokens. The *token* is either illustrated by a uppercase words such as EQUALS or SEMICOLON, or it is illustrated by a token wrapper "Token(*category*, *value*)" such as Token(operator, ’=’) or Token(seperator, ’:’).

5.4 Travelling salesman problem

Travelling salesman problem (TSP) is an optimizations problem: "given a graph of nodes, construct the shortest path such that all nodes are visited once and only once". A path visiting all nodes exactly once in a graph is called a Hamilton path. TSP tries to find the shortest Hamilton Path. TSP is NP-hard (can not be solved in polynomial time), and exhaustive search requires $O(n!)$ complexity which leads to the need to calculate $2 * 10^{18}$ different calculations even only for a graph with 20 points. Reducing computational complexity can be done by accepting sub-optimal solutions with various heuristics such as *greedy*, *insertions*, *Christofides* or *Nearest Neighbour heuristics* (Nilsson, 2003). The norm is to use a heuristic as the time to optimally solve TSP takes too long when working with larger problem spaces.

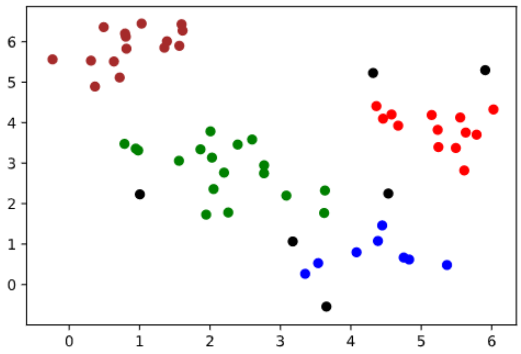


Figure 5.7: Example graph with vertices

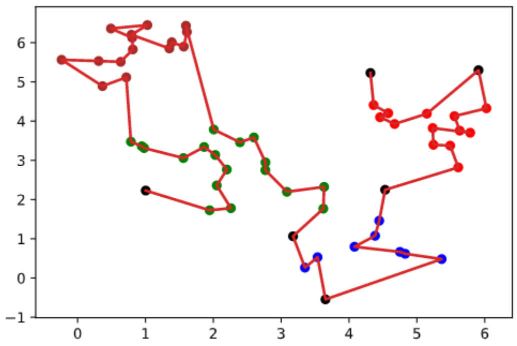


Figure 5.8: Example Hamilton path made with TSP

Chapter 6

Related Work

This chapter presents a closer look at the related work carried out in the field of automation and grading. Section 6.1 presents an overview of systems who automatically grades student submissions in various ways, section 6.2 presents an overview of how clustering is utilized by different systems and section 6.3 presents an overview of other works who have conducted experiments with sequences.

6.1 Autograding

Given biases and other unreliable elements introduced into the academia by humans, one might be induced to implement a more consistent, reliable, and automated method. Such a method can produce the same reliable output unaffected by the influence of biases or fatigue. The task of automation is simple in the aspect of calculating a grade on selected responses (multiple choice and similar task styles). However, the difficulty increases given the task of automatically grading subjects with constructed responses such as essays or source code (Srikant and Aggarwal). Autograders have proved to have a high correlation with human grades, but lack insight into the actual content and other higher-order aspects (Attali et al., 2013). Therefore, Autograding has proven to be insufficient as a single source of measure when dealing with submissions of importance, such as final exams. Several studies have conducted investigations in a human-machine collaboration where the autograder serves as a complement, providing access to its suggested grade and considerations (Attali et al., 2013). This human-machine collaboration has increased human reliability and consistency, however it also increases human and machine grade correlation. This correlation infers that human raters are influenced by the autograders low-level thinking, which reduces the importance of higher-order content in the grading process (Attali et al., 2013).

Using Latent Semantic Analysis (LSA) to automate grading of computer programming assignments were studied by Zen et al. (2011). The main idea was to analyze the source code using LSA, which uses text-words; hereby referred to as terms, and compares their similarity against predefined correct answers. LSA is often used in the field of Natural Language Processing (NLP) where the task is to make a machine understand human written text. LSA assumes that terms which appear in a similar position, have a similar meaning. LSA uses a term-document matrix to represent the occurrence of terms in a set of documents. Since Zen et al. (2011) automates grading on *source code*, which consists of alphanumerical, mathematical, and other specific syntax symbols and not regular text, noise is a problem. Noise in data is described in [section x] "meaningless information" and should either be removed or reduced so as not to interfere or create unreliability. Zen et al. (2011) asserts variable names and other alphanumerical characters as noise and reduces noise by complete removal, leaving only separators and operators ("if(n>1)" and "if((n==0)||(n==1))" is transformed into "(>)" and " ((==)||(==))"). *Cosine similarity* (Cos) is a metric used to calculate the similarity between the predefined correct answers. The similarity was finally used to calculate a grade. The comparison of this technique as to human grading were proven to be insufficient. LSA is able to grade assignments fast and consistently in addition to avoiding bias, however, LSA does not differentiate the order of symbols and therefore completely ignores the importance of syntax. Related research using LSA on source-code is the paper Alves dos Santos and Favero (2015).

Srikant and Aggarwal (2014) implemented a highly sophisticated machine learning system in order of autograding written source code. Advanced methods were used to extract features describing the complexity and the correctness of a program. As discussed previously in section 5.2, this is a task that should not be taken lightly as the difficulty of feature extraction and selection is very complicated. The machine learning algorithm (Srikant and Aggarwal, 2014) implements utilizes regression with 3-fold cross-validation to model the grades and linear ridge-regression, SVM-regression, and Random Forests to build the models. This approach is by far one of the best ways to acquire insight into higher-order concepts and ideas, but it has a major drawback as it uses pre-built correct solutions in its grade-calculation.

What makes automatic grading insufficient on important submissions such as final exams is their inability to truly grasp the high-order concepts. They often do not meet the requirements to be used in exams and often add additional workload. The lack of a compiler at Norwegian universities makes the usage of AST nearly impossible as AST needs source code without syntax errors. The need for additional workload to create an environment for the autograders to fully function is not efficient. This is workload such as generating different correct answers patterns that use the same techniques one predicts the students to use.

6.1.1 Combining constructed and selected responses

Since selected responses provide a quantitative approach and constructed responses provides a qualitative approach, why not combine the two? Take the best of both worlds and create an approach easily quantified and able to demonstrate a high level of knowledge? One such approach is Parson problems and researched in **??**. Parson problems is a tool suited explicitly for computer science classes and source code programming problems. It is a set of "blocks" written of source code the partaker needs to place in the correct order. By utilizing indentations as a variable additional to the blocks, the possible permutations quickly increase by the polynomial factor of *n!*. Therefore, even if the student does not write the textual content and is presented with a list of building-blocks, the student is able to demonstrate knowledge of a higher order. The blocks can more easily be automatically graded than text.

6.2 Clustering

Massive Open Online Courses (MOOCs), with several thousand attending students from all around the world, is rapidly becoming a popular alternative in education (Hatzipanagos and Gregson, 2015) (Basu et al., 2013). MOOCs are highly scalable as the lectures are automated with videos, texts, and social forums. Assessment is completed with weekly quizzes and multiple-choice questions. However, Hatzipanagos and Gregson (2015) states that MOOCs remain limited in its ability to assess complex and open-ended student assignments. Although the dynamic scalability of selected responses is perfect with MOOCs, constructed responses are not. As constructed responses are the superior way of demonstrating knowledge (Anderson and Biddle, 1975), they are also desired in MOOCs. As well as assigning numerical grades, the challenge is grading and giving quality feedback to the constructed responses without increasing the workload. Basu et al. (2013) coined the term "powergrading" as the solution to this problem. When a student delivers their answers to the same task, different groups of students have often made the same mistakes. Writing feedback to each group/subgroup instead of each individual reduces workload and increases consistency. Basu et al. (2013) uses Latent Semantic Analysis (LSA) to extract and select features, tf-idf as weighting and cosine similarity as distance function. Clustering into groups were done with the k-medoids algorithm (Kufman and Rousseeuw, 1987). Overall accuracy were computed to be 92.9%

Clustering based on source code written in programming courses to decrease workload was researched by (Barbosa et al., 2018). The research describes an attempt to decrease the time between submission and receiving feedback, minimize the evaluation effort as well as considering the different *criteria* of each evaluator. "Adaptive clustering" is the concept of not only clustering, but adapting the clusters to match the aspects of the evaluator using it. The clustering was done with Kmeans, and similarity was calculated with Jaccard similarity.

6.3 Sequence manipulation

Attali et al. (2013) experimented with the effects of sequence when graded. The purpose was to study if the sequence affected inter- and intra-rater reliability of grading and/or the effort to discern differences in essay quality. In the experiment, a shallow clustering algorithm were used to group similar essays on specific criteria. The **length of the essay** were chosen as the similarity feature and the clustering algorithm were simply to group essays of similar length together. Raters were asked to grade all essays from one group before moving to another group. The results of the experiment stated that the sequence had no effect on scores, reliability and did not seem to have any effect on the rating process at all. The participators gave feedback stating:

1. ‘scoring the groups of long responses was quite exhausting’,
2. ‘It can be very refreshing to work with and within the entire score range, not only for variety, but for clarity’,
3. ‘sequence might unduly influence a rater’s thinking to even subconsciously tend towards “longer must be better”’

Chapter 7

System Design

This chapter presents the developed artefact with the main goal of automating the sorting of any arbitrary sequence into an optimal sequence. Section 7.1 presents insight into what an *optimal sequence* might be, section 7.2 presents a general overview of the artefact and its parts, section 7.3 presents a detailed insight into the architecture of the *Preprocessor*, section 7.4 presents insight into how the artefact calculates distances, section 7.5 presents a detailed insight into how the submissions are clustered, and section 7.6 presents insight into how the sequence is finally generated.

7.1 Optimal sequence

The *optimal sequence* should be a sequence that increases reliability, validity, and fairness. Based upon the research done by Spear (1997) and other related works such as (Zhao et al., 2017), (Aslett, 2006), (Wolfe et al., 2001) and (Fitzgerald et al., 2013), this might be achievable by sorting the complete set of submission in such a way that equal or similar submissions are graded directly after one another. As similar items might get similar grades, the contrasting quality between items should be reduced. Such a sequence might reduce contrasting effects and give equal grades to equivalent submissions.

7.2 General Design

The general goal of the artefact is automatically sort any sequence of Python source code into an optimal sequence. The artefact is made up of separate *modules*. These modules contain *submodules* who are the building-blocks of the artefact. The submodules is mostly made up of already stress-tested algorithm with previously proven results. The modules and submodules is illustrated in Figure 7.1.

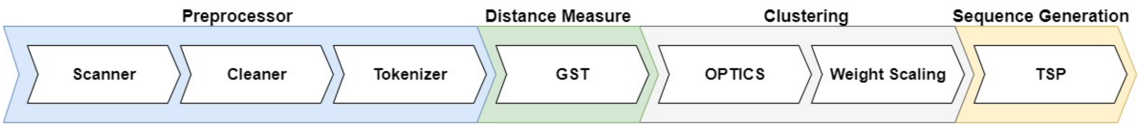


Figure 7.1: This model describes the general design of the artefact, its *modules* (color) and their respective *submodules* (white).

7.2.1 Design introduction

What I designed and implemented in this thesis was an algorithm/artefact that requires Python source code as input, and outputs a list of integers representing the submissions' position in a sequence. As this design does not depend on any already implemented digital examination tool, it can be used as a third party to any arbitrary system. The artefact is four separate modules: *Preprocessor*, *Distance measure*, *Clustering* and *Sequence generation.*

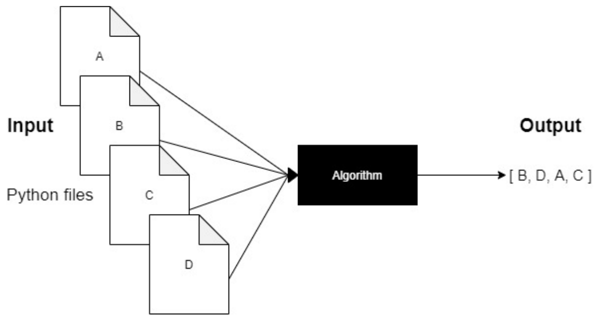


Figure 7.2: This model describes the very basic idea of the system. In this example, the Python files A-B represent student submissions. They are presented to the artefact, which outputs a list of identifiers describing their optimal position in a sequence. The output can be any arbitrary permutation of the input files.

As the problem of this thesis is an open-ended problem, there is no "correct" design, only suggestions. This design only describes a suggestion of how to solve the main problem. The rationale behind the implemented design is to keep it as simple as possible and to only use already stress-tested modules.

The general design is inspired by source code similarity systems for plagiarism detection (Durić and Gašević, 2013). The intention was to acquire some sort of similarity measure between each submitted source code and generate an arrangement based on that. The general design is made up of four main modules. The preprocessor, distance measure, clustering, and sequence generation.

7.3 Preprocessor

The preprocessor module has the critical task of streamlining the raw input of Python files, such that the following modules can operate without failure. Its main task is to iterate over the files, extract the content, and present it to the next module (distance measure) as a list of tokens. The Preprocessor is made up of three submodules, the Scanner, Cleaner, and tokenization. The Preprocessor design is a replication of the first stages of a compiler (lexical analysis) with a few modifications. A regular lexical analysis breaks if there are syntax errors, this Preprocessor does not.

7.3.1 Scanner

The Scanner simply reads the files line by line and assembles the lines together in a list. The inputs and outputs of the scanner is illustrated in Figure 7.3.

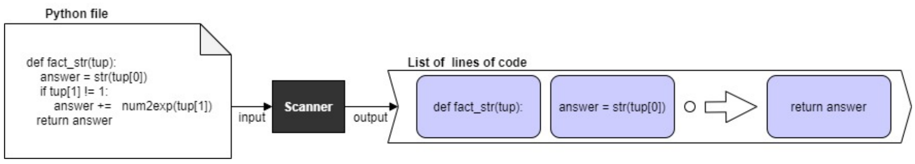


Figure 7.3: This model describes the scanning process. A & B are input files and the output is a list containing the content of A & B.

7.3.2 Cleaner

The Cleaner produces *lexemes* from the Scanner output. The lexemes need to be without noise, such as spaces, comments, and other insignificant symbols. The Cleaner removes these insignificant symbols and outputs a list of lexemes. The inputs and outputs of the scanner is illustrated in Figure 7.4.

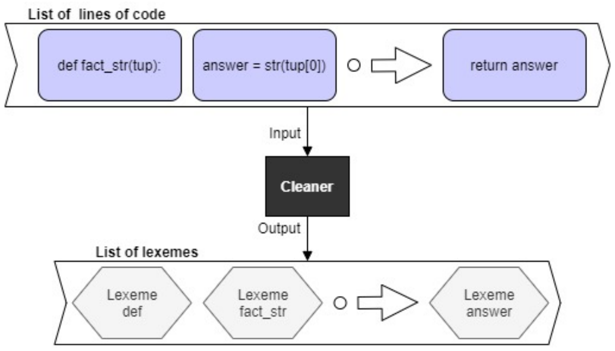


Figure 7.4: This model describes the cleaning process of the Cleaner. Output from the Scanner is takes as input and lexemes are given as output

7.3.3 Tokenization

The Tokenizer is a Final State Machine (FSM) which inputs lexemes generated from the Cleaner and outputs a list of tokens generated from the lexemes. While the lexeme, in its simplest form, is just a word, a token is a wrapper of information to the lexeme, adding semantics.

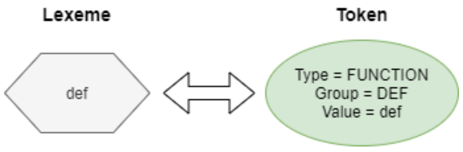


Figure 7.5: This model describes the relationship between a lexeme and a token. In this example, the lexeme is created from the word "def".

A token is described by the hierarchy (also called dimensions) *Type* => *Group* => *Value* (as illustrated in Figure 7.5). *Type* describes what kind of token it is (Keyword, Function, Separator, etc), *Group* describes what functionality the specific token has (Function, Flow, Separator, etc), and *Value* simply contains the lexeme the token is built from. The table in Appendix A describes all accepted Keywords, groups, and values. Together, the variables provide the original written text word with semantics we can use when computing similarity. Table 7.1 illustrates a small sample of the hierarchy.

Type	**Group**	**Value**
Keywords	CONTROL_FLOW	for, while, if, else, return
	MODULE	from, with, as, import
	EXCEPTION	try, except, finally, raise
Separator	BRACKET_START	(
	COLON	:
	COMMA	,
Operator	ARITHMETIC	+, -, *, **, /, //, %
	RELATIONAL	<, >, ==, ===, !=, <=, >=
	LOGICAL	and, or, not, !
Literal	BOOLEAN	true, false
	NONE	none, null
Variable	NUMBER	numbers
	STRING	"text written within quotation marks"
	VARIABLE	assigned to user defined variable names
Function	FUNCTION_DEFINE	def
	FUNCTION	variables defined as callable functions
Unknown	UNKNOWN	lexemes not assigned any other type and group

Table 7.1: Small sample of the token hierarchy illustrating the levels *Type*, *Group* and *Value*. The full lookup table can be found in appendix A

7.3.3.1 Syntax Analyzer

Handling lexemes that do not fit any of the regular syntax types is an issue. In a regular compiler, if something is not recognized immediately, the compiler stops and returns an error. We do not want our preprocessor to stop if it must handle something unidentifiable. The Preprocessor is designed to solve this by introducing a unique *Unknown* token illustrated in Figure 7.6. The Unknown token lets the Preprocessor continue to create tokens even though the syntax is incorrect. This is the major difference between the designed Preprocessor and a regular compiler.

The drawback of using the Unknown token is that it adds noise. It does not add any semantics except for the fact that it is unknown and therefore not easily identified or used. Thus it should be reduced as much as possible. The lexeme an Unknown token is built upon are often variable identifiers. They are strings of arbitrary length and values of any alphanumerical characters. As described in 6.1, such variables pose a problem as they make noise. We therefore want to process the Unknown tokens through a *syntax analyzer* to discover if they can be changed into one of the other tokens.

The syntax analyser evaluates the Unknown tokens and recognizes tokens with equal lexeme values. If it have previously generated a token with an equal lexeme value, the Unknown token is transformed into a copy of the found token. This process is illustrated bu Figure 7.7.

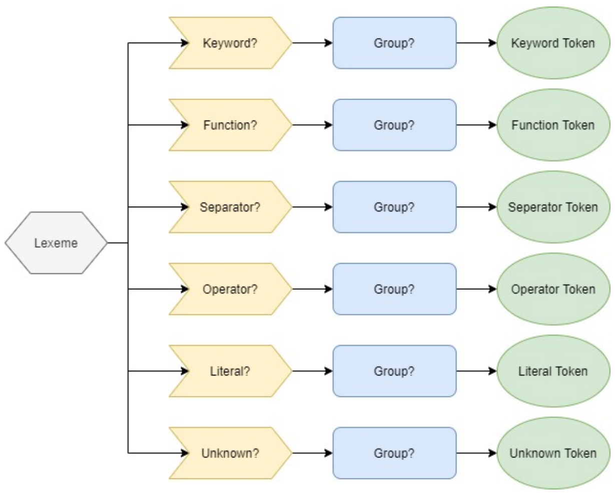


Figure 7.6: This model describes the tokenization process. It inputs lexemes and outputs Tokens.

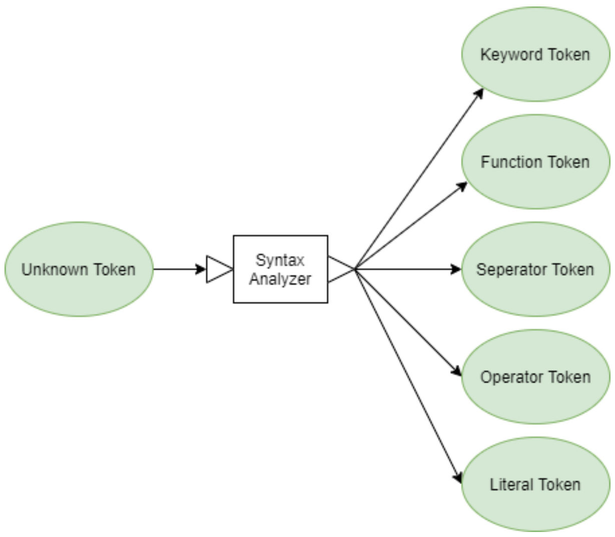


Figure 7.7: This model illustrates the Syntax analyser as it inputs a *Unknown* token and tries to map it into one of the other token types.

7.4 Distance measure

The Distance measure module is tasked with calculating the distance between the all the input files. The files is represented as lists of tokens, also called token-strings from the Tokenizer output.

7.4.1 Greedy String Tiling

Greedy String Tiling (GST) is "for comparing pairs of strings to determine the degree to which they are similar" (Wise, 1993). The metric inputs two token-strings (TS1 and TS2) and calculates a numerical distance between the two. It was first proposed by Wise (1993) in 1993 and used in source code similarity systems like JPlag (Prechelt et al.), and suggested as one of the best source code similarity algorithms when comparing plagiarism detection (Durić and Gašević, 2013).

The metric is based on a one-to-one matching and is similar to the "longest common substring" (LCS) algorithm (Babenko and Starikovskaya, 2011). The objective is to find as many matching elements in a row and to mark them such that they are not counted twice. The rows of matched elements are called a "block". LCS and GST try to find as many blocks as possible and then calculate a numerical value from the blocks.

The difference between LCS and GST is that the LCS algorithm is "order-preserving", which means that it does not ignore different permutations of substrings. That is a problem since source code can be written in multiple ways and still have the exact same functionality. GST ignores permutations of substrings and will calculate the same distance for all arbitrary permutations of two strings. This functionality has some costs and makes the runtime complexity of GST to be$O(n^3)$ while LCS (depending on how you implement it) generally has $O(n*k)$ complexity. GST can be improved to almost $O(n)$ using Karp-Rabin pattern matching algorithm (Richard Karp Michael).

GST iterates the source code token by token and outputs a list of *tiles*. Let set A be the set of all tokens in TS1 and set B be the set of all tokens in TS2. A *tile* describes a set of *matches* in succession. A match is any element in the in the intersection $A \cap B$. A tile is written of the form "(x,y,z)" where x is the position of the first element in the succession, y is the position of the the last element and z is the number of elements in the tile. The pseudo code for GST is described in the appendix.

After the execution of GST, the numerical distance between the token-strings can be calculated by following the distance function formulas (7.1 and 7.2) originally described by Prechelt et al.:

$$\text{sim}(A,B) = 2 * \text{coverage}(\text{tiles}) / (|A| + |B|) \tag{7.1}$$

$$\text{coverage}(\text{tiles}) = \sum_{\text{match}(a,b,length)\in\text{tiles}} \text{length} \tag{7.2}$$

A token can be compared on the three different dimensions (Type, Group and Value), each providing their own level of "**strictness**" as they carry different semantics. As an example, if given the task of comparing colors, choosing a level of strictness is important. With a high level of strictness, the colors blue and light blue are not equal. Reducing the level strictness increases the similarity of blue and light blue. With the right amount of strictness, blue and light blue is equal to each other, while blue and red are note equal. Comparing two colors without any strictness at all, all colors are one and the same.

When comparing two tokens, the *dimensions* describe the level of *strictness*. Comparing two tokens on their *Value* (i.e lexeme) is very strict as often variables such as variable identifier can vary a lot. This creates few to no matches at all. Comparing two tokens on their *Type*, creates only as many different tokens as there are different types, making two files too similar. Choosing the level of strictness gives different results when comparing source code which leads to a totally different performance of the whole algorithm. The strictness variable is therefore very important to the whole artefact. Since source code is ambiguous, we want to use the correct level of strictness to achieve the best performance. In this thesis I chose to use strictness level of *Group*, as it provides the highest amount of semantics and an average strictness. Example 1 and Example 2 shows the difference between using Group or Value as dimension for comparison.

Example 1.

```
A = ['def', 'func', '(', 'arg',  ')', ':']
B = ['func', '(', 'param', ')']

GST(A,B) => [(1, 2, 2), (4, 4, 1)]
GST(B,A) => [(0, 1, 2), (3, 3, 1)]
```

Example 1: This example illustrates the comparison of the token-string A and B. They are compared on their *Value* variable and the output of GST are the lists of (x,y,z) tiles.

Example 2.

```
A = [DEF, FUNCTION, BRACKET_START, BRACKET_END, COLON]
B = [FUNCTION, BRACKET_START, BRACKET_END, COLON]

GST(A,B) => [(1, 3, 3)]
GST(B,A) => [(0, 2, 3)]
```

Example 2: This example illustrates the comparison of the token-string A and B. They are compared on their *Group* variable and the output of GST are the lists of (x,y,z) tiles.

7.5 Clustering

Having a way of comparing source code and calculating a distance matrix, we now need to use the distances to generate a *optimal path*. Clustering is a way of grouping data points into separate clusters, and as we want similar objects to appear closer to each other in the final sequence, we group the objects into separate clusters. The algorithms

OPTICS with DBSCAN is used for clustering and the concepts is explained in 5.1.6 and 5.1.7.

7.5.1 Weight scaling

After clustering is done on the data points and before the TSP algorithm is run, weight scaling needs to happen. If the weights are not scaled, clustering is unnecessary and provides no benefit at all. In Figure 7.8, we see an example of the algorithm OPTICS run on 30 data points with x,y coordinates in the range [0,8] with eps=0.7 and minPts=3. It shows 4 different clusters and 9 noise points (black). Let's assume that each data point is a student submission and a teacher has graded all the submissions (the grading of the submissions is something that happens after all of the algorithms has run. But just for explanation let's assume that the points are "theoretically" given. That means that they deserve the points given to them and hopefully will be given that amount of points when graded). All the data points in each cluster got the same amount of points (this is unlikely in a actual scenario) where data points in the blue cluster got 2 points, the red cluster got 3 points, the green cluster got 4 points, and the purple cluster got 5 points. The noise points are wildcards. They could have got any amount of points, but let's assume that they share received a similar amount of points relative to their nearest cluster.

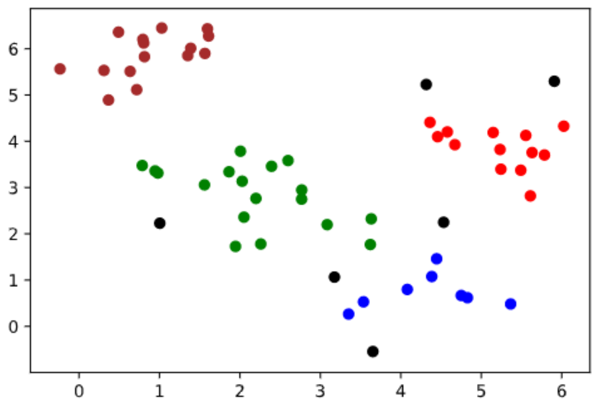


Figure 7.8: Example of clusters after running OPTICS algorithm with eps=0.7 and minPts=3. This illustrates four independent clusters (points colored red, blue, green and purple) and noise points (black).

Now that we have the clusters show in 7.8, we need to organize a Hamilton path between all the points. This is easy. Just draw a line from one point to another and never visit the same point twice and you end up with a sequence which a teacher can grade. But, here lies the problem. If you could have just drawn random lines between all the points you will end up with something similar to Figure 7.9. Then why do we even cluster? Let's assume that the graph in Figure 7.10 is the Hamilton path you have drawn. This could have been any arbitrary order of the points, but some thought has

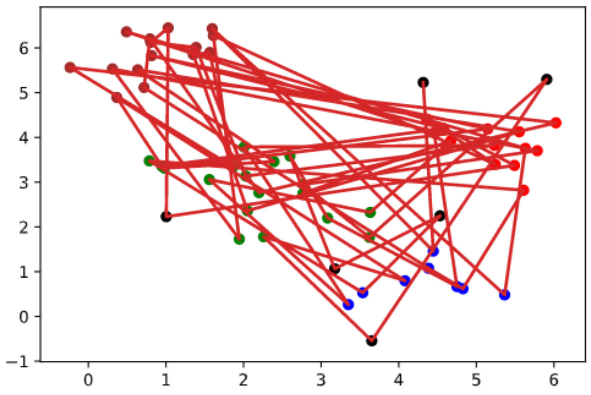


Figure 7.9: Example of a random Hamilton path drawn between each point

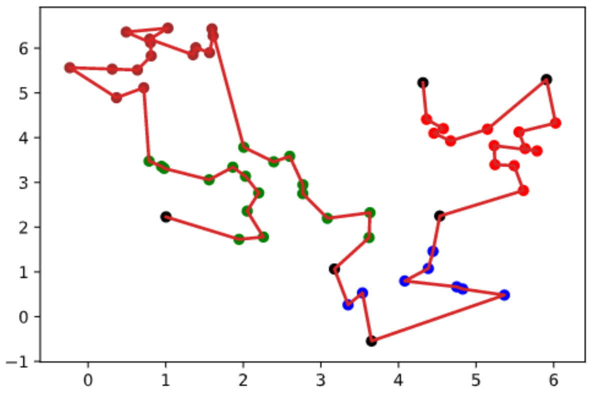


Figure 7.10: Example of a Hamilton path drawn between each point

been given to create the shortest path. This path is better than total randomness, but it has some flaws. The path starts of in the green cluster, then moves into the purple and then back into the green cluster. Looking at this, all the points could be any color and not clustered at all. The path does not respect the borders of the clusters. It can move in and out and then back into a cluster again as we can see in Figure 7.11. We want all the points inside a cluster to be followed directly after one another, not what we see in Figure 7.10, where the path cross cluster borders. What we want to see is what we see in Figure 7.12. This Hamilton path is somewhat longer than in Figure 7.10, but is more *optimal* as it respects the cluster borders.

To make the TSP respect the cluster borders, it must be given an incentive to travel all the points inside a cluster before moving on to the next. This is of course impossible to do for every arbitrary cluster graph, but should be acquired if possible. The method to I chose was to decrease the *intra-cluster-distance* and increase the *inter-cluster-distance*. That means that the distance is increased between all the points which are not in the same cluster, but decreased between all the points in the same cluster. This creates a shorter path for the TSP to order all the points in a cluster after one another and not move to another cluster before it's absolutely necessary.

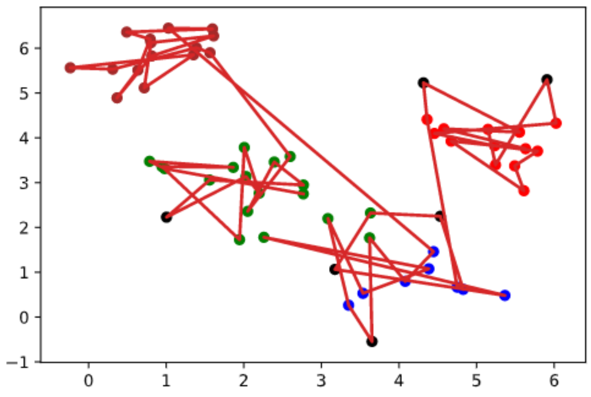


Figure 7.11: Example of a Hamilton path That does not respect the cluster borders

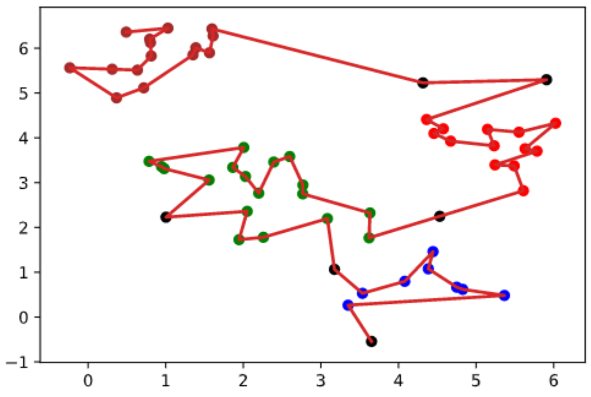


Figure 7.12: Example of a Hamilton path that respects the cluster borders

7.6 Sequence generation

As explained in 5.4, TSP finds the shortest Hamilton path between all nodes in a multidimensional graph. TSP is used to create the final output of the artefact. Since an exhaustive search requires O(n!) complexity, the greedy version is chosen, but in reality, any version of the TSP can be used. Other version such as the combination of the CONCORDE algorith and TSP was done by Agarwala et al. (2000) and delivers a complex but fast method of computing the shortest path. While the greedy version has some flaws, it was chosen because it is extremely fast. It has the approximation factor of O(log|V|) (Rosenkrantz et al., 2008) but will often yield a subotimal path.
The Sequence generation module inputs a distance matrix and calculates the shortest Hamilton path. It outputs a sequence in which should be the "optimal" path in the form of a list with positional identifiers: [5, 4, 8, n, ..., 1, 3].

Part IV

Experiments, Results, Discussion, and Conclusion

Chapter 8

Experiments

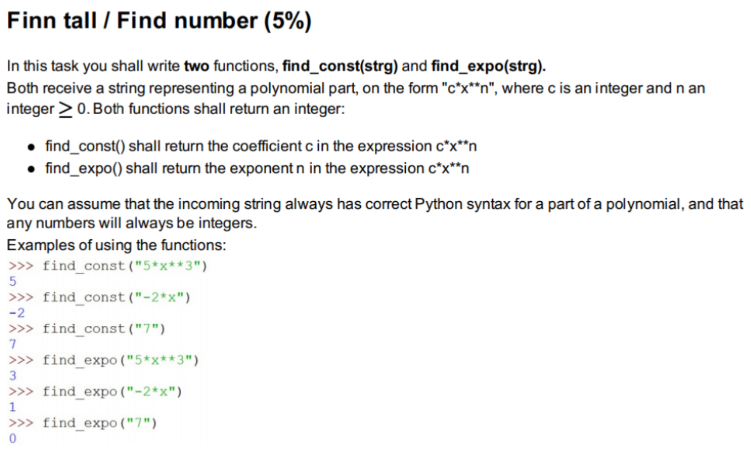
This chapter presents a detailed insight into the different experiments conducted in order to answer the research questions presented in section 1.3. Section 8.1 and 8.2 presents how the experiments were conducted. The experiment setup is described back in section 2.2.

8.1 Experiment 1

This experiment was conducted in order to generate empirical data for analysis related to **RQ 1**. The goal of this experiment was to observe the results of students acting as raters with the task of grading 20 student submissions and the effects of different sequences. The experiment solely served as a proof of concept for the second, larger experiment. Therefore, in this experiment the 10 participants were divided into a control and test group, 5 in each. The chosen submissions for this experiment were the submissions of the first 20 submissions from data source 1.

The text explaining the task for data source 1 and used in this experiment is illustrated in Figure 8.1. The proposed solution to the task is illustrated in Figure 8.2. Note that this is not the only correct answer to the task, only a proposed solution. Students can get the full score even if their solution is significantly different from the proposed solution.

Ten computer science students in 3rd grade at NTNU was randomly chosen to participate in the experiment for one hour. The participants were paid 200,- NOK for the one hour they participated. The experiment was conducted inside a classroom with ten separated desks with approximately 1,5m apart. The students were given 11 double printed pages and a pen. The first page contained the text illustrated in Figure 8.1 and the proposed solution illustrated in Figure 8.2. The first page also included guidelines for awarding the correct grades, illustrated in Table 8.1, together with an example of a student submission. The example submission was included as a warm-up exercise for the participants and used to resolve uncertainties the participants might have. The ten



Finn tall / Find number (5%)

In this task you shall write **two** functions, **find_const(strg)** and **find_expo(strg).**
Both receive a string representing a polynomial part, on the form "c*x**n", where c is an integer and n an integer $\geq$ 0. Both functions shall return an integer:

- find_const() shall return the coefficient c in the expression c*x**n
- find_expo() shall return the exponent n in the expression c*x**n

You can assume that the incoming string always has correct Python syntax for a part of a polynomial, and that any numbers will always be integers.
Examples of using the functions:

```
>>> find_const("5*x**3")
5
>>> find_const("-2*x")
-2
>>> find_const("7")
7
>>> find_expo("5*x**3")
3
>>> find_expo("-2*x")
1
>>> find_expo("7")
0
```

Figure 8.1: This text explain task 10 given on the final exam in the NTNU course TDT4127 November 2018.

```
def find_expo(expr):
    if "**" in expr:
        return int(expr.split("**")[1])
    elif "x" in expr:
        return 1
    else:
        return 0

def find_const(expr):
    if "*x" in expr:
        return int(expr.split("*x")[0])
    elif "-x" in expr:
        return -1
    elif "x" in expr:
        return 1
    else:
        return int(expr)
```

Figure 8.2: This text illustrates a proposed solution to task 10 given on the final exam in the NTNU course TDT4127 November 2018.

remaining pages had two submissions printed on them, one at each side. This arrangement resulted in when the pages stacked on top of each other with only one submission is visible at a time. The first 20 minutes were used to explain the task, the proposed solution, and other necessities for the participants to settle into the role as professional raters. The participants used the remaining 40 minutes to grade the 20 submissions.

Grade	Rationale
0	Minor/zero attempt was made. Only repeats information already given in the assignment text
1	Minor attempt, lots of mistakes or very slim code
2	Slight attempt, lots of mistakes however the student demonstrate reason or partial logic
3	good attempt, big mistakes or many minor mistakes
4	good attempt with small errors
5	perfect attempt with only minor textual mistakes

Table 8.1: This table illustrates the guidelines the participant had access to while grading the 20 submitted source codes.

To decrease variability, the participants were given a sett of rules they had to follow:

1. 2 minutes must be used for each task. If you finish early, wait until 2 minutes is used before moving to the next.
2. A grade between 0-5 **must** be assigned to each submission. No blank answers. If uncertain of which grade to assign, follow your intuition and assign a grade. No explanation must be given, only the numerical grade.
3. Do not disturb the other participants and no cooperation of any for is allowed.
4. The current page and the first explanatory page is the only two pages allowed to be visible for the participants.
5. You are not allowed to go back and view previous pages.

These rules were set to strengthen the *simulation* of being a professional rater. The 2-minute rule is for the ones who rush through without much rational thinking and to create a sense of time restraints as all raters have deadlines they need to adhere to. The reason the allowance of viewing previous papers is restricted, is to create the illusion of grading hundreds of submissions over a multitude of days. Our memory is not able to remember all previous submissions, and as the participants only graded 20 samples they needed to forget them, and was therefore not allowed to go back and revisit. Naturally, professional raters do visit previous graded submissions for quality assurance. Nevertheless, as described in Chapter 3 it seems that this does not eliminate the problem as a whole.

To increase validity, the students were assigned desks in order of their arrival in such fashion that they were placed further apart from the previous arrived person. The participants were divided into a test and a control group. The control group received the 20 submissions in the original sequence (the sequence the submissions original was graded in). The test group received the exact same 20 submissions, however the difference was that they were reordered into a new sequence. This setup gave five participants grading the original sequence and five participants grading a reordered sequence.

Before the experiment, the five test sequences and the five control sequences were shuffled into one pile. When the experiment started, the papers were handed out to the participants from the top of the pile creating a random distribution of participants. There was no other difference between the handed out papers than the sequence. The participants had no way of knowing which group they were assigned to or even that there were groups at all.

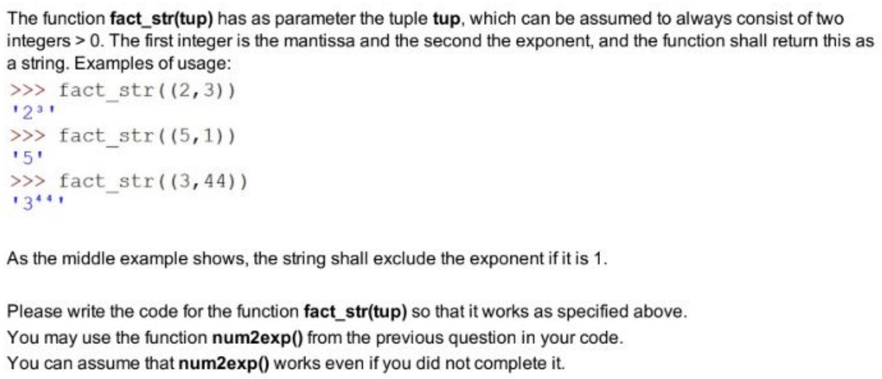
8.1.0.1 Reordering the sequence

The method of generating the sequence used by the test group was not a priority in this experiment. The sequence was automatically reordered by using a slightly different design than what is explained as the design of the artefact in chapter 7. As this experiment was conducted before the final design was implemented, a simpler algorithm was used. This algorithm provides a simple solution using Levenshtein Distance as explained in 7.4, and and TSP to generate a sequence. No use of tokens or clustering. The sequence generated from using Levenshtein Distance and TSP can be illustrated by the following list: [10, 19, 2, 9, 14, 20, 4, 15, 1, 11, 17, 6, 16, 3, 5, 12, 13, 7, 18, 8]

8.2 Experiment 2

This experiment was conducted in order to generate empirical data for analysis related to **RQ 1**. The goal of this second experiment was to extend experiment 1 with a more significant amount of participants and a well constructed optimal sequence of submissions. As described in 2.2.3, this experiment was conducted with a digital questionnaire with 40 participants. As in experiment 1, the participants were divided into a control group and a test group at random. When the participants engaged in the questionnaire, the first choice they made was to select between two different hyperlinks sending them to either the control or test questionnaire. No information was given as to what separated the two links, only that they should choose one either by flipping a coin or simply by selecting. The only difference between the questionnaires was the sequence of the submissions. This randomness validly assigned participants to a random group, resulting in 24 participants in the control group and 16 participants in the test group. The questionnaires can be found in appendix C.

The collection of random anonymous participants were done in the following way: E-mails containing information and instructions were posted to many different student organizations who is known to contain students with knowledge of Python. The organizations served as a proxy and asked their student members to participate. The participants were paid 200,- NOK for a fully completed questionnaire, and it was stated it would take between 40-60 minutes of their time to complete. The proxy then only informed with the number of participants as it made the whole process anonymous. The text explaining the task chosen for this experiment is illustrated in Figure 8.3. The proposed solution to the task is illustrated in Figure 8.4.


The function **fact_str(tup)** has as parameter the tuple **tup**, which can be assumed to always consist of two integers > 0. The first integer is the mantissa and the second the exponent, and the function shall return this as a string. Examples of usage:

```
>>> fact_str((2,3))
'2³'
>>> fact_str((5,1))
'5'
>>> fact_str((3,44))
'3⁴⁴'
```

As the middle example shows, the string shall exclude the exponent if it is 1.

Please write the code for the function **fact_str(tup)** so that it works as specified above.
You may use the function **num2exp()** from the previous question in your code.
You can assume that **num2exp()** works even if you did not complete it.

Figure 8.3: This text explain task 4 given on the final exam in the NTNU course TDT4127 November 2019.

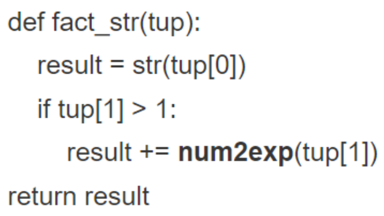

```
def fact_str(tup):
    result = str(tup[0])
    if tup[1] > 1:
        result += num2exp(tup[1])
return result
```

Figure 8.4: This text illustrate a proposed solution to task 4 given on the final exam in the NTNU course TDT4127 November 2019.

8.2.0.1 Reordering the sequence

This experiment measured the effects of reordering the sequence of submissions before grading. The experiment's only measure was changing the independent variable "order" to observe the effects of the dependent variable "grade". To analyze the relationship between the cause and effect and assure as few possible undesirable causes which could affect the result, no other variables than the order was changed.

The sequence was not created solely by the thesis design explained in 7, but in combination with a human-based opinion. This combination created an "optimal" sequence as it was first reordered by the thesis-design algorithm and the further reordered by a human. The following figures illustrates the difference between the sequence of the control and test group. Figure 8.5 illustrates the grades given by the original professional rater who graded the exam in 2019. Figure 8.6 illustrates the same submissions and their grades, only ordered into the "optimal" sequence. This "optimal" sequence can be illustrated by the following list: [15, 17, 7, 8, 0, 4, 16, 18, 5, 9, 14, 1, 11, 19, 13, 20, 6, 2, 12, 10].

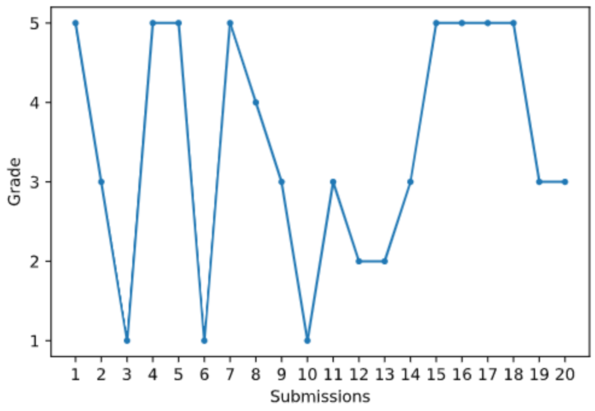


Figure 8.5: This figure illustrates the grades given to the submissions (1-20) after the final exam in the NTNU subject TDT4127 held in November 2019. These grades is the original grades given by a professional rater after the exam. The sequence of this figure is the **original sequence** they were graded in.

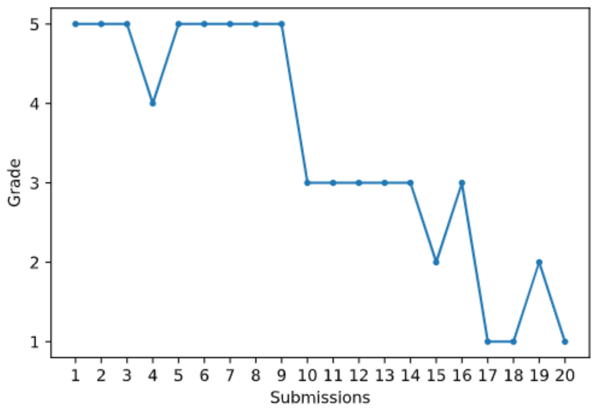
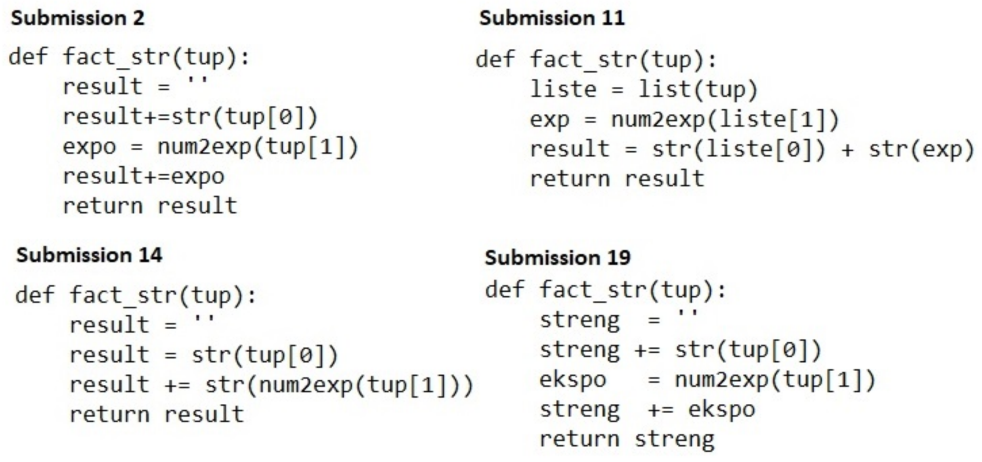


Figure 8.6: This figure illustrates the grades given to the submissions (1-20) after the final exam in the NTNU subject TDT4127 held in November 2019. These grades is the original grades given by a professional rater after the exam. The sequence of this figure is the **optimal sequence** generated by a combination of the thesis design and a human opinion. Note that the grades have not changed since Figure 8.5, only the sequence.

8.2.1 Inconsistencies

As of measuring if equal submissions receive equal grades, four near equal submissions was placed in the sequence. These were the submissions: 2, 11, 14 and 19 and illustrated in figure 8.7



Submission 2

```
def fact_str(tup):
    result = ''
    result+=str(tup[0])
    expo = num2exp(tup[1])
    result+=expo
    return result
```

Submission 11

```
def fact_str(tup):
    liste = list(tup)
    exp = num2exp(liste[1])
    result = str(liste[0]) + str(exp)
    return result
```

Submission 14

```
def fact_str(tup):
    result = ''
    result = str(tup[0])
    result += str(num2exp(tup[1]))
    return result
```

Submission 19

```
def fact_str(tup):
    streng  = ''
    streng += str(tup[0])
    ekspo   = num2exp(tup[1])
    streng  += ekspo
    return streng
```

Figure 8.7: Four samples from data source 2 judged equal or near equal. All submissions got 3 points from the originial grader grading the exam.

8.3 Experiment 3

This experiment was conducted in order to generate empirical data for analysis related to **RQ 2**. The goal of the third experiment was to record the output of the developed artefact described in chapter 7, when provided with different data material. As this experiment did not make usage of human participants, it only made use of the artefact ant the material from source 2 to 6 (2.2.1). The artefact was run once for every data source creating six different sets of empirical data.

Chapter 9

Results

This chapter presents the empirical data generated by the experiments described in chapter 8 and the analysis conducted on the data. Section 9.1 and 9.2 presents a detailed overview and quantitative analysis of the empirical data generated of Experiment 1 & 2, and section 9.3 presents the results of the artefact.

9.1 Results - Experiment 1

This is the data generated from experiment 1 described in section 8.1 and presented as numerical values in Table 9.1. The numerical values in the table are the grades the participants of the experiment gave to the 20 Python source code submissions from **data source 1** as if they were grading the exam. The table is split into three parts: *control, original rater,* and *test.* The columns of the tables are the participants (Rater1 as R.1, original rater as R.O) and table rows are the python source code (Submission 1 as S.1), where each row represent one submission. The values for the test group were originally created in the the optimal sequence, but reordered back into the original for ease of comparison. While the grades given by the original rater are **not** used in the experiment, they are added as a reference to illustrate the opinion of an actual professional rater on the quality of the submissions.

	Control						**Test**				
	R.1	**R.2**	**R.3**	**R.4**	**R.5**	**R.O**	**R.6**	**R.7**	**R.8**	**R.9**	**R.10**
S.1	1	1	3	3	1	2	3	1	1	1	4
S.2	3	0	4	3	4	3	5	3	3	3	4
S.3	3	4	2	4	3	4	5	3	2	2	4
S.4	1	0	3	2	1	1	2	2	3	3	1
S.5	2	1	5	2	3	2	3	3	1	1	2
S.6	1	0	1	1	2	1	2	3	2	2	1
S.7	3	1	4	2	3	3	1	2	1	1	3
S.8	2	2	4	2	2	2	4	2	2	2	3
S.9	2	3	3	3	3	2	4	3	4	4	5
S.10	2	2	1	3	2	2	4	3	2	2	2
S.11	1	0	2	3	3	2	4	4	3	3	5
S.12	4	3	4	3	3	3	3	2	1	1	1
S.13	4	2	3	5	4	4	2	1	2	2	0
S.14	2	2	3	3	3	2	4	3	3	3	3
S.15	1	1	2	3	2	2	4	3	3	3	3
S.16	2	0	1	3	2	1	4	4	3	3	2
S.17	2	0	3	3	4	2	3	4	5	5	5
S.18	2	2	4	4	3	3	4	3	2	2	2
S.19	1	1	3	3	4	2	2	3	3	3	3
S.20	1	1	1	3	2	1	3	2	2	2	2

Table 9.1: This table illustrates the grades given by the Experiment 1 participants to 20 submissions of Python source code. The table is split into control and test group with the original professional rater grading the actual exam as reference (R.O).

9.1.1 Experiment 1 - Descriptive Analysis

Table 9.2 illustrate the descriptive analysis of the empirical data generated from experiment 1. It illustrates the **mean**, **median** and **standard deviation** for the control and test group. The average values for each column at the bottom suggests that the *test* group gave higher grades on average and had a higher inter-rater reliability (i.e lower standard deviation).

	Mean		Median		Standard Deviation	
	Control	**Test**	**Control**	**Test**	**Control**	**Test**
S.1	1.8	2	1	1	1.095	1.414
S.2	2.8	3.6	3	3	1.643	0.894
S.3	3.2	3.2	3	3	0.837	1.304
S.4	1.4	2.2	1	2	1.14	0.837
S.5	2.6	2	2	2	1.517	1.000
S.6	1	2	1	2	0.707	0.707
S.7	2.6	1.6	3	1	1.14	0.894
S.8	2.4	2.6	2	2	0.894	0.894
S.9	2.8	4	3	4	0.447	0.707
S.10	2	2.6	2	2	0.707	0.894
S.11	1.8	3.8	2	4	1.304	0.837
S.12	3.4	1.6	3	1	0.548	0.894
S.13	3.6	1.4	4	2	1.14	0.894
S.14	2.6	3.2	3	3	0.548	0.447
S.15	1.8	3.2	2	3	0.837	0.447
S.16	1.6	3.2	2	3	1.14	0.837
S.17	2.4	4.4	3	5	1.517	0.894
S.18	3	2.6	3	2	1	0.894
S.19	2.4	2.8	3	3	1.342	0.447
S.20	1.6	2.2	1	2	0.894	0.447
AVG	**2.34**	**2.71**	**2**	**3**	**1.020**	**0.829**

Table 9.2: Experiment 1 - Descriptive analysis

9.1.2 Experiment 1 - Inferential Analysis

As all grades analysed is ordinal, statistical analysis is done by nonparametric methods. Wilcoxon–Mann–Whitney two-sample rank-sum test (U) can be used to investigate whether two independent samples were selected from populations having the same distribution ($H_0 : \sim X_1 = \sim X_2$, $H_1 : \sim X_1 \neq \sim x_2$, $n_1 = n_2 = 5$) (McKnight and Najab, 2010). Such a test was done for each submission between the groups creating 20 independent comparisons. Table 9.3 illustrate the U value, the probability significance p-value (two-tailed) and a common language effects size value f (McGraw and Wong, 1992) generated from the tests.

SUB	S.1	S.2	S.3	S.4	S.5	S.6	S.7	S.8	S.9	S.10
U	11.5	10.0	12.0	7.0	10.0	4.0	6.0	10.5	2.0	8.0
p	0.4055	0.2861	0.4568	0.1164	0.2938	0.0291	0.0772	0.3028	0.0088	0.1439
f	0.46	0.4	0.48	0.28	0.4	0.16	0.24	0.42	0.08	0.32
SUB	**S.11**	**S.12**	**S.13**	**S.14**	**S.15**	**S.16**	**S.17**	**S.18**	**S.19**	**S.20**
U	2.0	1.5	1.5	6.0	2.0	3.0	2.5	9.5	11.0	6.5
p	0.0116	0.008	0.0087	0.0466	0.0092	0.0206	0.0158	0.2481	0.3615	0.0868
f	0.08	0.06	0.06	0.24	0.08	0.12	0.1	0.38	0.44	0.26

Table 9.3: This table illustrate multiple Mann–Whitney two-sample rank-sum test between each submission for the control and test group from Experiment 1. SUB is the submissions, U is the test value, p is the probability and f is the common language effect size.

As many of the p-values in Table 9.3 suggest different significance were some rejects H_0 and some do not, the probability of statistical type I errors (false positive) is highly probable. As the inflation of probability of type I error increases with the increase in the number of comparisons, a *Bonferroni correction* can be used to counteract this problem ($\alpha = 0.05$ to $\alpha/20 = 0.0025$). With this, none of the p-values is significant after the correction suggesting a true null. However, a *Fisher's combined probability test* can be used to combine all the independent p-values to test for a global null hypothesis $H_0 : \sim X_1 = \sim X_2$ to a $p <$ 1.94e-08, rejecting the null and suggesting the distribution is unequal between the groups.

9.1.2.1 Inter and intra-rater reliability

Related to **RQ 1.3** (*Does an optimal sequence increase inter-rater or intra-rater reliability?*), Inter-rater reliability can be measured by in the internal inter-rater variance of the groups. To measure a significant difference of variance, the difference of between-group variance for each submission ($H_0 : \sigma_1 = \sigma_2$, $H_1 : \sigma_1 \neq \sigma_2$, $n_1 = n_2 = 5$) was calculated by a Levene's test (W) (Derrick et al.). Table 9.4 illustrate the test values W, the p-values and the common language effect size f.

SUB	S.1	S.2	S.3	S.4	S.5	S.6	S.7	S.8	S.9	S.10
W	0.062	0.348	1.0	0.2	0.118	0.0	0.133	0.125	0.4	0.182
p	0.8089	0.5716	0.3466	0.6666	0.7404	1.0	0.7245	0.7328	0.5447	0.6811
f	0.002	0.014	0.04	0.008	0.005	0.0	0.005	0.005	0.016	0.007
SUB	**S.11**	**S.12**	**S.13**	**S.14**	**S.15**	**S.16**	**S.17**	**S.18**	**S.19**	**S.20**
W	1.0	0.182	0.133	0.4	1.6	0.2	0.348	0.2	2.667	0.8
p	0.3466	0.6811	0.7245	0.5447	0.2415	0.6666	0.5716	0.6666	0.1411	0.3972
f	0.04	0.007	0.005	0.016	0.064	0.008	0.014	0.008	0.107	0.032

Table 9.4: This table illustrate multiple Levene's test of equal variance between each submission for the control and test group from Experiment 1. SUB is the submissions, W is the test value, p is the probability and f is the common language effect size.

As none of the p-values in Table 9.4 is significant even before a *Bonferroni correc-*

tion and a *Fisher's combined probability test* is calculated to $p = 0.975$, it suggests that $H_0(\sigma_1 = \sigma_2)$ is true. This suggests that there is no significant difference of internal inter-rater reliability between the groups. As the standard deviation is different between the groups as seen in Table 9.2, a T-test can be used as the standard deviation comes from a normal distribution with a p = 0.053 (two-tailed), not rejecting the null and further suggesting H_0 is true.

Measuring *intra-rater reliability* is difficult, nevertheless it is measured in this thesis by the *inconsistencies in equal grading of equal submissions* for each rater (related to **RQ 1.2**), and *serial position effect* (as done by Kramer (2017). Does submissions earlier/later in the sequence tend to get a higher/lower grade?). As this experiment did not contain any equal submissions, inconsistencies could not be measured. However, *Serial position* is measured by the Spearman's *rho* correlation coefficient (*rho*) for the monotonic relationship between the position of the submissions (1 to 20) and the grades (0 to 5).

The correlation coefficients and the significance of the correlation coefficient is presented in Table 9.5. The *mean* with a 95% confidence level interval for both groups show near zero correlation, suggesting that there is no significant instance of serial position effect in any of the groups. However, some of values are significant as a correlation coefficient > 0.3 or < -0.3 suggest a significant correlation, the Fisher's p-value is not significant for any of the groups ($p = 0.404$ (control), $p = 0.290$ (test)).

Control						
Rater	**R.1**	**R.2**	**R.3**	**R.4**	**R.5**	**Mean**
rho	-0.15	0.033	-0.241	0.363	0.231	0.047[-0.186, 0.264]
p	0.529	0.891	0.305	0.116	0.327	
Test						
Rater	**R.6**	**R.7**	**R.8**	**R.9**	**R.10**	**Mean**
rho	-0.077	0.275	0.327	0.327	-0.135	0.143[-0.044, 0.364]
p	0.747	0.24	0.159	0.159	0.57	

Table 9.5: Spearman's *rho* correlation coefficient for serial position. The mean and a 95% confidence interval is calculated and show a near zero correlation for both groups.

9.1.2.2 Contrast effect

To evaluating the existence of contrast effect (related to **RQ 1.1**), a Spearman's *rho* correlation coefficient is calculated to measure the monotonic relationship between the N-1 submission grades for all participants (does the directly following grade tend to be affected of the previous?). The correlation coefficient, the mean with a 95% confidence interval and the p values is presented in Table 9.6. Both groups show near zero correlation and suggest that there is no significant instance of contrast effects in any of the groups. Even though participant 4 have a significant correlation, the Fisher's p-value is not significant for any of the groups (control$p = 0.429$, test$p = 0.832$).

Control						
Rater	**R.1**	**R.2**	**R.3**	**R.4**	**R.5**	**Mean**
rho	-0.065	-0.09	-0.116	0.428	-0.272	-0.023[-0.391, 0.200]
p	0.792	0.713	0.637	0.067	0.261	
Test						
Rater	**R.6**	**R.7**	**R.8**	**R.9**	**R.10**	**Mean**
rho	0.149	0.161	-0.143	-0.143	-0.118	-0.019[-0.158, 0.121]
p	0.542	0.511	0.561	0.561	0.631	

Table 9.6: Spearman's *rho* correlation coefficient for N-1 submission grades, testing for contrast effect. The mean and a 95% confidence interval is calculated and show a near zero correlation for both groups.

9.2 Results - Experiment 2

This is the empirical data from experiment 2 described in section 8.2 and presented as numerical values in two different tables, Table 9.8 (control) and Table 9.9 (test). The numerical values in the tables are the grades the participants of the experiment gave to the 20 Python source code submissions from **data source 2** as if they were grading the exam. Table 9.9 is reordered back into original order as in Table 9.1 for ease of measure. As for the experiment's validity, the participants was chosen randomly and were asked to select which survey they would answer by themselves. This created a skewed difference in the number of participants in the test and control group. 24 participants answered the control survey while 16 participants answered the test survey, making 40 participants in total.

As the empirical data generated for this experiment was conducted using a self-administrative digital questionnaire, the time the participator used was measured as well. The time is calculated from when a participator opened the questionnaire to submitting the final answer. Table 9.7 presents the time measure for each rator (Hour = h and minute = m). As the participants were strictly instructed to use 40 minutes in total to grade all the tasks, and the 10-20 minutes it took them to read the instructions, all significantly lower amount of time used than this suggests for invalid data. The participants were incentivized by payment to participate in the experiment, and some might therefore not be interested in the experiment itself and rush through in few minutes with irrational grading and therefore invalid grades. Because of these threats to the validity of the experiment, removal of the data generated by a subset of the participants was done based upon their time used. The removed participants include: R.2, R.14, R.16, R.17, R.22, R.23, R.30, R.31, R.38. This caused a reduction in the sample size for each groups, effectively reducing the test group down to 13 participants and the control group down to 18 participants. R.33 and R.39 have significantly higher amounts of time used, however this does not suggest directly invalid grades and is therefore not removed.

Rater	**R.1**	**R.2**	**R.3**	**R.4**	**R.5**	**R.6**	**R.7**	**R.8**
TIME USED	51m	22m	34m	28m	36m	1h 34m	33m	37m
Rater	**R.9**	**R.10**	**R.11**	**R.12**	**R.13**	**R.14**	**R.15**	**R.16**
TIME USED	59m	35m	1h 26m	51m	37m	3m	1h 28m	22m
Rater	**R.17**	**R.18**	**R.19**	**R.20**	**R.21**	**R.22**	**R.23**	**R.24**
TIME USED	15m	32m	1h 5m	37m	1h 7m	20m	23m	29m
Rater	**R.25**	**R.26**	**R.27**	**R.28**	**R.29**	**R.30**	**R.31**	**R.32**
TIME USED	29m	30m	38m	26m	1h 56m	10m	21m	36m
Rater	**R.33**	**R.34**	**R.35**	**R.36**	**R.37**	**R.38**	**R.39**	**R.40**
TIME USED	5hr 7m	40m	38m	32m	36m	16m	48h 40m	30m

Table 9.7: Time used by participants in experiment 2. The raters R.2, R.14, R.16, R.17, R.22, R.23, R.30, R.31, R.38 are removed as their used time is significantly low and suggest non valid values.

	R.1	R.2	R.3	R.4	R.5	R.6	R.7	R.8	R.9	R.10	R.11	R.12	R.13	R.14	R.15	R.16	R.17	R.18	R.19	R.20	R.21	R.22	R.23	R.24
S.1	5	5	5	5	5	5	5	5	5	5	5	5	5	4	5	5	5	5	5	5	5	4	4	4
S.2	3	2	3	3	3	3	2	3	3	3	3	3	3	2	2	2	2	2	3	2	4	2	1	2
S.3	1	0	0	2	3	2	2	3	1	2	1	1	2	2	0	1	2	2	2	1	2	2	2	3
S.4	5	5	4	5	4	5	5	5	5	5	5	5	5	5	4	5	4	4	5	5	5	4	5	2
S.5	5	3	3	5	4	5	5	4	3	5	3	4	5	5	5	5	3	3	3	4	4	3	3	4
S.6	1	1	0	2	2	1	2	2	2	1	2	1	1	3	0	1	2	3	1	0	2	3	2	3
S.7	5	5	4	5	4	4	5	4	5	5	5	5	5	4	5	5	5	5	5	5	5	4	4	5
S.8	3	3	4	5	5	4	5	4	3	4	4	5	4	2	3	3	3	4	5	3	5	4	5	4
S.9	3	4	4	4	3	4	4	4	3	3	4	4	4	3	2	2	2	5	4	3	4	3	3	1
S.10	2	0	0	2	2	2	2	2	3	1	0	1	3	5	0	0	0	3	2	0	2	3	2	1
S.11	1	3	2	3	3	3	3	3	2	3	3	3	4	4	3	2	3	4	3	2	4	2	3	2
S.12	2	4	2	3	4	2	3	4	3	3	4	4	2	1	2	3	2	4	5	2	4	2	5	2
S.13	3	1	1	2	2	2	2	3	2	3	3	2	1	5	1	2	3	3	3	0	3	3	5	3
S.14	1	3	3	3	3	3	2	3	3	3	4	3	3	4	3	2	3	4	3	2	4	3	3	3
S.15	5	5	5	5	4	4	5	5	5	5	5	5	5	2	5	5	5	5	5	4	5	4	5	5
S.16	5	5	5	5	4	5	5	5	5	5	5	5	5	3	5	5	5	5	5	5	5	4	5	5
S.17	5	3	4	5	4	4	5	5	4	4	5	5	3	5	4	5	5	5	5	4	5	4	5	1
S.18	5	5	5	5	4	5	5	5	5	5	5	5	5	4	5	5	5	5	5	4	5	2	3	5
S.19	3	2	3	3	2	3	3	3	3	3	3	3	3	1	3	2	3	3	3	2	4	3	2	3
S.20	2	2	0	3	3	2	2	2	2	2	1	2	3	2	1	2	2	2	3	0	3	2	3	4

Table 9.8: Empirical data Experiment 2. Control group

	R.25	R.26	R.27	R.28	R.29	R.30	R.31	R.32	R.33	R.34	R.35	R.36	R.37	R.38	R.39	R.40
S.1	5	5	4	4	4	5	4	4	5	5	4	5	5	5	5	5
S.2	3	4	5	5	3	4	4	4	4	4	3	5	4	5	3	5
S.3	5	5	5	4		4	2	3	5	5	5	5	5	5	5	5
S.4	3	4	3	3	4	3	3	3	3	3	5	5	3	5	3	3
S.5	5	5	5	5	5	4	5	5	5	5	5	5	5	5	5	5
S.6	5	5	5	4	4	4	4	3	5	5	4	5	5	5	4	5
S.7	5	5	5	5	4	5	3	4	5	5	5	5	5	5	5	5
S.8	5	5	5	5	5	5	4	5	5	5	5	5	5	5	5	5
S.9	5	5	3	5	4	3	2	5	5	5	2	4	5	2	5	3
S.10	4	4	4	3	3	2	2	3	4	4	2	4	3	3	3	4
S.11	3	3	2	2	1	2	2	2	2	2	0	2	2	4	3	3
S.12	3	3	2	2	2	2	3	2	3	2	1	2	2	4	3	3
S.13	3	3	2	2	2	2	3	2	2	2	3	2	2	4	3	3
S.14	3	3	2	2	2	3	3	2	3	2	2	2	2	4	3	3
S.15	2	3	2	1	2	2	3	2	4	2	1	2	3	3	3	2
S.16	3	2	1	1	2	4	4	1	2	2	1	1	4	3	2	3
S.17	2	1	1	1	3	1	3	1	1	2	1	1	1	2	1	2
S.18	2	1	2	1	1	3	1	0	0	2	2	0	0	2	1	2
S.19	2	3	3	3	3	3	5	2	3	2	3	0	3	4	3	4
S.20	2	2	2	1	3	2	4	1	0	0	1	0	3	2	1	2

Table 9.9: Empirical data Experiment 2. Test group

9.2.1 Experiment 2 - Descriptive Analysis

Table 9.10 illustrate the descriptive analysis of the empirical data generated from experiment 2. It illustrates the **mean**, **standard deviation**, **median** and **mode** for the control and test group. The average vales for each column at the bottom suggests that the *control* group gave higher grades on average, however the test group had a higher inter-rater reliability (i.e lower within standard deviation).

	Mean		Standard deviation		Median		Mode	
	Control	**Test**	**Control**	**Test**	**Control**	**Test**	**Control**	**Test**
S.1	4.944	5.000	0.236	0.000	5	5	5	5
S.2	2.778	2.308	0.548	0.630	3	2	3	2
S.3	1.667	1.077	0.907	0.862	2	1	2	2
S.4	4.611	4.538	0.778	0.660	5	5	5	5
S.5	4.111	4.308	0.832	1.032	4	5	5	5
S.6	1.444	1.385	0.922	0.650	2	1	2	1
S.7	4.778	4.615	0.428	0.768	5	5	5	5
S.8	4.111	3.462	0.758	0.776	4	3	4	3
S.9	3.500	3.462	0.924	0.660	4	4	4	4
S.10	1.556	1.385	1.042	1.044	2	1	2	2
S.11	2.833	2.385	0.786	0.506	3	2	3	2
S.12	3.056	2.615	0.998	0.961	3	3	2	3
S.13	2.167	2.231	0.924	0.832	2	2	3	2
S.14	2.944	2.077	0.725	0.862	3	2	3	2
S.15	4.833	4.615	0.383	0.506	5	5	5	5
S.16	4.944	4.846	0.236	0.376	5	5	5	5
S.17	4.278	4.077	1.018	0.760	5	4	5	4
S.18	4.889	5.000	0.323	0.000	5	5	5	5
S.19	2.944	2.385	0.416	0.506	3	2	3	2
S.20	2.056	1.923	1.056	0.954	2	2	2	2
AVG	3.422	3.185	0.712	0.667	4	3	4	3

Table 9.10: Experiment 2 - Descriptive analysis of the empirical data illustrated in Table 9.8 and Table 9.9

9.2.2 Experiment 2 - Inferential Analysis

As in 9.1.2, a Mann–Whitney (U) test can be used to investigate whether the two independent groups were selected from populations having the same distribution (H_0 : $\sim X_1 = \sim X_2$, $H_1 : \sim X_1 \neq \sim x_2$, $n_1 = 13, n_2 = 18$). Such a test was done for each submission between the groups creating 20 independent comparisons. Table 9.11 illustrate the U value, the probability significance p-value (two-tailed) and a common language effects size value f generated from the tests.

SUB	**S.1**	**S.2**	**S.3**	**S.4**	**S.5**	**S.6**	**S.7**	**S.8**	**S.9**	**S.10**
U	110.5	72.5	77.0	105.0	97.0	108.5	112.0	64.0	108.0	105.0
p	0.0293	0.1932	0.449	0.0018	0.0891	0.1844	0.1343	0.1108	0.0016	0.257
f	0.472	0.31	0.329	0.449	0.415	0.464	0.479	0.274	0.462	0.449
SUB	**S.11**	**S.12**	**S.13**	**S.14**	**S.15**	**S.16**	**S.17**	**S.18**	**S.19**	**S.20**
U	74.0	96.5	114.0	50.5	91.5	105.5	91.5	104.0	55.5	101.5
p	0.1977	0.0217	0.0452	0.279	0.1938	0.3562	0.3919	0.0117	0.3437	0.3078
f	0.316	0.412	0.487	0.216	0.391	0.451	0.391	0.444	0.237	0.434

Table 9.11: This table illustrate multiple Mann–Whitney two-sample rank-sum test between each submission for the control and test group from Experiment 2. SUB is the submissions, U is the test value, p is the probability and f is the common language effect size.

To counteract the inflation of probabilities of multiple comparison leding to a increased probability of type I error, the *Bonferroni correction* alpha is calculated ($\alpha = 0.05$ to $\alpha/20 = 0.0025$). Even with this correction, two samples where calculated with p < .0025, suggesting that the samples were from different distributions. The Fisher's $p =$ 1.197e-06 further suggests the difference in distribution and a rejection of the null.

9.2.2.1 Inter and intra-rater reliability

Related to **RQ 1.3** (*Does an optimal sequence increase inter-rater or intra-rater reliability?*), Inter-rater reliability can be measured by in the internal inter-rater variance of the groups. To measure a significant difference of variance, the difference of between-group variance for each submission ($H_0 : \sigma_1 = \sigma_2$, $H_1 : \sigma_1 \neq \sigma_2$, $n_1 = 13, n_2 = 18$) was calculated by a Levene's test (W). Table 9.12 illustrate the test values W, the p-values and the common language effect size f.

SUB	**S.1**	**S.2**	**S.3**	**S.4**	**S.5**	**S.6**	**S.7**	**S.8**	**S.9**	**S.10**
W	0.715	0.498	0.013	0.074	0.009	3.895	0.567	0.166	0.066	0.061
p	0.4046	0.4862	0.9088	0.7868	0.9268	0.058	0.4577	0.6869	0.7991	0.8068
f	0.003	0.002	0.0	0.0	0.0	0.017	0.002	0.001	0.0	0.0
SUB	**S.11**	**S.12**	**S.13**	**S.14**	**S.15**	**S.16**	**S.17**	**S.18**	**S.19**	**S.20**
W	0.304	1.386	0.682	0.426	1.865	0.802	0.599	1.52	1.865	0.014
p	0.5854	0.2486	0.4157	0.5193	0.1826	0.3779	0.4453	0.2275	0.1826	0.9079
f	0.001	0.006	0.003	0.002	0.008	0.003	0.003	0.006	0.008	0.0

Table 9.12: This table illustrate multiple Levene's test of equal variance between each submission for the control and test group from Experiment 2. SUB is the submissions, W is the test value, p is the probability and f is the common language effect size.

As none of the p-values in Table 9.12 is significant even before a *Bonferroni correction* and the Fisher's $p = 0.760$, it infers that $H_0(\sigma_1 = \sigma_2)$ is true. This suggests that there is no significant difference of internal inter-rater reliability between the groups. As the standard deviation is different between the groups as seen in Table 9.10, a T-test can be used as the standard deviation comes from a normal distribution with a p = 0.626

(two-tailed), not rejecting the null and further suggesting H_0 is true.

As in experiment 1, *serial position* (does submissions earlier/later in the sequence tend to get a higher/lower grade?) is measured by the Spearman's *rho* correlation coefficient by investigating the monotonic relationship between the position of the submissions (1 to 20) and the grades (0 to 5) . The correlation coefficients for the control group is presented in Table 9.13 and the test group is presented in Table 9.14. A 95% confidence level interval was calculated for each group mean. The *mean* of both groups show near zero correlation suggesting that there is no significant instance of serial position effect in any of the groups. The Fisher's *p*-value is not significant for any of the groups (control$p = 0.999$, test$p = 1$) further suggesting a true null.

Serial position effect. Control group						
Rater	**R.1**	**R.3**	**R.4**	**R.5**	**R.6**	**R.7**
rho	0.018	0.064	0.024	-0.165	-0.087	0.0
p	0.94	0.787	0.92	0.488	0.717	1.0
Rater	**R.8**	**R.9**	**R.10**	**R.11**	**R.12**	**R.13**
rho	0.007	0.046	-0.041	0.089	0.049	-0.08
p	0.977	0.848	0.864	0.71	0.836	0.739
Rater	**R.15**	**R.18**	**R.19**	**R.20**	**R.21**	**R.24**
rho	0.088	0.177	0.143	-0.142	0.077	0.162
p	0.714	0.456	0.548	0.551	0.746	0.495
Mean	**0.024[-0.0589, 0.116]**					

Table 9.13: Spearman's *rho* correlation coefficient for serial position of the control group. The mean and a 95% confidence interval is calculated and show a near zero correlation.

Serial position effect. Test group							
Rater	**R.25**	**R.26**	**R.27**	**R.28**	**R.29**	**R.32**	**R.33**
rho	-0.024	-0.037	-0.051	0.008	-0.057	0.019	-0.006
p	0.921	0.875	0.832	0.974	0.813	0.936	0.979
Rater	**R.34**	**R.35**	**R.36**	**R.37**	**R.39**	**R.40**	
rho	-0.097	-0.039	0.022	0.108	-0.002	0.151	
p	0.684	0.87	0.927	0.65	0.995	0.526	
Mean	**0.0[-0.0491, 0.082]**						

Table 9.14: Spearman's *rho* correlation coefficient for serial position of the test group. The mean and a 95% confidence interval is calculated and show a near zero correlation.

9.2.2.2 Contrast effect

To evaluating the existence of contrast effect (related to **RQ 1.1**), a Spearman's *rho* correlation coefficient is calculated to measure the linear relationship between the N-1 submission grades for all participants. The correlation coefficient for the control group is presented in Table 9.15 and the test group is presented in Table 9.16. Both that there is no significant instance of contrast effects in any of the groups. The Fisher's *p*-value is not significant for any of the groups (control$p = 0.997$, test$p = 0.995$).

Contrast effect. Control group						
Rater	**R.1**	**R.3**	**R.4**	**R.5**	**R.6**	**R.7**
rho	0.082	0.244	0.157	-0.029	0.074	0.163
p	0.739	0.314	0.521	0.906	0.765	0.504
Rater	**R.8**	**R.9**	**R.10**	**R.11**	**R.12**	**R.13**
rho	0.183	0.035	0.037	0.134	0.147	-0.098
p	0.452	0.886	0.881	0.583	0.548	0.690
Rater	**R.15**	**R.18**	**R.19**	**R.20**	**R.21**	**R.24**
rho	0.065	0.198	0.054	0.080	0.174	0.034
p	0.792	0.417	0.825	0.745	0.475	0.890
Mean	**0.096[0.021, 0.172]**					

Table 9.15: Spearman *rho* correlation coefficient for N-1 submission grades, testing for contrast effect in the control group. The mean and a 95% confidence interval is calculated and show a near zero correlation.

Contrast effect. Test group							
Rater	**R.25**	**R.26**	**R.27**	**R.28**	**R.29**	**R.32**	**R.33**
rho	-0.061	-0.027	0.079	0.143	0.116	0.101	-0.089
p	0.803	0.914	0.748	0.560	0.637	0.680	0.717
Rater	**R.34**	**R.35**	**R.36**	**R.37**	**R.39**	**R.40**	
rho	0.124	0.051	0.230	-0.197	-0.137	-0.014	
p	0.613	0.837	0.343	0.418	0.577	0.955	
Mean	**0.0073[-0.1043, 0.119]**						

Table 9.16: Spearman *rho* correlation coefficient for N-1 submission grades, testing for contrast effect in the test group. The mean and a 95% confidence interval is calculated and show a near zero correlation.

9.2.2.3 Inconsistencies

As this experiment *did* contain equal submissions, inconsistencies of equal grading could be measured. As the submissions 2, 11, 14 and 19 were equal (with only minor differences), they should receive the same grade. The standard deviation for each rater between the grades given to the four submissions should therefore be 0. However, the mean standard deviation were calculated to 0.182 (test) and 0.348 (control). 44.44% of the control group participants gave inconsistent grades to equal source codes, while only 23% of the test group did the same. This suggest that the test group had a significant lower amount of inconsistencies than the control group. Table 9.17 and Table

9.18 illustrate the grades and the standard deviation for the groups. A T-test was used to measure a significant difference of variance as the variances follow a normal distribution ($p = 0.239$, two-tailed). As this p-value is not < 0.05, the null hypothesis of equal variance ($H_0 : \sigma_1 = \sigma_2$) can not be rejected.

Inconsistencies. Control group									
Rater	**R.1**	**R.3**	**R.4**	**R.5**	**R.6**	**R.7**	**R.8**	**R.9**	**R.10**
S.2	3	3	3	3	3	2	3	3	3
S.11	1	2	3	3	3	3	3	2	3
S.14	1	3	3	3	3	2	3	3	3
S.19	3	3	3	2	3	3	3	3	3
STDEV	1.155	0.5	0	0.5	0	0.577	0	0.5	0
Rater	**R.11**	**R.12**	**R.13**	**R.15**	**R.18**	**R.19**	**R.20**	**R.21**	**R.24**
S.2	3	3	3	2	2	3	2	4	2
S.11	3	3	4	3	4	3	2	4	2
S.14	4	3	3	3	4	3	2	4	3
S.19	3	3	3	3	3	3	2	4	3
STDEV	0.5	0	0	0	0.957	0	0	0	0.577
Mean Standard deviation = 0.348									

Table 9.17: This table illustrates the grades given to the submissions 2, 11 ,14 and 19 by the control group. It also illustrates the standard deviation of the grade for each rater. All the submissions were near equal and should get the same grade and a standard deviation of 0.

Inconsistencies. Test group							
Rater	**R.25**	**R.26**	**R.27**	**R.28**	**R.29**	**R.32**	**R.33**
S.2	3	3	2	2	2	2	3
S.11	3	3	2	2	2	2	2
S.14	3	3	2	2	1	2	2
S.19	3	3	2	2	2	2	3
STDEV	0	0	0	0	0.5	0	0.577
Rater	**R.34**	**R.35**	**R.36**	**R.37**	**R.39**	**R.40**	
S.2	2	1	2	2	3	3	
S.11	2	3	2	2	3	3	
S.14	2	0	2	2	3	3	
S.19	2	2	2	2	3	3	
STDEV	0	1.291	0	0	0	0	
Mean Standard deviation = 0.182							

Table 9.18: This table illustrates the grades given to the submissions 2, 11 ,14 and 19 by the test group. It also illustrates the standard deviation of the grade for each rater. All the submissions were near equal and should get the same grade and a standard deviation of 0.

9.3 Results - Artefact

The following figures illustrates the original and optimal paths created by the artefact for datasource 2 to 6 defined in 2.2.1. For all following experiments using the artefact, the manual inputs of the variables *MinPts* and *EPS* must be set to some value. *MinPts* is chosen to be always set to 2 because if it exists two or more equal items in a data source, they should form a cluster. The *EPS* values are described in Table 9.19 and the rationale behind the *EPS* is discussed in 9.3.5.

Source	**2**	**3**	**4**	**5**	**6**
EPS	0.75	0.65	0.70	0.60	0.50

Table 9.19: EPS values chosen for the datasources when generating empirical data using the artefact

9.3.1 GST similarity and grades

Measuring *similarity* between two items is dependent on the context and the chosen features. The features that makes two apples similar is not the same features that make two source codes similar. Choosing the correct features for the artefact to measure similarity in the same manner as human would is hard. If the artefact measures similarity between source code in the same manner as a human does, the items clustered by the artefact should have similar grades. To actually test this, the artefact was made to generate clusters for each EPS value (0.01 to 1), for each datasource and average the standard deviation of the grades for each cluster (ignoring noise). This process created a total of 32,121 different clusters with an average amount of items in each cluster of 43,8 with a median of 28 items (a great illustration of this is the *EPS-graph* in Figure 9.4). The standard deviation of the grades in each cluster for each datasource was calculated and the mean of the standard deviations is presented in Table 9.20. This was done twice for each datasource, one with the actual clusters and grades and one with the same grades but with their position randomly shuffled. This randomness created a control standard deviation and the mean of this is also presented in Table 9.20.

data source	2	3	4	5	6
St.Dev cluster	0.1576	0.1445	0.2060	0.2284	0.2453
St.Dev random	0.5645	0.3110	0.8764	1.0925	1.5188

Table 9.20: This table illustrate the mean standard deviation of the grades assigned to the same cluster for all cluster generated for all EPS values (32,121 clusters). It also illustrate the same clusters, only the items are randomly assigned to the clusters instead.

The values in Table 9.20 present a much lower standard deviation value for the clustered grades. A T-test testing difference in variance ($H_0 : \sigma_1 = \sigma_2$) with $p = 0.032$, rejecting the null. This suggests that using GST to measure similarity between Python source code is very similar to how a human measures similarity between Python source code.

9.3.2 Sequence generation

The main goal of the artefact was to generate an optimal sequence given a set of Python source codes. For each datasource, the optimal sequence was generated and measured. Figure 9.1 illustrate the optimal path for datasource 2 and presents the submissions as colored dots where each color represent one cluster. The clusters are identified by a numerical number (i.e 0 to 17) while noise is included as smaller navy blue colored dots and represented by the negative number *-1*. The dots are mapped to their position in the sequence and the grade the original professional rater assigned. Figure 9.2 is a generated reachability plot after the OPTICS algorithm has run. The figures illustrating the rest of the datasources can be found in appendix B. Note that all following sequences are generated using a Greedy TSP heuristic generating sub-optimal paths.

When clustering, the goal is to cluster similar items. Related to reducing contrast effects, the goal of clustering is to cluster items of similar quality and receive similar grades. Therefore, we want each cluster to contain items who got the exact same grade, or as similar grade as possible. Figure 9.3 illustrate all the given grades to datasource 2 by the professional rater. It also illustrate all the clusters (0 to 17) and the grades for each item in the clusters by *mean* and *standard deviation.*

The graphs presented in this section are only to illustrate the sequences and the clusters as the grades is not available when the artefact is supposed to run, which is before the grading process.

9.3.3 Measuring sequence optimality

Measuring the *optimality* of a sequence is relative to how an optimal sequence is defined. In this thesis, the optimal sequence is defined in section 7.1 and is based on two key concepts: reduction of contrasting items and clustering of similarity. These two concepts produce two different measurements, *amount of contrasting items* and *path-length.* Other measurements can be used instead of or additional to the chosen measurements, as the definition of *optimality* is relative to the context of use.

9.3.3.1 Amount of contrasting items

The amount of contrasting items can be calculated by first defining what a contrasting items is, and as illustrated in Table 3.1, contrasting items are items in a sequence who are of opposite quality and directly following each other. The definition of what quality *is*, and when two items goes from being equal or similar and to being of opposite quality is outside the scope of this thesis. I have simply chosen that when two following items have a difference of two grades or more, they are defined as being of opposite quality

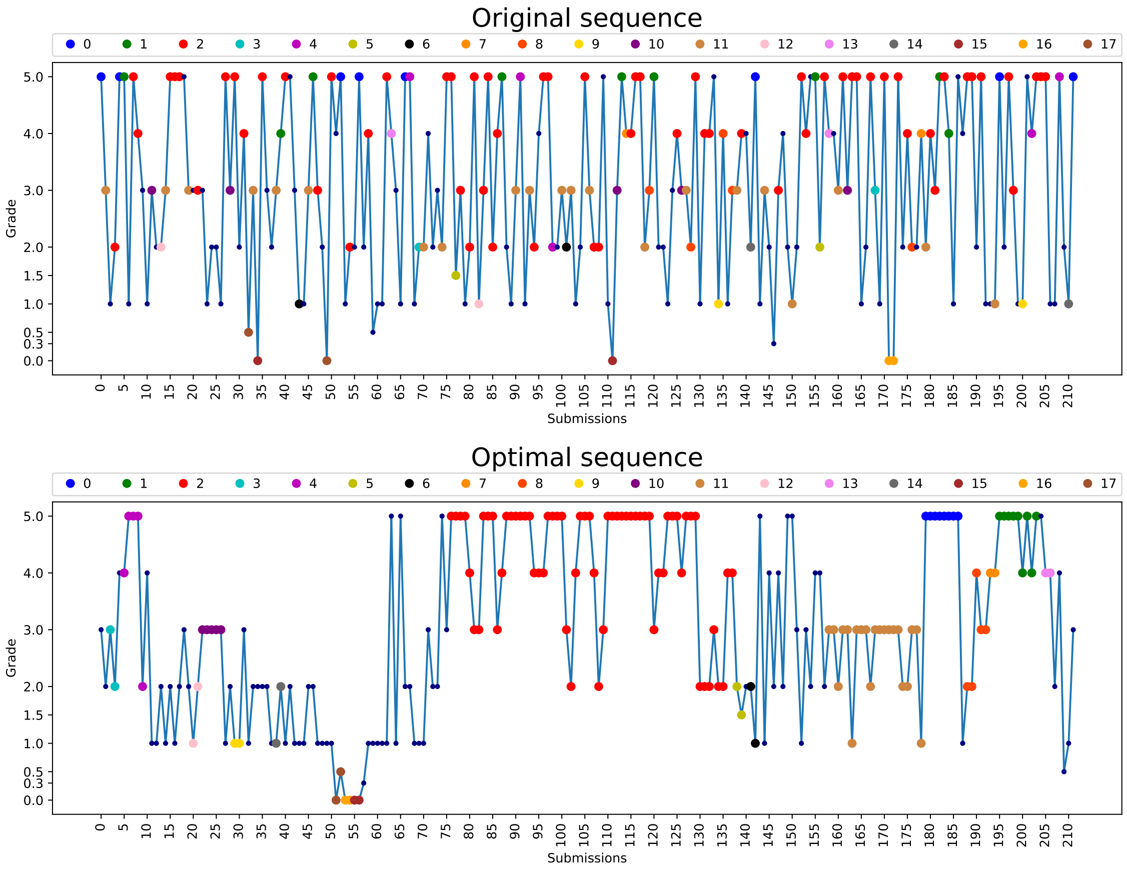


Figure 9.1: Original and optimal path (before and after the artefact has run) for **datasource 2**. Each color represent one cluster, and small navy-blue dots are noise.

(i.e (5 and 3), (4 and 2) or (5 and 1) are samples of contrasting items). The numerical difference of two grades are chosen as the research of Spear (1997) items of average quality was the ones who were effected the most, both increased and decreased in grade relative to the previous. The numerical value of two or more on a range of [0-5] might compare items of grade 3 to items of grade 1 or 5. Therefore, the **amount of contrasting items** can be calculated by the following equation where *G(S)* is the grades given in a sequence *S*:

$$\text{contrast}(G(S)) = \sum_{g_i \in G} \begin{cases} 1, & |g_i - g_{i+1}| >= 2 \\ 0, & \text{otherwise} \end{cases} \tag{9.1}$$

Where the value *contrast* is the amount of contrasting items in a sequence. Calculating the *contrast* value for each dataset yields Table 9.21. The table also includes a calculated average of 1000 pseudo-random permutations of the sequence for each data

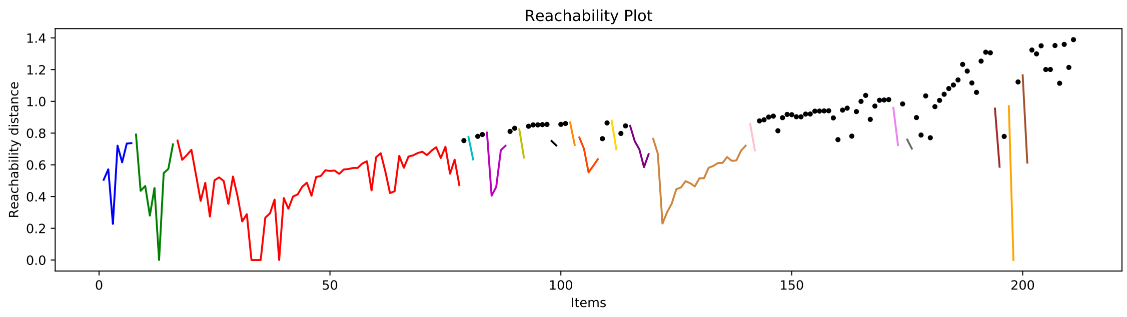
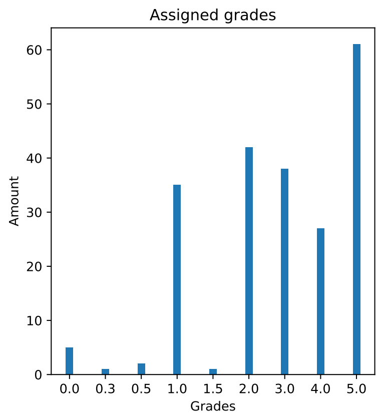


Figure 9.2: Reachability plot. Displaying the reachability distance on the y-axis creating *valleys* if items are similar. The colors represent the assignment to clusters equal to the ones used in Figure 9.1.



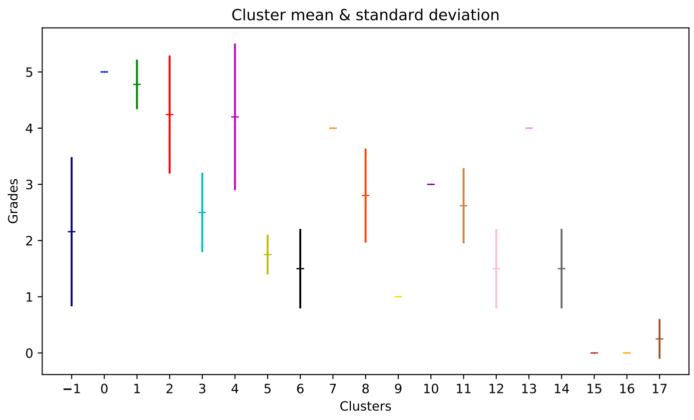


Figure 9.3: **Assign Grades** illustrates grades and their frequency given by the original rater to data source 2. **Cluster mean & standard deviation** illustrated the mean and standard deviation of grades for each cluster made. The colors and numbers on the *x-axis* are cluster identifiers and equal to the ones used in Figure 9.1. Note that the negative number "-1" is noise and not assigned any cluster.

source. Measuring significant difference of the means between the optimal and the *avg of 1000* with a T-test ($H_0 : \overline{x}_1 = \overline{x}_2$) results in $p < 0.0001$ (one-tailed), rejecting the null and suggesting that the optimal reduces the amount of contrast effects.

9.3.3.2 Path-length

Path-length is the measure of how similar each item is to the previous in a sequence. The numerical value representing the distance between two items is calculated by the

Source	optimal	original	avg of 1000
2	47	113	111.234
3	56	110	108.06
4	49	131	131.385
5	45	108	115.677
6	56	129	131.446

Table 9.21: This table illustrate how many contrasting values found for the original and optimal sequence for each data source.

artefact's distance function (i.e GST), where a lower distance means more similar items. This distance is **not** the weight scaled distance, but the distance calculate purely by the distance function. The value *Path-length* is the sum of in the in between distances from one item to the next and calculated by the following formula where S is the sequence and *dist* is the distance function:

$$\text{path-length}(S) = \sum_{d_i \in S} \text{dist}(d_i, d_{i+1}) \tag{9.2}$$

Calculating the *path-length* value for each dataset yields Table 9.22. The table also includes a calculated average of 1000 pseudo-random permutations of the sequence for each data source. Measuring significant difference of the means between the optimal and the *avg of 1000* with a T-test ($H_0 : \overline{x}_1 = \overline{x}_2$) results in $p < 0.0001$ (one-tailed), rejecting the null and suggesting that the optimal reduces the path-length.

Source	optimal	original	avg of 1000
2	50.600	93.810	93.892
3	57.625	96.958	97.794
4	66.775	115.822	116.675
5	56.795	96.965	99.707
6	41.562	89.809	91.346

Table 9.22: This table illustrate the calculated path-length for the original and optimal sequence for each data source.

9.3.4 Optimizing EPS

EPS is explained in section 5.1.6 to be the maximum distance to consider if two items is neighbours or not. The optimal distance can drastically differ amongst different densities, and is not easy to calculate. When choosing EPS values, a method is to generate a graph displaying all EPS values between 0 and 1 and generating cluster for each value (i.e a *EPS-graph*) as shown in figure 9.4. Such a graph can be generated for each datasource and by visually inspecting the graph, one can choose a EPS value. Figure 9.4 illustrates a EPS-graph generated from datasource 2. Different EPS values yields different generated optimal paths and in such, the generated optimal path is heavily dependent on the EPS value (and of course the MinPts). The generated EPS-graphs for each data-source can be found in appendix B.

At value 1 in Figure 9.4, almost all the items are clustered in cluster 0. One can argue that the goal was to cluster all the items, and therefore this is the optimal value. This is true if it was the goal. However, the goal was to cluster *similar* items, not *all* items. Dealing with large data-sets, noise is probably inevitable. When we decide that all items should be placed into one cluster, noise is mixed into the cluster as well. The *optimal* EPS is therefore the value that reduced the inclusion of noise in the clusters as well as creating individual clusters housing similar items (items with similar quality and grade). However, to reach this goal automatically is outside the scope of this thesis. The EPS values chosen in Table 9.19 were simply chosen by human intuition as they seemed to be fairly balanced between the inclusion of noise and the separation of differing items.

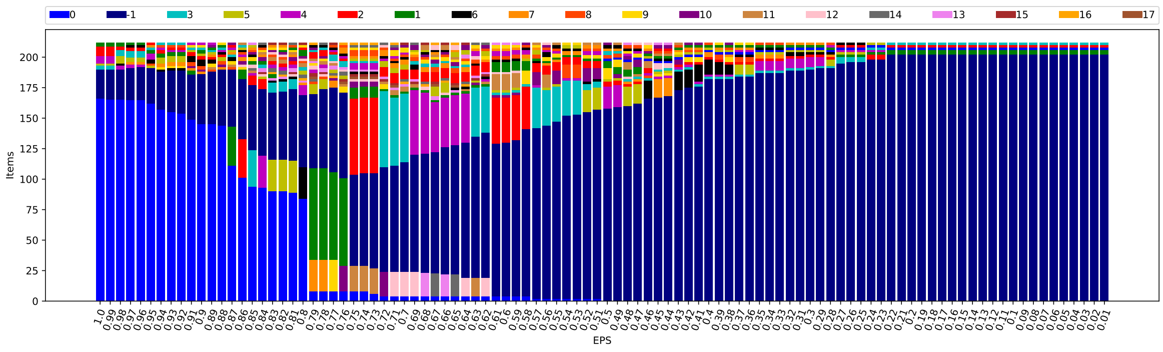


Figure 9.4: Graph illustrating all the different EPS values and the clusters generated for each value between 0.01 and 1. This graph is generated from datasource 2. Each color represent one distinct cluster. Notice the items change from almost only noise at 0.01 to almost all items in one cluster at 1.

9.3.5 Finding inconsistencies

The artefact can be used to find inconsistencies by evaluating items with low or zero distance and compare their given grades. Running the algorithm on data source 2, only **one** case were found in which the source code was near identical (with minor differences regarding syntax), but received different grades. The submissions were S.39 and S.182 with the grades 4 and 5 respectively. Of all 212 submissions of datasource 2, and for searching through all the other datasources, this was the only serious mistake i found, which in turn suggest that the raters of these exams was very valid, fair and reliable. The same test were run for all data sources (3 to 6), however no other cases of inconsistency were found.

Chapter 10

Discussion, Conclusion and Future Work

In this chapter the results in chapter 9 are discussed and a presentation of the conclusion and future work are made. Section 10.2 discuss the results from Experiment 1 and 2, and section 10.3 discuss the results from the artefact. Section 10.4 lists possible threats to the validity of the research done in this thesis. The chapter ends with a a conclusion i section 10.5 and a list of possible future work in section 10.6.

10.1 Validity of results

First of all, the results generated in chapter 9 needs to be viewed with the utmost respect towards the aspect of validity as there might be a chance that **none** of the reported descriptive and especially the inferential analysis results are valid. As statistical analysis is only a method of using models to acquire insight into data, there is no guarantee for the models used in this thesis to exactly mirror the goal of the analysis. Therefore, all results discussed in this chapter is based on the assumption that the chosen models are correct, however other models might serve a better fit. The models used in this thesis is chosen as they were the ones deemed the most suitable to generate the correctly desired insight, however they might prove to not serve this fit. There is also substantial reason to believe that type I and type II errors are present in the data. With this in mind, the results from chapter 9 can be discussed.

10.2 Findings of the experiments

In both experiments, the sample size can be argued to be of very small sizes and no significant conclusion can be drawn based on the results generated. However, it does provide insight we can discuss. Experiment 1 only served as a pilot experiment before the larger experiement 2 was conducted to gain experience and knowledge of how well the experiment would work when scaled. The results from Experiment 1 is therefore as stated earlier, not in focus but might serve some insight. The experiments was conducted to illustrate if the sequence of submissions have any effects on the grades in terms of a change in the amount of contrasting items and inconsistencies of equal grading. The experiment was also conducted to get insight into if the sequence had any effect on inter and/or intra-rater reliability.

As the results show, no significant amount of contrast effect were found in any of the groups in both experiments. In fact, no significant amount of change in inter or inter-rater reliability were found either. Even though testing for inconsistencies in experiment 2 gave a lower amount for the test group, it was not significant enough to suggest a difference between the groups. The results therefore suggests that there is no difference between the groups at any level and supports the findings of Attali et al. (2013) of no significant difference on grades based on ordering. However to state that there is no difference only based on not observing any of the measured effects might be wrong as the results from Spear (1997) and Kramer (2017) did find contrasting effects. A larger sample size might prove a significant difference as the results only suggest a non existent difference on the measured levels for a small sample size.

Related to the research questions, the results suggests that the answer to **RQ 1.1** and **RQ 1.3** can not directly be answered based on the results as no effects were found, and therefore no difference was observed. As for **RQ 1.2**, the results from this research suggest that inconsistencies are not reduced, however this might prove differently with a larger sample size.

10.3 Findings of the artefact

When evaluating the sequence, the grades are already assigned by a professional rater. So as for measuring the optimality of a sequence only serve a *hypothetical* measure. To actually measure the optimality, one needs to observe the effects of generating the sequences before they are graded, not after (as done in this thesis). Therefore, relative to **RQ 2**, to prove if the generated sequences are optimal is probably not possible after the grades are assigned. However, the high similarity between GST and how a human rater measures source code suggests that GST **can** be used to cluster equal submissions with similar grades (**RQ 2.1**). Though, to be able to prove this, a larger sample size and samples with different architecture and syntax is needed. One thing to also note is that correlation does not mean causality, so even thought the results show that the measurement of a human and GST is similar, it does not mean that is guaranteed or the similarity is purely caused by GST.

Relative to **RQ 2.2**, the optimal sequence was described in section 7.1 as a sequence which reduces contrasting items and similar items are graded directly after one another. This can be argued to be achieved as the results from section 9.3.3.1 and section 9.3.3.2 suggest that contrasting items are reduced and similar items are graded directly after one another. However, Spear (1997) show findings that contrasting effects are *increased* when multiple items of similar quality are followed by a contrasting item. This might result in the the optimal sequences generated by this artefact, does not reduce contrasting effects but actually *increases* them. This problem might be solved by using the *quality* of the source code as and additional factor when generating the sequence. The sequence can then be generated to not have contrasting clusters directly after one another, but the sequence can either start in the high quality cluster and move to lower quality cluster or vice versa. This might create a more smoothed out sequence which might reduce contrast effects. However, automatically measuring *quality* of source code might be a challenge.

Another effect might also be increased rather than decreased when using a optimal sequence. This effect is the assimilation effect and is described as the effect of treating similar items similarly when measured together rather than independent. This was the basis of the idea that equal submissions should get equal grades. However, the problem arises when a submission of low quality is similar to submissions of high quality, it might be treated equally or be affected of the mere presence of the high quality submissions. This might increase the grade of the low quality submission to an unfair extent.

Noise is also still a problem. section 9.3.5 describes how one can manually select an EPS, but not automatically select one and definitely not how to extract the optimal one. Selecting the Optimal EPS depends on the similarity measure, features used and the spacial dimension in the clusters. Even though GST might be used to as a similarity measure, it still generates a lot of noise in combination with the OPTICS algorithm. Viewing the illustration of the optimal sequence in Figure 8.6 the noise is also somewhat clustered by grades, but at a more uneven rate. If a better similarity measure, a better clustering algorithm or a better sequence generation method were used, the result might be improved. Noise is also heavily dependent on the material from the data source. If the material from one source where very different from one another, everything would be noise, or everything would be clustered. Therefore, the sequence is not only dependent on the artefact modules, but the material itself. Proving that **all** sequence can be manipulated into a optimal sequence is therefore only possible to do by the means of induction and assumptions.

One thing to note is that if the artefact is used, raters might behave differently based upon their knowledge of the usage. If a rater knows that the sequence is optimally generated and items following each other have a higher chance of receiving the same or a similar grade, an assimilation effect might occur. This means that in a sequence, a low quality submission might be affected by its position among high quality submissions and receive a similar grade as the high quality submissions, and vice versa. As well as this, the knowledge of the usage of the artefact might also serve as an incentive for the rater to increase focus in details, but decrease the punishment resulting in a more homoscedastic variance of grades.

As the artefact was used to find inconsistencies in the data sources and only one were found, it suggest the original raters of the exams was very consistent and fair respectively. However, the method of measuring for inconsistencies might be prone to Type II error (false negative) and no conclusion can be made by this that there exist no other case of inconsistency other than the one that were found. There was in total 1039 submission (data source 2 to 6) and manually validate the equality of the source code and the grades was not done as it would have taken a significantly larger amount of time than the value gained. However, it could be done for a subset of the submissions such as the ones clustered at a very low EPS values (<0.10). By utilizing the artefact as a mean to find inconsistencies *after* the grading is done, is a second possible use-case for the artefact. By running the algorithm again and extracting similar submissions which have received different grades, might reduce even more inconsistencies.

10.4 Threats to validity

The validity of the research done in this thesis is susceptible to different threats and as a disclaimer, the threats listed in this section has a fair chance to only be a subset of the total set of threats. With this said, the threats to validity in this thesis are:

1. **Sample size and geolocation**: Only 40 participants participated in the experiement were all was positioned in the same geolocation. A bigger sample size from multiple geolocations would increase the validity.

2. **Knowledge of participants**: All participants were students and no background test were conducted to validate the students knowledge of Python. The only reason for the belief that they did have knowledge of Python was by the fact that it was stated that the experiment participators needed knowledge of Python. Moreover, participants were selected only from selected groups known to have certain knowledge of Python.

3. **Experience of participants**: As all participants were students, the chance of the presence of experience with professional rating of final exams in any of the participants are minuscule. Therefore, the results from this thesis can not directly be inferred on actual professional raters without further research done.

4. **Self-administration**: As experiment 2 was a self-administrated questionnaire, the participators carried it out without supervision. The chance of invalidity is therefore high as unregulated, unobservable environments and other factors might effect the participators. These factors might be that a participator lacked focus or interest in the task, not completing it solely alone but in cooperation with someone else (might create diffusion, interaction between control and test group participants), or taking long breaks from the experiment effectively reducing the effects a sequence might impose.

5. **Incentives**: The participants were incentivized to participate by the offering of money (200,- NOK). This might have posed an effect on the collection of invalid data as some participators might have filled out the questionnaire by the sole incentive of receiving payment and not by contribution to the experiment.

6. **Obedience to instructions**: As many of the participants did not fully follow the specified 2-minute rule (by completing the questionnaire in experiment 2 much faster than what was instructed), they might also have broken other rules. Rules as: going back to previous tasks and review the given grades or conduct the experiment in an insufficient environment not fully letting them make use of the instructed grade-criteria attached. This might have lead to poor data quality as the grades given did not fully follow the necessary criteria.

7. The removal of participants in experiment 2 was done by pure intuition. This might impose that some of the removed participants were valid, or some of the participants were invalid and should have been removed as well.

8. **Statistical errors**: As of the small sample size and the various statistical methods used in this thesis, the values might be biased or prone to errors. Statistical Type I and Type II errors are also inclined to occur.

10.5 Conclusion

The goal of this research was to investigate if an optimal sequence reduce sequential effects when grading Python source codes, and develop a method to reduce sequential effects by automation. No instance of sequential effects was observed and therefore no relationship was measured between the variables suggesting that the sequence does not have any effect on the grades. However, to conclude that there is no difference is wrong as the sample sizes were small, the test results might have errors and only the presence of some effects were suggested to be non-existent and does not directly imply that all other effects are also non-existent.

Automating the generation of an optimal sequence which should reduce sequential effects was suggested to be fairly efficient by the means of the artefact described in this thesis. Using GST to measure similarity between Python source codes is significantly supported as similar to a human which further can be used to acquire a more robust method in the future. However, the optimal sequence defined in this thesis might prove to increase sequence effects more than it reduces them. Therefore, even though the artefact does generate an optimal sequence, the reduction of sequence effects might be more dependent on the definition of the optimal sequence and further research needs to be done to define a more ideal definition.

10.6 Future Work

The research and results presented in this thesis are not perfect and should be further researched. The next steps could be to conduct research into these different areas:

1. **Optimal sequence**
   (a) In this thesis, research was done to observe if sequence did have an effect on the given grades. However, not much research was done to define what an *optimal* sequence actually is. Further research can be done to identify the true aspects of an optimal sequence.
2. **Artefact Architecture**
   (a) The architecture of the artefact is only a prototype and not optimized. Optimization could be done to extract better tokens or an AST, test different distance measures, different clustering algorithms such as Hierarchical clustering, and maybe most of all find a better way than a TSP greedy heuristic to generate an optimal path.
   (b) The presented EPS-graph in Figure 9.4 and all the other EPS-graphs in appendix B demonstrate the existence of large amounts of noise. By changing different architectures and used algorithms, or finding a better way to optimizing the EPS value, noise can be reduced.
3. **Similarity and grade**
   (a) As this thesis only presents a strong suggestion between *GST* similarity and human given grades, it does not present any correlation between any other mean of similarity and grade. Research could be done to extract features where the calculated similarity have a higher correlation with the received grades over many different data sources.
4. **Sequence effects**
   (a) As this thesis did not find any sequence effects, it does not prove that the does not exist any when grading Python source codes. Further research could be done to investigate the existence of sequence effects. As well as this, further investigation could be done in the field of reducing inconsistencies.

Appendices

Appendix A

Preprocessor lookup table

Type	**Group**	**Value**
Keywords	'CONTROL_FLOW' (Flow)	for, while, break, continue, if, else, elif, return, yield, pass
	'MODULE' (MOD)	from, with, as, import
	'EXCEPTION' (EXC)	try, except, finally, raise
	'OTHER' (OTH)	del, assert
Function	'LAMBDA'(LAMB)	lambda
	'FUNCTION' (FUNC)	variables defined as callable functions
	'FUNCTION_CALL' (CALL)	.(dot), variable()
	'FUNCTION_DEFINE' (DEF)	def
	'GLOBAL' (GLOB)	global, nonlocal
	'CLASS' (CLASS)	class
Separator	'BRACKET_START'	(
	'BRACKET_END'	)
	'COLON'	:
	'COMMA'	,
	'SEMICOLON'	;
Operator	'ARITHMETIC' (ARI)	+, -, *, **, /, //, %
	'RELATIONAL' (REL)	<, >, ==, ===, !=, <=, >=
	'LOGICAL' (LOG)	‖, &, *not*, !
	'BITWISE' (BIT)	«, », , ' *and*, *or*
	'ASSIGNED' (ASS)	=, +=, -=, *= /=, %=, &=, ‖ =, <<=, >>= , ** =, // =, @ =, :=, ::=
	'MEMBERSHIP' (MEM)	in
	'IDENTITY' (IDE)	is
Literal	'BOOLEAN'	true, false
	'LIST_START'	[
	'LIST_END'	]
	'COLLECTION_START'	{
	'COLLECTION_END'	}
	'NONE'	none, null
Variable	'NUMBER'	numbers
	'STRING'	strings, list of characters, usage of "" or ''
	'VARIABLE'	assigned to user defined variable names
Unknown	'UNKNOWN'	lexemes not assigned any other type and group

Appendix B

Artefact Illustrations

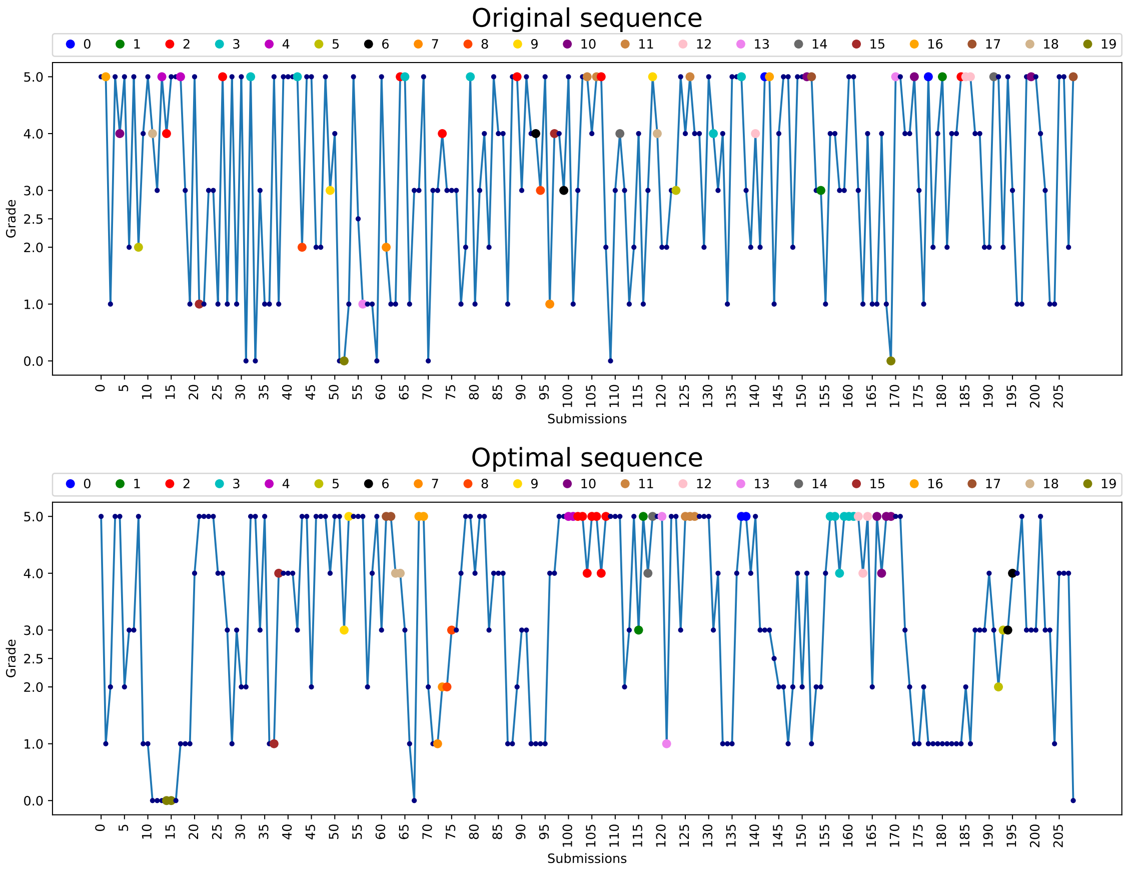


Figure B.1: Original and optimal path (before and after the artefact has run) for **datasource 3**. Each color represent one cluster, and small navy-blue dots are noise.

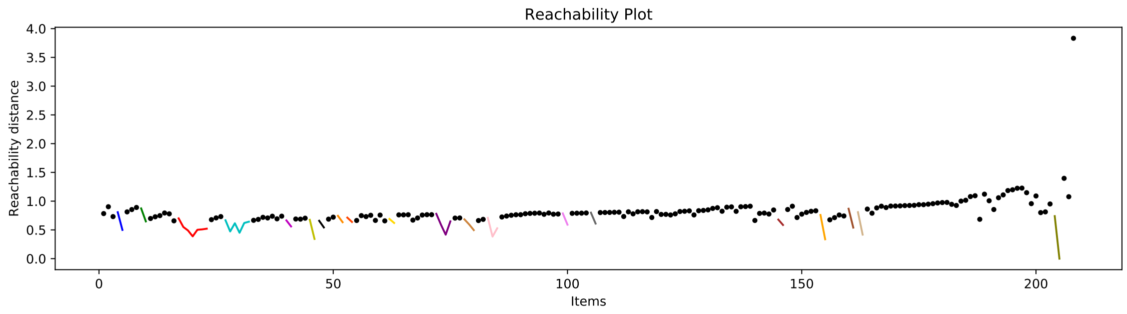


Figure B.2: Reachability plot generated after OPTICS is run for **datasource 3**

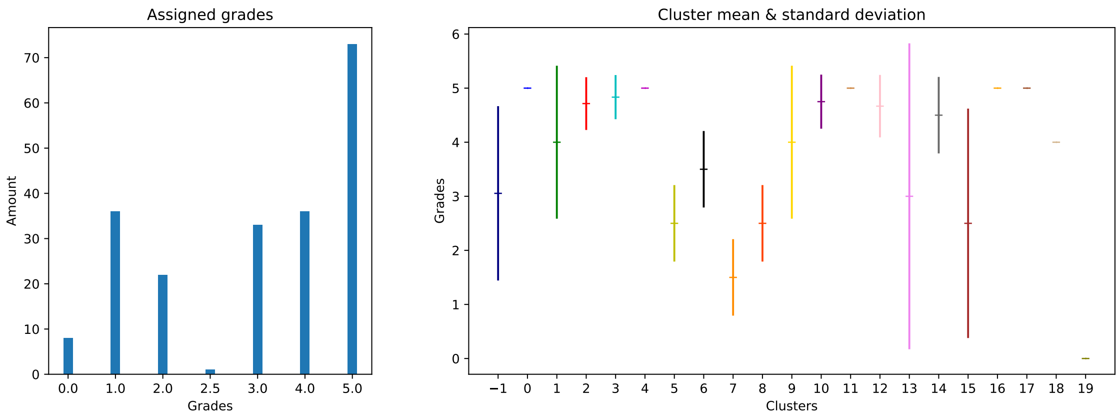


Figure B.3: Mean and standard deviation of grades pr cluster made for **datasource 3**. The colors represent the same clusters as in figure **??**. Cluster "-1" is noise.

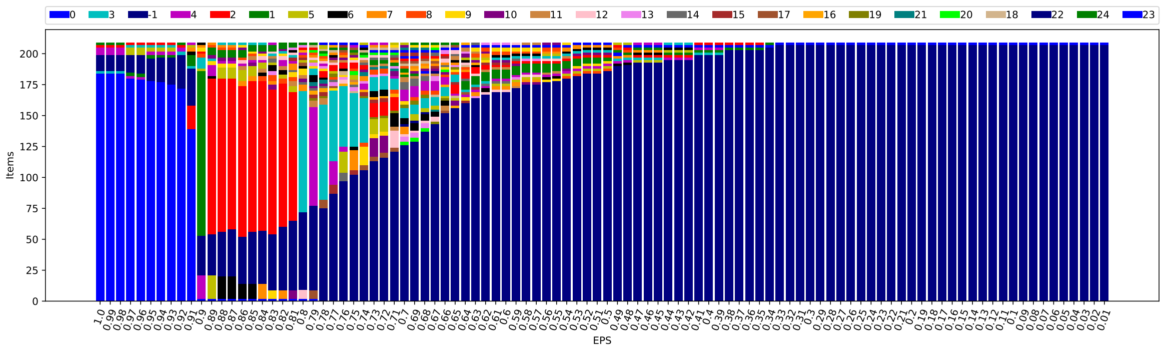


Figure B.4: Graph illustrating all the different EPS values and the clusters generated for each value between 0.01 and 1 for **datasource 3**. Each color represent one distinct cluster.

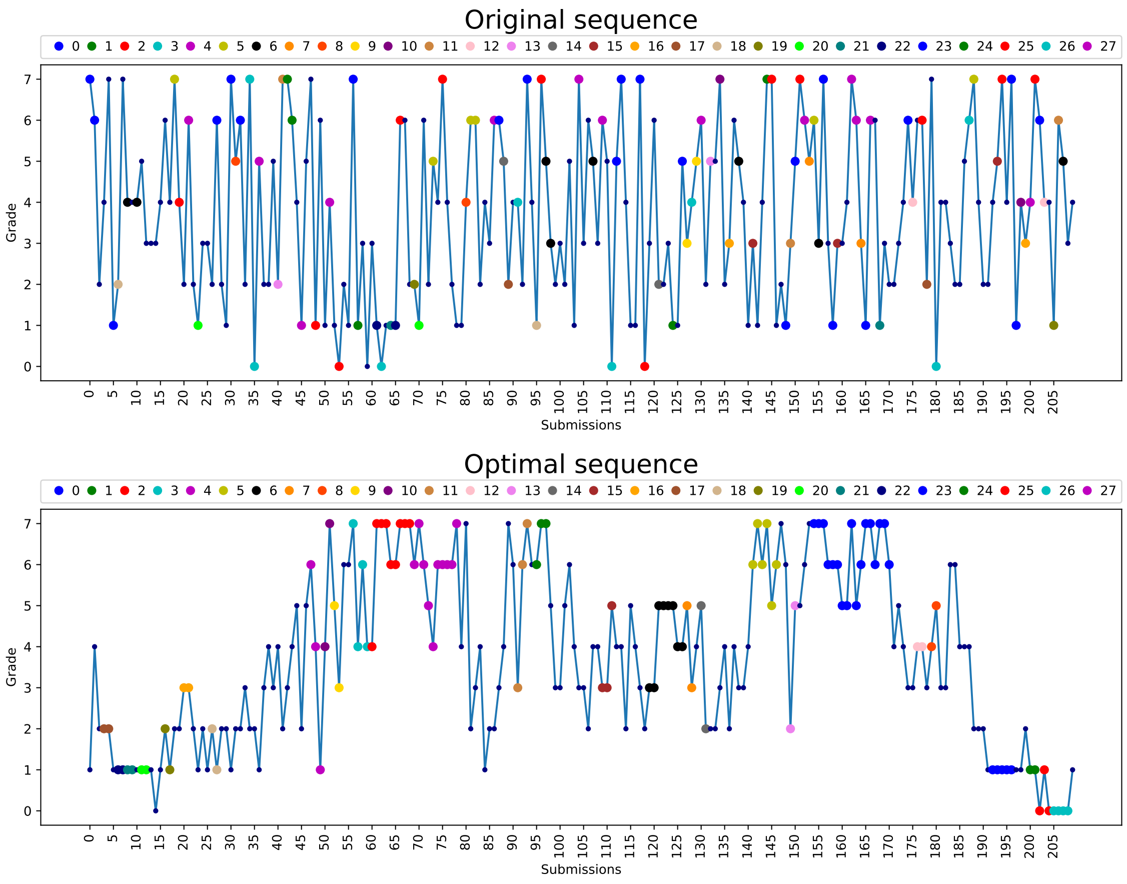


Figure B.5: Original and optimal path (before and after the artefact has run) for **datasource 4**. Each color represent one cluster, and small navy-blue dots are noise.

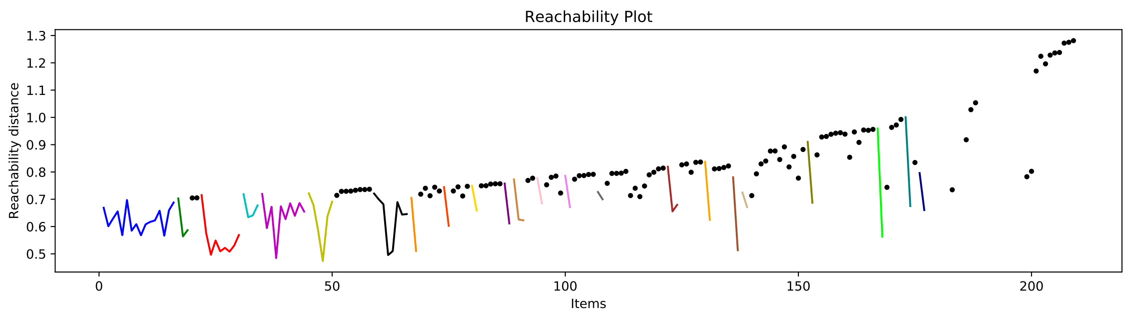


Figure B.6: Reachability plot generated after OPTICS is run for **datasource 4**

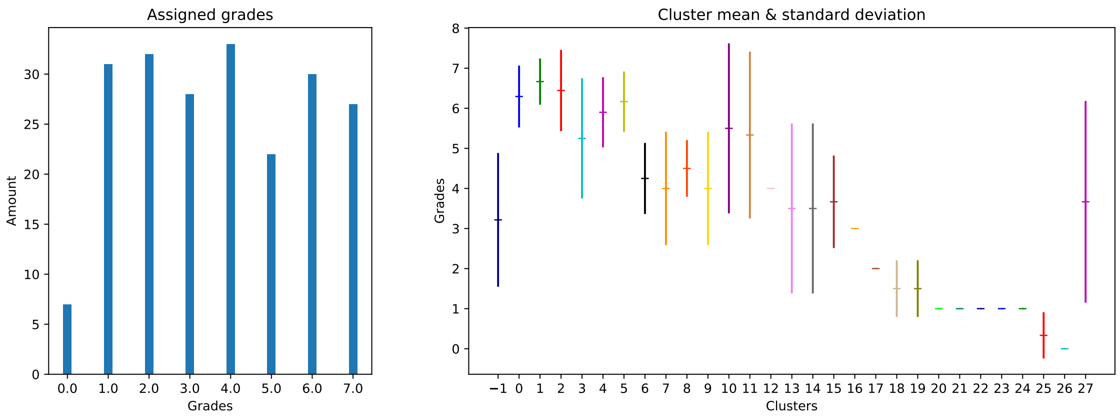


Figure B.7: Mean and standard deviation of grades pr cluster made for **datasource 4**. The colors represent the same clusters as in figure **??**. Cluster "-1" is noise.

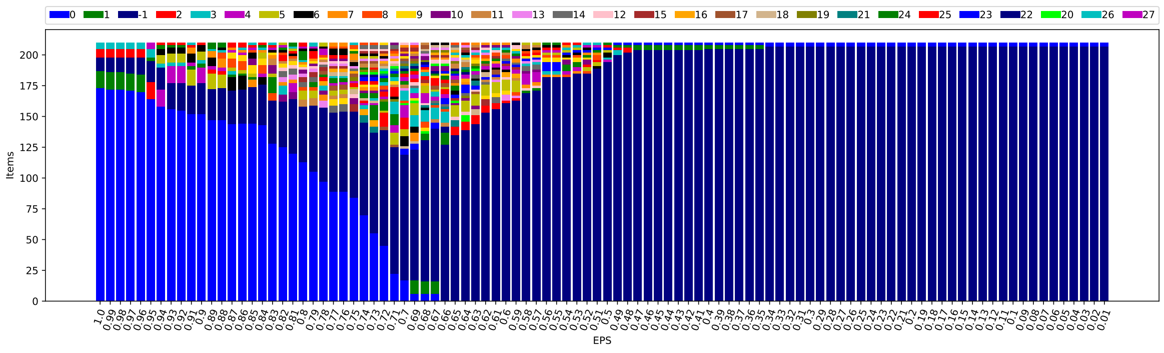
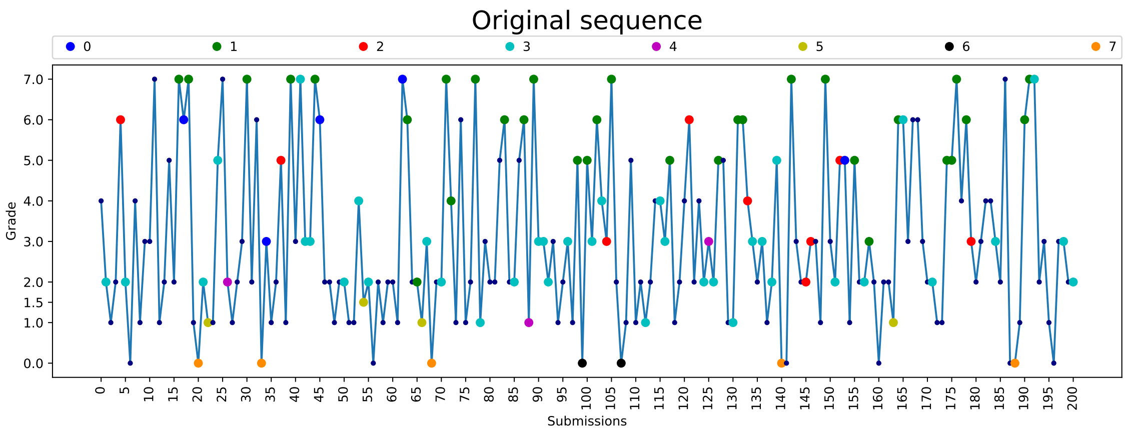


Figure B.8: Graph illustrating all the different EPS values and the clusters generated for each value between 0.01 and 1 for **datasource 3**. Each color represent one distinct cluster.



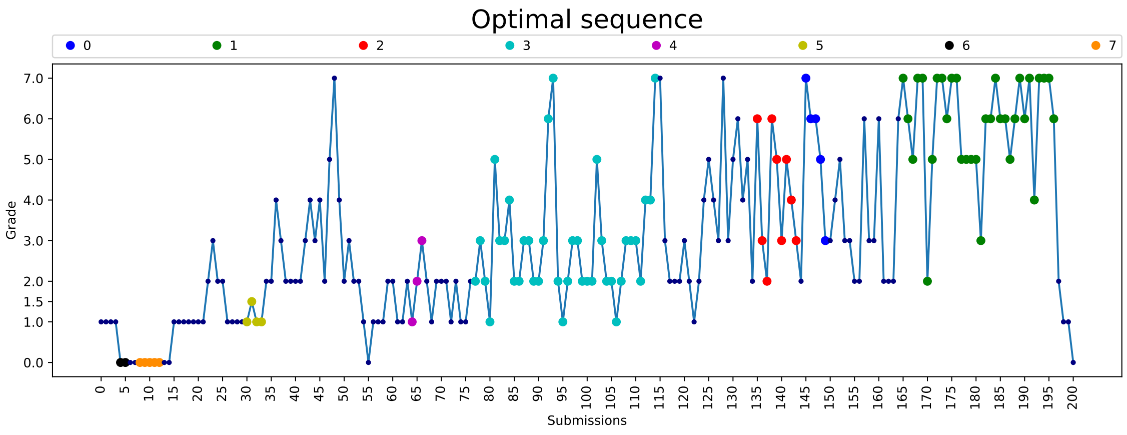


Figure B.9: Original and optimal path (before and after the artefact has run) for **data-source 5**. Each color represent one cluster, and small navy-blue dots are noise.

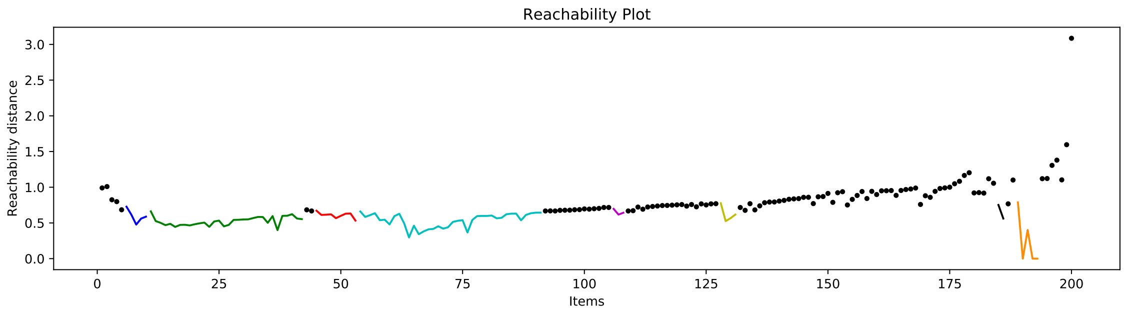


Figure B.10: Reachability plot generated after OPTICS is run for **datasource 5**

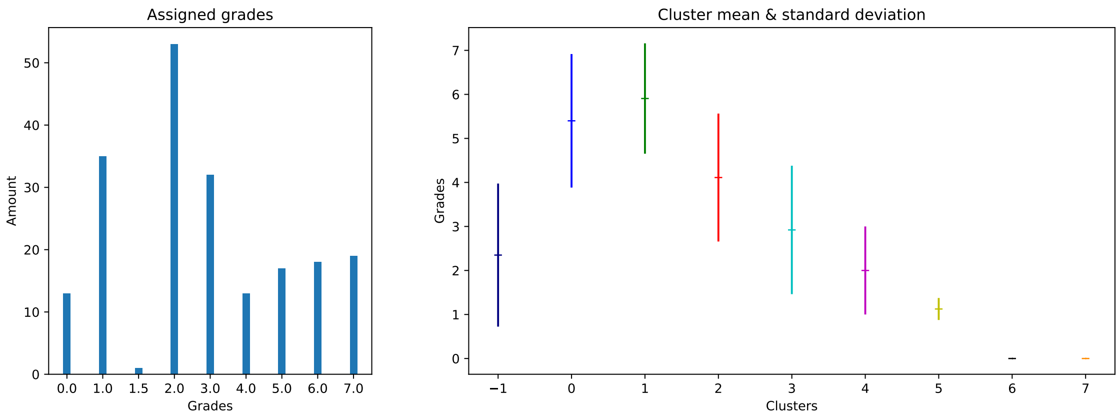


Figure B.11: Mean and standard deviation of grades pr cluster made for **datasource 5**. The colors represent the same clusters as in figure **??**. Cluster "-1" is noise.

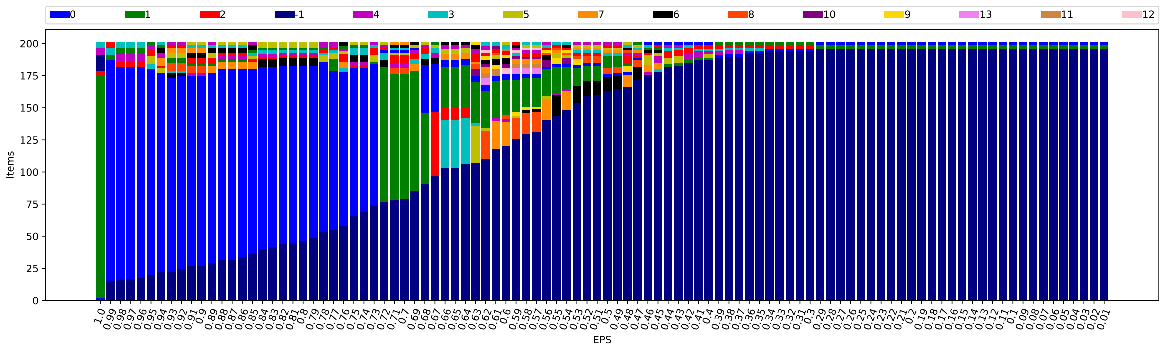
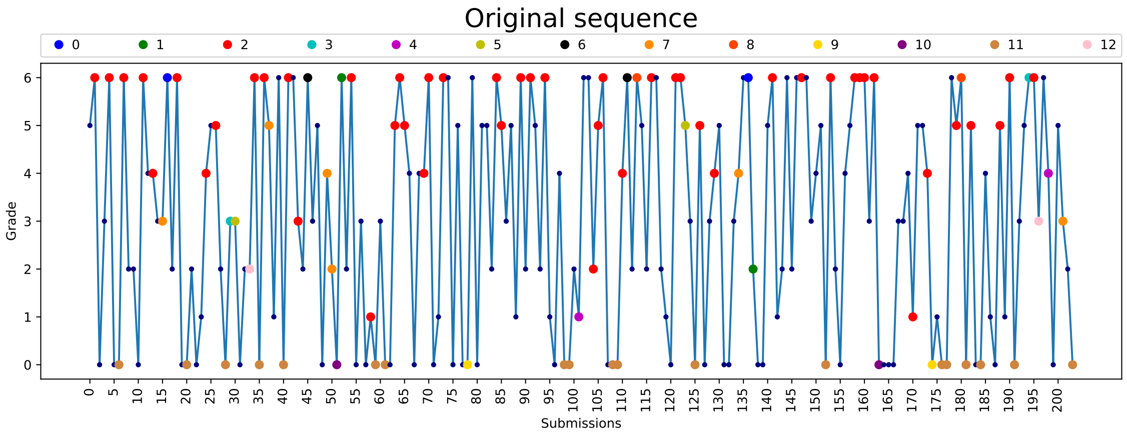


Figure B.12: Graph illustrating all the different EPS values and the clusters generated for each value between 0.01 and 1 for **datasource 5**. Each color represent one distinct cluster.



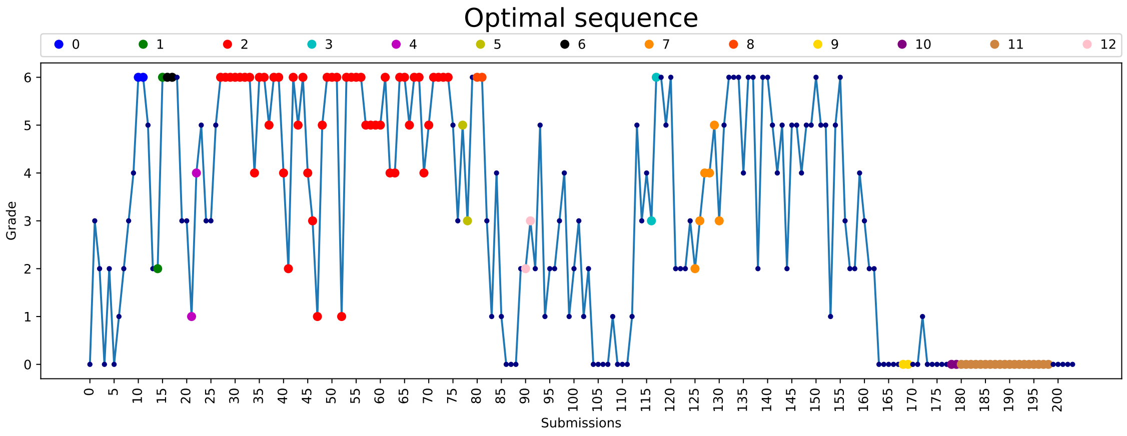


Figure B.13: Original and optimal path (before and after the artefact has run) for **datasource 6**. Each color represent one cluster, and small navy-blue dots are noise.

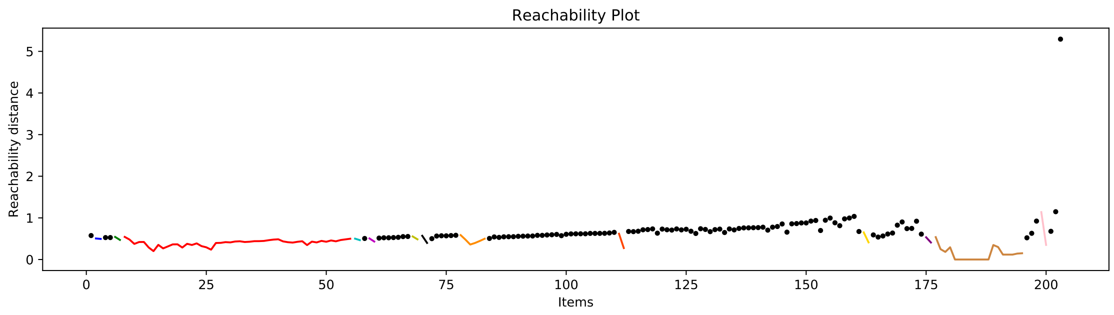


Figure B.14: Reachability plot generated after OPTICS is run for **datasource 6**

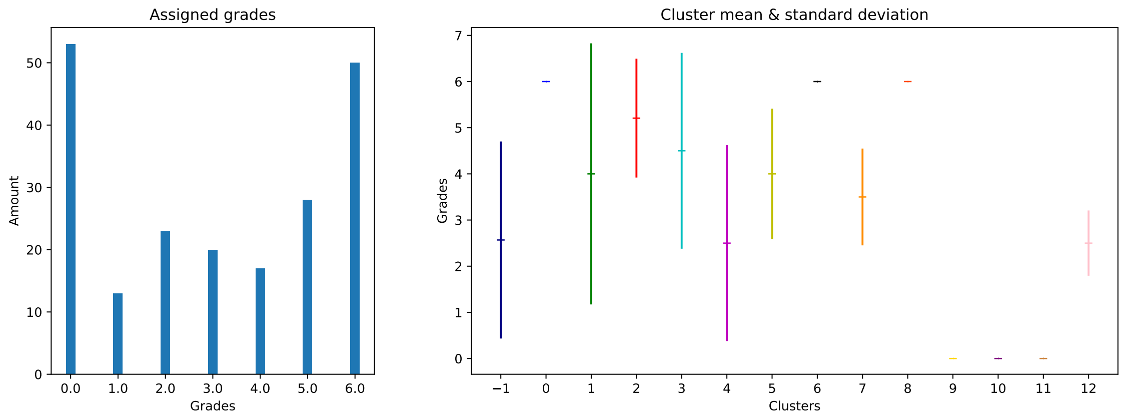


Figure B.15: Mean and standard deviation of grades pr cluster made for **datasource 6**. The colors represent the same clusters as in figure **??**. Cluster "-1" is noise.

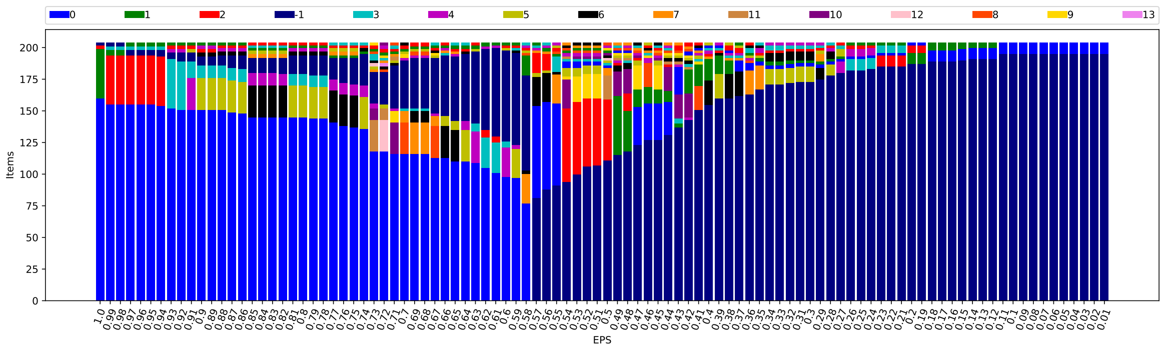
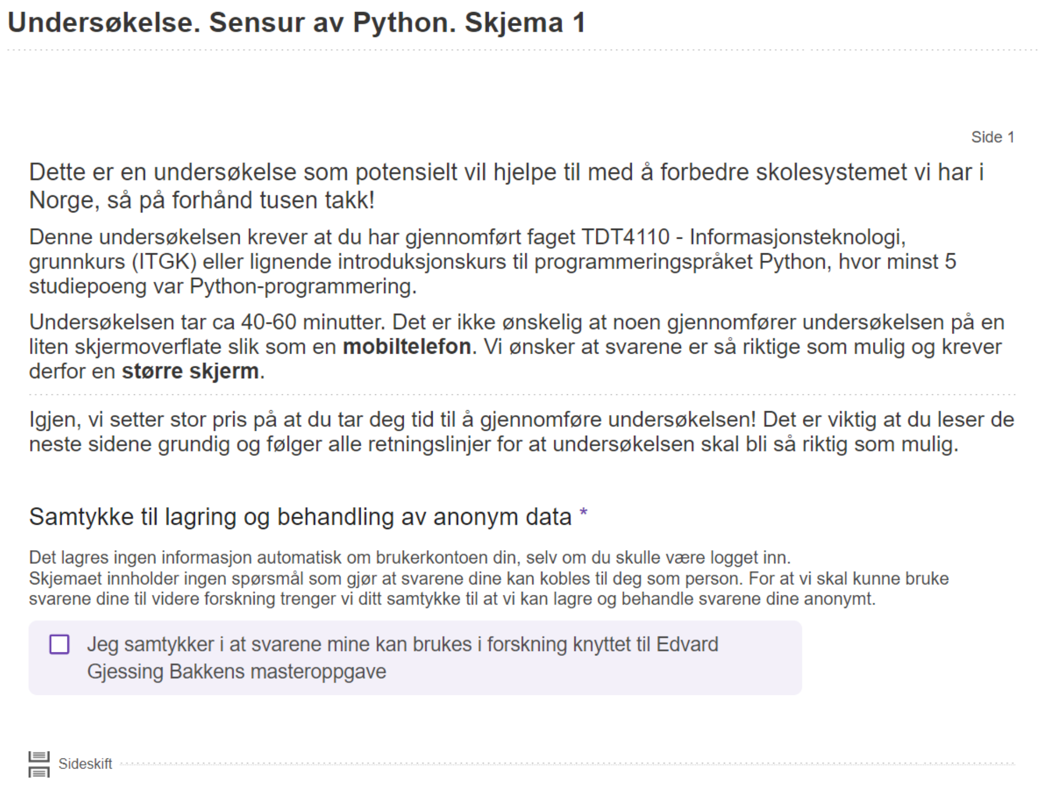


Figure B.16: Graph illustrating all the different EPS values and the clusters generated for each value between 0.01 and 1 for **datasource 6**. Each color represent one distinct cluster.

Appendix C

Nettskjema



Undersøkelse. Sensur av Python. Skjema 1

Side 1

Dette er en undersøkelse som potensielt vil hjelpe til med å forbedre skolesystemet vi har i Norge, så på forhånd tusen takk!

Denne undersøkelsen krever at du har gjennomført faget TDT4110 - Informasjonsteknologi, grunnkurs (ITGK) eller lignende introduksjonskurs til programmeringspråket Python, hvor minst 5 studiepoeng var Python-programmering.

Undersøkelsen tar ca 40-60 minutter. Det er ikke ønskelig at noen gjennomfører undersøkelsen på en liten skjermoverflate slik som en **mobiltelefon**. Vi ønsker at svarene er så riktige som mulig og krever derfor en **større skjerm**.

Igjen, vi setter stor pris på at du tar deg tid til å gjennomføre undersøkelsen! Det er viktig at du leser de neste sidene grundig og følger alle retningslinjer for at undersøkelsen skal bli så riktig som mulig.

Samtykke til lagring og behandling av anonym data *

Det lagres ingen informasjon automatisk om brukerkontoen din, selv om du skulle være logget inn.
Skjemaet innholder ingen spørsmål som gjør at svarene dine kan kobles til deg som person. For at vi skal kunne bruke svarene dine til videre forskning trenger vi ditt samtykke til at vi kan lagre og behandle svarene dine anonymt.

☐ Jeg samtykker i at svarene mine kan brukes i forskning knyttet til Edvard Gjessing Bakkens masteroppgave

Sideskift

Side 2

I denne undersøkelsen skal du teste deg selv som sensor med å gi poeng i området 0 (stryk) opp til 5 (perfekt) på 20 Python programmeringssvar.

Vi ønsker at du ikke bruker mer eller mindre en **2 minutter pr. svar**. Derfor ønsker vi at du bruker en klokke (muligens mobilen din med stoppeklokke som teller ved siden av deg mens du gjennomfører undersøkelsen). Dette er for at du ikke skal stresse deg igjennom, men ta deg litt tid til å sette deg inn i oppgaven og gi ett så godt svar som mulig. Hvis du bruker mer enn 2 min, må du bare skrive det tallet som føles mest riktig. Det er ikke liv og død på spill om du ikke får det helt nøyaktig, men vi ønsker at det skal være så riktig som mulig.

Så ta fram en stoppeklokke og gjør deg klar!

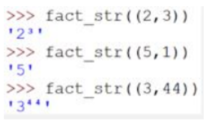
Sideskift

Side 3

Nedenfor er oppgaveteksten. Sørg for at du forstår hva som skal gjøres og tenk litt på hvordan du ville ha løst oppgaven.

Oppgavetekst:

The function **fact_str(tup)** has a parameter the tuple **tup**, which can be assumed to always consist of two integers > 0. The first integer is the mantissa and the second the exponent, and the function shall return this as a **string**. Examples of usage:



```
>>> fact_str((2,3))
'2³'
>>> fact_str((5,1))
'5'
>>> fact_str((3,44))
'3⁴⁴'
```

As the middle example shows, the string shall exclude the exponent if it is 1.
Please write the code for the **fact_str(tup)** so that it works as specified above.
You may use the function **num2exp()** from the previous question in your code.
You can assume that **num2exp()** works even if you did not complete it.

Løsningsforslag:

```
def fact_str(tup):
    result = str(tup[0])
    if tup[1] > 1:
        result += num2exp(tup[1])
return result
```

- Løsningsforslaget omgjør først tup[0](mantissen) til en streng. Dette er veldig viktig fordi funksjonen skal returnere en streng, ikke en int.
- Den har logikk for å kontrollere om tup[1](eksponenten) er større enn 1 og returnerer mantissen med eller uten eksponenten.

Funksjonen **num2exp()** tar ett tall(int) som parameter og **returnerer en streng** av tallet i opphøyd posisjon. Det er ikke viktig å vite noe mer om hvordan den fungerer unntatt at "1" != num2exp(1). Altså at input ikke er lik output.

Eksempel:

num2exp(5) returnerer '⁵'
num2exp(12) returnerer '¹²'

Side 4

Når man skriver kode gjør man ofte to typiske feil. Feil i **syntaks** og feil i **logikk**. Sørg for at du forstår hva de er og forskjellen på de.

- **Syntaksfeil** (Hindrer kompilering av koden)
  - Brudd på språkets syntaks
    - f.eks. feil eller manglende separatorer (som kolon, komma, parentes, fnutter rundt strenger)
  - Feilstaving av ord
    - f.eks. skrevet hwile i stedet for while.
- **Logiske feil** (Koden oppnår ikke ønsket resultat eller vil krasje når den blir kjørt)
  - Koden oppnår ikke de satte målene eller har dårlig konstruert funksjonalitet som krasjer koden. Det kan være feil slik som:
    - Skrevet >= istedenfor <=
    - Indekserer en liste med en streng, ikke int; (list["1"] vil krasje).
    - En bit av koden blir aldri kjørt når den er nødvendig.
    - Skrevet operatoren + istedenfor *
    - Mangler nødvendig if/else eller for/while, etc

Nedenfor ser du retningslinjene du skal forholde deg til for å gi riktig poengskår.

(5 poeng): Perfekt svar, eller nær perfekt (små syntaksfeil, f.eks. glemt kolon)

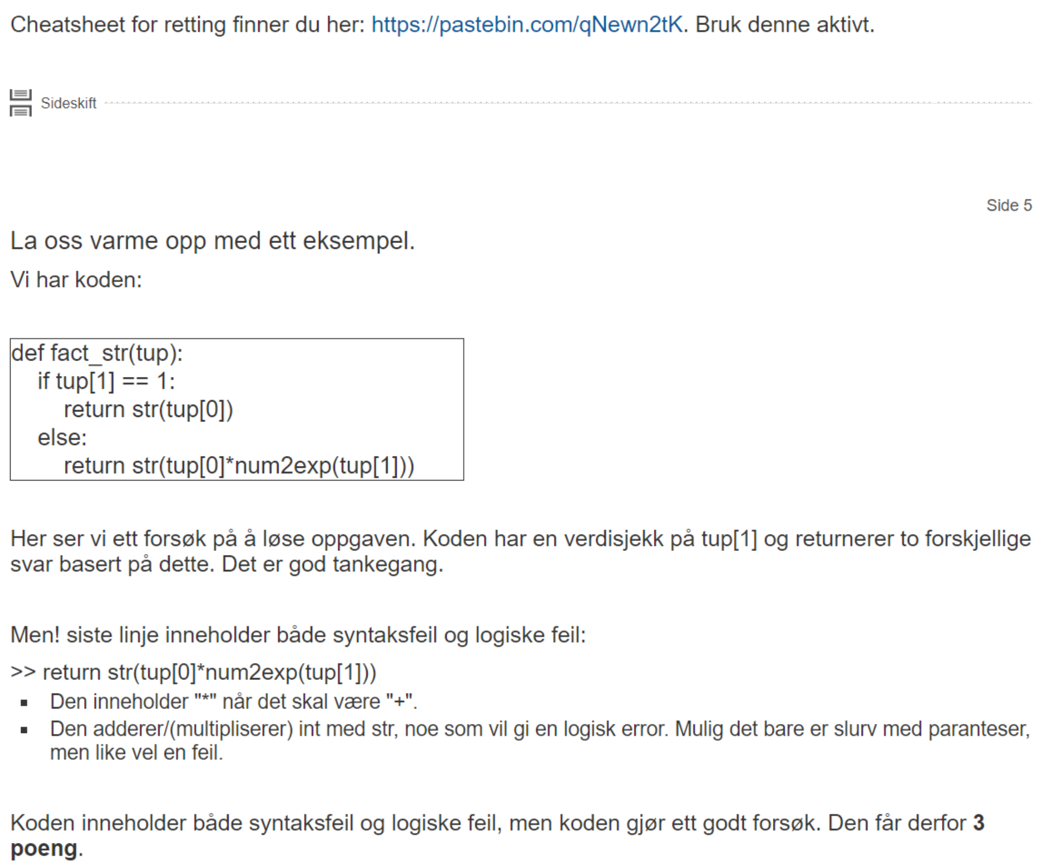
(4 poeng): Har mye av løsningen korrekt, men med flere påtagelige **syntaksfeil**

(3 poeng): Har mye av løsningen korrekt, men med flere påtagelige **logiske feil**

(2 poeng): Masse logiske og syntaksfeil. Delvis svar på oppgave

(1 poeng): Veldig tynn kode og/eller masse logiske og syntaksfeil.

(0 poeng): Blankt svar, nær blankt, kun repetering av oppgavetekst, eller totalt irrelevant kode.



Cheatsheet for retting finner du her: https://pastebin.com/qNewn2tK. Bruk denne aktivt.

Sideskift

Side 5

La oss varme opp med ett eksempel.

Vi har koden:

```
def fact_str(tup):
    if tup[1] == 1:
        return str(tup[0])
    else:
        return str(tup[0]*num2exp(tup[1]))
```

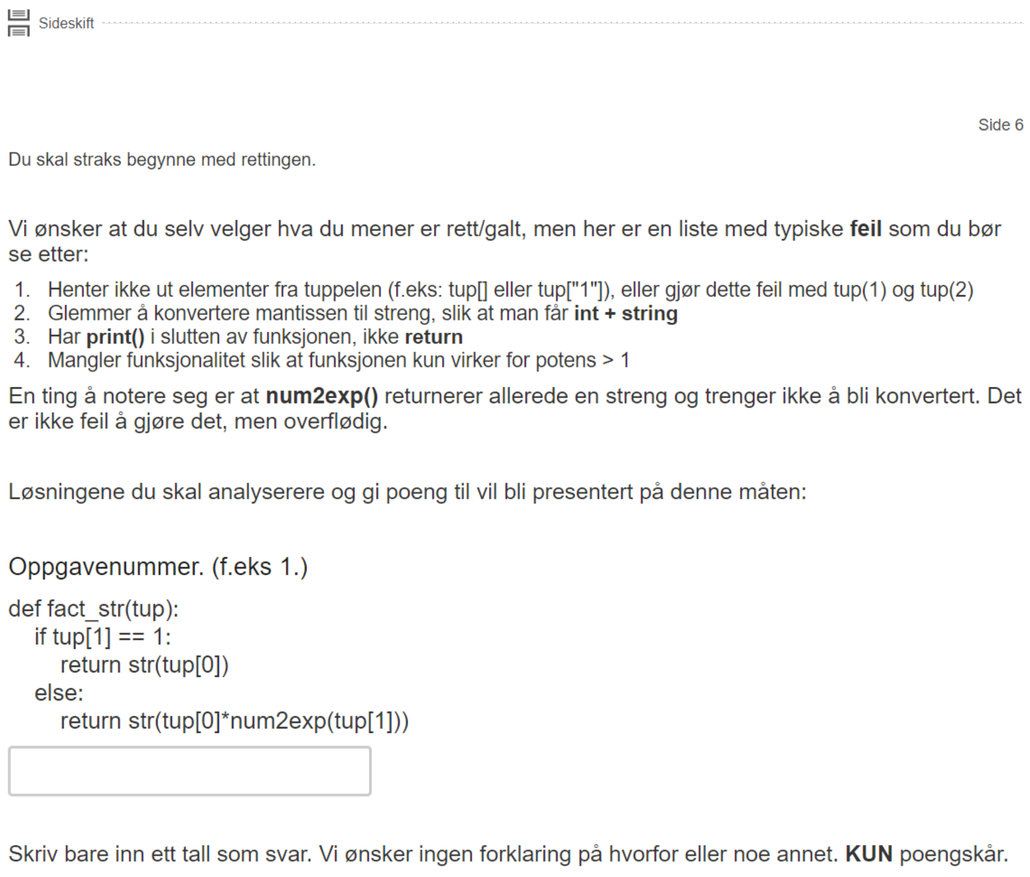
Her ser vi ett forsøk på å løse oppgaven. Koden har en verdisjekk på tup[1] og returnerer to forskjellige svar basert på dette. Det er god tankegang.

Men! siste linje inneholder både syntaksfeil og logiske feil:

>> return str(tup[0]*num2exp(tup[1]))

- Den inneholder "*" når det skal være "+".
- Den adderer/(multipliserer) int med str, noe som vil gi en logisk error. Mulig det bare er slurv med paranteser, men like vel en feil.

Koden inneholder både syntaksfeil og logiske feil, men koden gjør ett godt forsøk. Den får derfor **3 poeng**.



Sideskift

Side 6

Du skal straks begynne med rettingen.

Vi ønsker at du selv velger hva du mener er rett/galt, men her er en liste med typiske **feil** som du bør se etter:

1. Henter ikke ut elementer fra tuppelen (f.eks: tup[] eller tup["1"]), eller gjør dette feil med tup(1) og tup(2)
2. Glemmer å konvertere mantissen til streng, slik at man får **int + string**
3. Har **print()** i slutten av funksjonen, ikke **return**
4. Mangler funksjonalitet slik at funksjonen kun virker for potens > 1

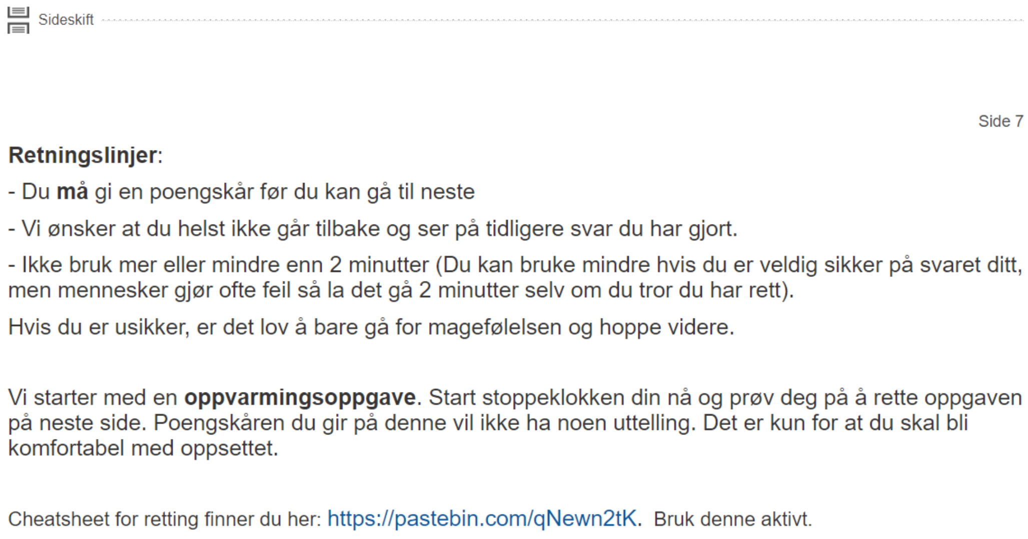
En ting å notere seg er at **num2exp()** returnerer allerede en streng og trenger ikke å bli konvertert. Det er ikke feil å gjøre det, men overflødig.

Løsningene du skal analyserere og gi poeng til vil bli presentert på denne måten:

Oppgavenummer. (f.eks 1.)

```
def fact_str(tup):
    if tup[1] == 1:
        return str(tup[0])
    else:
        return str(tup[0]*num2exp(tup[1]))
```

Skriv bare inn ett tall som svar. Vi ønsker ingen forklaring på hvorfor eller noe annet. **KUN** poengskår.



Sideskift

Side 7

Retningslinjer:

- Du **må** gi en poengskår før du kan gå til neste

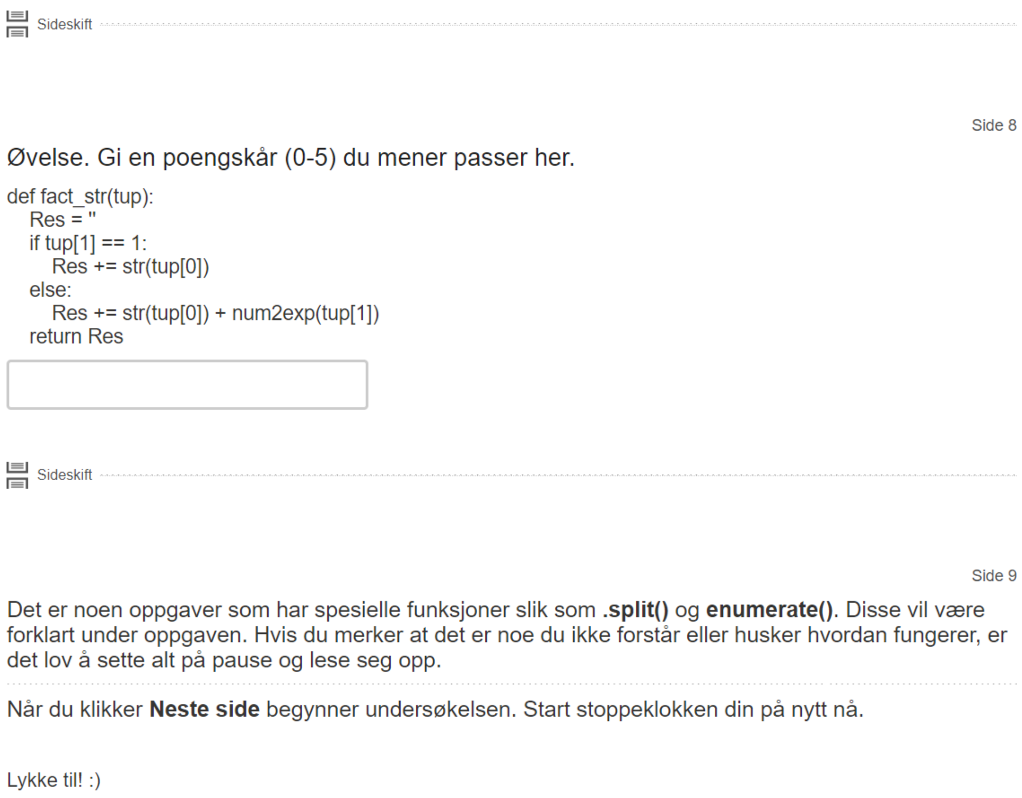
- Vi ønsker at du helst ikke går tilbake og ser på tidligere svar du har gjort.

- Ikke bruk mer eller mindre enn 2 minutter (Du kan bruke mindre hvis du er veldig sikker på svaret ditt, men mennesker gjør ofte feil så la det gå 2 minutter selv om du tror du har rett).

Hvis du er usikker, er det lov å bare gå for magefølelsen og hoppe videre.

Vi starter med en **oppvarmingsoppgave**. Start stoppeklokken din nå og prøv deg på å rette oppgaven på neste side. Poengskåren du gir på denne vil ikke ha noen uttelling. Det er kun for at du skal bli komfortabel med oppsettet.

Cheatsheet for retting finner du her: https://pastebin.com/qNewn2tK. Bruk denne aktivt.



Sideskift

Side 8

Øvelse. Gi en poengskår (0-5) du mener passer her.

```
def fact_str(tup):
    Res = ''
    if tup[1] == 1:
        Res += str(tup[0])
    else:
        Res += str(tup[0]) + num2exp(tup[1])
    return Res
```

Sideskift

Side 9

Det er noen oppgaver som har spesielle funksjoner slik som **.split()** og **enumerate()**. Disse vil være forklart under oppgaven. Hvis du merker at det er noe du ikke forstår eller husker hvordan fungerer, er det lov å sette alt på pause og lese seg opp.

Når du klikker **Neste side** begynner undersøkelsen. Start stoppeklokken din på nytt nå.

Lykke til! :)

Sideskift

Side 10

1. *

```
def fact_str(tup):
    answer = str(tup[0])
    if tup[1] != 1:
        answer += num2exp(tup[1])
    return answer
```

Sideskift

Side 11

2 *

```
def fact_str(tup):
    result = ''
    result+=str(tup[0])
    expo = num2exp(tup[1])
    result+=expo
    return result
```

Bibliography

, . 6. Kvalitetssikring av eksamen og sensur. URL: https://www.udir.no/tall-og-forskning/finn-forskning/rapporter/vurderinger-og-forelopige-anbefalinger-fra-eksamensgruppa/6.kvalitetssikring-av-eksamen-og-sensur/.

, . Cosine Similarity - an overview | ScienceDirect Topics. URL: https://www.sciencedirect.com/topics/computer-science/cosine-similarity.

, . Digital eksamen | Unit. URL: https://www.unit.no/en/node/496.

, . DIGITAL VURDERING OG EKSAMEN. Technical Report.

, . Shared marks - Knowledge Base - Inspera. URL: https://inspera.atlassian.net/wiki/spaces/KB/pages/87359579/Shared+marks.

, . SIGKDD News : 2014 SIGKDD Test of Time Award. URL: https://www.kdd.org/News/view/2014-sigkdd-test-of-time-award.

, . What is Statistical Features | IGI Global. URL: https://www.igi-global.com/dictionary/identification-of-wireless-devices-from-their-physical-layer-radio-f
60630.

Agarwala, R., Applegate, D.L., Maglott, D., Schuler, G.D., Schäffer, A.A., 2000. A fast and scalable radiation hybrid map construction and integration strategy. Genome Research 10, 350–364. doi:10.1101/gr.10.3.350.

Albluwi, I., 2018. A Closer Look at the Differences between Graders in Introductory Computer Science Exams. IEEE Transactions on Education 61, 253–260. doi:10.1109/TE.2018.2805706.

Alon, U., Zilberstein, M., Levy, O., Yahav, E., 2018. A general path-based representation for predicting program properties, in: Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), Association for Computing Machinery, New York, New York, USA. pp. 404–419. URL: http://dl.acm.org/citation.cfm?doid=3192366.3192412, doi:10.1145/3192366.3192412.

Anderson, R.C., Biddle, W.B., 1975. On asking people questions about what they are reading. Psychology of Learning and Motivation - Advances in Research and Theory 9, 89–132. doi:10.1016/S0079-7421(08)60269-8.

Ankerst, M., Ankerst, M., Breunig, M.M., Kriegel, H.p., Sander, J., 1999. OPTICS: Ordering Points To Identify the Clustering Structure , 49–60URL: https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.129.6542.

Aslett, H.J., 2006. Reducing variability, increasing reliability: exploring the psychology of intra-and inter-rater reliability. Investigations in university teaching and learning 4.

Attali, Y., Lewis, W., Steier, M., 2013. Scoring with the computer: Alternative procedures for improving the reliability of holistic essay scoring. Language Testing 30, 125–141. URL: http://journals.sagepub.com/doi/10.1177/0265532212452396, doi:10.1177/0265532212452396.

Babenko, M.A., Starikovskaya, T.A., 2011. Computing the longest common substring with one mismatch. Problems of Information Transmission 47, 28–33. doi:10.1134/S0032946011010030.

Barbosa, A.d.A., Costa, E.d.B., Brito, P.H., 2018. Adaptive clustering of codes for assessment in introductory programming courses, in: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Springer Verlag. pp. 13–22. doi:10.1007/978-3-319-91464-0{_}2.

Basu, S., Jacobs, C., Vanderwende, L., 2013. Powergrading: a Clustering Approach to Amplify Human Effort for Short Answer Grading. Transactions of the Association for Computational Linguistics 1, 391–402. doi:10.1162/tacl{_}a{_}00236.

Bhargava, S., Fisman, R., 2014. Contrast effects in sequential decisions: Evidence from speed dating. Review of Economics and Statistics 96, 444–457. doi:10.1162/REST{_}a{_}00416.

Brank, J., Mladenić, D., Grobelnik, M., Liu, H., Mladenić, D., Flach, P.A., Garriga, G.C., Toivonen, H., Toivonen, H., 2011. Feature Selection, in: Encyclopedia of Machine Learning. Springer US, Boston, MA, pp. 402–406. URL: http://link.springer.com/10.1007/978-0-387-30164-8_306, doi:10.1007/978-0-387-30164-8{_}306.

Brusch, J., Trapnes, N., Sindre, G., . Digitalization of Exams A study of how digitalization can improve the examination processes at NTNU. Technical Report.

Chire, 2011a. File:DBSCAN-Illustration.svg - Wikimedia Commons. URL: https://commons.wikimedia.org/wiki/File:DBSCAN-Illustration.svg.

Chire, 2011b. File:OPTICS.svg - Wikimedia Commons. URL: https://commons.wikimedia.org/wiki/File:OPTICS.svg.

Derrick, B., Candidate, P., Ruck, A., Toher, D., White, P., . TESTS FOR EQUALITY OF VARIANCES BETWEEN TWO SAMPLES WHICH CONTAIN BOTH PAIRED OBSERVATIONS AND INDEPENDENT OBSERVATIONS. Technical Report.

Durić, Z., Gašević, D., 2013. A source code similarity system for plagiarism detection. Computer Journal 56, 70–86. doi:10.1093/comjnl/bxs018.

Ester, M., Kriegel, H.P., Sander, J., Xu, X., 1996. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. Technical Report. URL: www.aaai.org.

Fitzgerald, S., Hanks, B., Lister, R., McCauley, R., Murphy, L., 2013. What are we thinking when we grade programs?, in: Proceeding of the 44th ACM technical symposium on Computer science education - SIGCSE '13, ACM Press, New York, New York, USA. p. 471. URL: http://dl.acm.org/citation.cfm?doid=2445196.2445339, doi:10.1145/2445196.2445339.

Haldar, R., Mukhopadhyay, D., 2011. Levenshtein Distance Technique in Dictionary Lookup Methods: An Improved Approach URL: http://arxiv.org/abs/1101.1232.

Haselton, M.G., Nettle, D., Murray, D.R., 2015. The Evolution of Cognitive Bias, in: The Handbook of Evolutionary Psychology. John Wiley & Sons, Inc., Hoboken, NJ, USA, pp. 1–20. URL: http://doi.wiley.com/10.1002/9781119125563.evpsych241, doi:10.1002/9781119125563.evpsych241.

Hatzipanagos, S., Gregson, J., 2015. The Role of Open Access and Open Educational Resources : A Distance Learning Perspective University of London Centre for Distance Education , Visiting Fellow , UK. Electronic Journal of e-Learning 13, 97–105.

Hindle, A., Barr, E.T., Su, Z., Gabel, M., Devanbu, P., . On the Naturalness of Software. URL: http://en.wikiquote.

Klein, J., El, L.P., 2003. Impairment of teacher efficiency during extended sessions of test correction. European Journal of Teacher Education 26, 379–392. doi:10.1080/0261976032000128201.

Kramer, R.S.S., 2017. Sequential effects in Olympic synchronized diving scores. Royal Society Open Science 4, 160812. URL: https://royalsocietypublishing.org/doi/10.1098/rsos.160812, doi:10.1098/rsos.160812.

Kufman, L., Rousseeuw, P.J., 1987. Clustering by means of medoids .

Landauer, T.K., Foltz, P.W., Laham, D., 1998. An introduction to latent semantic analysis. Discourse Processes 25, 259–284. doi:10.1080/01638539809545028.

Lewis, D.D., 1992. Feature selection and feature extraction for text categorization, Association for Computational Linguistics (ACL). p. 212. doi:10.3115/1075527.1075574.

Li, Q., Hu, J., Ding, J., Zheng, G., 2014. Fisher's method of combining dependent statistics using generalizations of the gamma distribution with applications to genetic pleiotropic associations. Biostatistics (Oxford, England) URL: https://doi.org/10.1093/biostatistics/kxt045.

Lockton, D., 2012. Cognitive Biases, Heuristics and Decision-Making in Design for Behaviour Change. SSRN Electronic Journal doi:10.2139/ssrn.2124557.

McGraw, K.O., Wong, S.P., 1992. A Common Language Effect Size Statistic. Psychological Bulletin 111, 361–365. doi:10.1037/0033-2909.111.2.361.

McKnight, P.E., Najab, J., 2010. Mann-Whitney U Test, in: The Corsini Encyclopedia of Psychology. John Wiley & Sons, Inc., Hoboken, NJ, USA, pp. 1–1. URL: http://doi.wiley.com/10.1002/9780470479216.corpsy0524, doi:10.1002/9780470479216.corpsy0524.

Midtbø, T., Rossow, A., Sagbakken, B., 2018. Måling av sensorreliabilitet ved vurdering av norskprøve i skriftlig framstilling. Acta Didactica Norge 12, 12. doi:10.5617/adno.6358.

de Moira, A.P., Massey, C., Baird, J.A., Morrissy, M., 2002. Marking Consistency over Time. Research in Education 67, 79–87. doi:10.7227/rie.67.8.

Muckler, F.A., Seven, S.A., 1992. Selecting Performance Measures: "Objective" versus "Subjective" Measurement. Human Factors: The Journal of the Human Factors and Ergonomics Society 34, 441–455. URL: http://journals.sagepub.com/doi/10.1177/001872089203400406, doi:10.1177/001872089203400406.

Nilsson, C., 2003. Heuristics for the Traveling Salesman Problem. Technical Report.

Oates, B.J., 2005. Researching information systems and computing. Sage .

Page, L., Page, K., 2010. Last shall be first: A field study of biases in sequential performance evaluation on the Idol series. Journal of Economic Behavior and Organization 73, 186–198. doi:10.1016/j.jebo.2009.08.012.

Pang-Ning, T., Steinbach, M., Karpatne, A., Kumar, V., 2005. Introduction to Data Mining. URL: https://www-users.cs.umn.edu/~kumar001/dmbook/index.php.

Prechelt, L., Malpohl, G., Philippsen, M., . Finding Plagiarisms among a Set of Programs with JPlag. Technical Report. URL: http://www.jplag.de.

Ragkhitwetsagul, C., Krinke, J., Clark, D., 2018. A comparison of code similarity analysers. Empirical Software Engineering 23, 2464–2519. doi:10.1007/s10664-017-9564-7.

Raikes, N., Fidler, J., Gill, T., 2009. Must examiners meet in order to standardise their marking? An experiment with new and experienced examiners of GCE AS Psychology. Technical Report.

Ramesh, D., Liu, A.Z., Echeverria, A.J., Song, J.Y., Waytowich, N.R., Lasecki, W.S., . Yesterday's Reward is Today's Punishment: Contrast Effects in Human Feedback to Reinforcement Learning Agents Human-Agent Interaction, Contrast Effects, Reinforcement Learn-ing ACM Reference Format. Technical Report. URL: www.ifaamas.org.

Richard Karp Michael, b.M., . Efficient randomized pattern-matching algorithms. Technical Report.

Rosenkrantz, D.J., Stearns, R.E., Lewis, P.M., 2008. Approximate algorithms for the traveling salesperson problem, Institute of Electrical and Electronics Engineers (IEEE). pp. 33–42. doi:10.1109/swat.1974.4.

Rye, J.F., 2014. Konsistente karakterer? Uniped 37, 63–77. doi:10.3402/uniped.v37.22654.

Alves dos Santos, J.C., Favero, E.L., 2015. Practical use of a latent semantic analysis (LSA) model for automatic evaluation of written answers. Journal of the Brazilian Computer Society 21, 1–8. URL: https://journal-bcs.springeropen.com/articles/10.1186/s13173-015-0039-7, doi:10.1186/s13173-015-0039-7.

Shah, A.K., Oppenheimer, D.M., 2008. Heuristics Made Easy: An Effort-Reduction Framework. Psychological Bulletin 134, 207–222. doi:10.1037/0033-2909.134.2.207.

Sidorov, G., Velasquez, F., Stamatatos, E., Gelbukh, A., Chanona-Hernández, L., 2014. Syntactic N-grams as machine learning features for natural language processing. Expert Systems with Applications 41, 853–860. doi:10.1016/j.eswa.2013.08.015.

Skedsmo, G., Huber, S.G., 2018. Reliability, validity and fairness—key issues in assessing the quality of teaching, instructional leadership and school practice. doi:10.1007/s11092-018-9290-8.

Spear, M., 1997. The influence of contrast effects upon teachers' marks. Educational Research 39, 229–233. doi:10.1080/0013188970390209.

Srikant, S., Aggarwal, V., . A System to Grade Computer Programming Skills using Machine Learning URL: http://dx.doi.org/10.1145/2623330.2623377, doi:10.1145/2623330.2623377.

Srikant, S., Aggarwal, V., 2014. A system to grade computer programming skills using machine learning, in: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Association for Computing Machinery. pp. 1887–1896. doi:10.1145/2623330.2623377.

Srull, T.K., 1984. The Effects of Subjective Affective States on Memory and Judgment. ACR North American Advances NA-11.

Tayi, G.K., Ballou, D.P., . Examinining Data Quality. Technical Report.

Trier, D., Jain, A.K., Taxt, T., 1996. Feature extraction methods for character recognition - A survey. Pattern Recognition 29, 641–662. doi:10.1016/0031-3203(95)00118-2.

Tversky, A., Kahneman, D., 1971. Belief in the law of small numbers. Psychological Bulletin 76, 105–110. doi:10.1037/h0031322.

Tversky, A., Kahneman, D., 1974. Judgment under uncertainty: Heuristics and biases. Science 185, 1124–1131. doi:10.1126/science.185.4157.1124.

UNIT, 2019. Unit-Direktoratet for IKT og fellestjenester i høyere utdanning og forskning. Technical Report.

Vithlani, P., Scholar, R., Kumbharana, C.K., 2015. Structural and Statistical Feature Extraction Methods for Character and Digit Recognition. Technical Report 24.

Wise, M.J., 1993. (PDF) String Similarity via Greedy String Tiling and Running KarpRabin Matching. URL: https://www.researchgate.net/publication/262763983_String_Similarity_via_Greedy_String_Tiling_and_Running_Karp-Rabin_Matching.

Wolfe, E.W., Moulder, B.C., Myford, C.M., 2001. Detecting differential rater functioning over time (DRIFT) using a Rasch multi-faceted rating scale model. Journal of applied measurement 2, 256–80. URL: http://www.ncbi.nlm.nih.gov/pubmed/12011510.

Zen, K., Iskandar, D.N., Linang, O., 2011. Using latent semantic analysis for automated grading programming assignments, in: 2011 International Conference on Semantic Technology and Information Retrieval, STAIR 2011, pp. 82–88. doi:10.1109/STAIR.2011.5995769.

Zhao, H., Andersson, B., Guo, B., Xin, T., 2017. Sequential Effects in Essay Ratings: Evidence of Assimilation Effects Using Cross-Classified Models. Frontiers in Psychology 8, 933. URL: http://journal.frontiersin.org/article/10.3389/fpsyg.2017.00933/full, doi:10.3389/fpsyg.2017.00933.

Bakken, Edvard Gjessing

NTNU
Kunnskap for en bedre verden