Algorithms and AI in Education

Univesity of Copenhagen - Faculty of Science - Computer Science Department

Niklas Hjuler

July 2019

0.1 ABSTRACT

In this thesis we show several new results in a broad spectrum, ranging from theoretical computer science to data analysis. A brief summary of the results are:

One-Way Trail Orientations: We show that there exists a strong orientation of a graph if and only if the graph is two edge connected, even if the edges are partitioned into trails, thus improving Robbins theorem from 1939.

Dominating Sets and Connected Dominating Sets in Dynamic Graphs: We show the first results in how to maintain a $O(\log(n))$ approximation of a minimum dominating set and a minimum connected dominating set. The solution are maintained in running time $O(\delta \cdot \text{polylog}(n))$

Fully Dynamic Consistent Facility Location: We show how to maintain a constant approximation for the facility location problem for general metrics with running time $O(\log(n)n)$ and total recourse $O(n)$.

Detecting ghostwriters in high schools: Author verification, i.e. the task of verifying if an acclaimed author of an essay actually is the author. This is a known problem in the Natural Language Processing community. What is new, is doing this in the educational setting, challenging whether the student has actually written the assignment in question. With our method on a balanced data set, we achieve an accuracy of 87.5 percent.

Investigating Writing Style Development in High School: For every student we make a profile by comparing how similar a current assignment is to previous assignments. The profiles are then clustered and analyzed. Furthermore, we compare how the average similarity evolves over time.

Sequence Modelling For Analysing Student Interaction with Educational Systems: Using log data we model student behavior as a distribution of Markov chains. The Markov chains are analyzed, and 125.000 of the sessions are deemed suboptimal from a learning perspective.

Tracking Behavioral Patterns among Students in an Online Educational System: We make a soft clustering of a students activity during one week, using log data from an educational system. Based on the results, we give suggestions for an improved learning experience for the students.

DABAI: A data driven project for e Learning in Denmark: In this paper we give an overview of the goals of the projects involved in DABAI, and what issues the educational companies are interested in.

This thesis is written as a manuscript with an addendum of published papers and papers in submission. The descriptions of the papers are largely based on the introduction of the papers, with some parts being close to identical.

0.2 DANSK RÉSUMÉ

I denne thesis viser vi et bredt spektrum af resultater, fra teoretisk datalogi til mere praktisk data analyse. En kort beskrivelse af alle resultaterne er:

One-way trail orientations: Vi viser, at der findes en stærk orientering af en graf, hvis og kun hvis grafen er 2 kants forbundet, selv hvis kanterne er partitioneret til veje. Dette forbedrer Robbins theorem fra 1939.

Dominating sets and connected dominating sets in dynamic graphs: Vi viser de første resultater til at bevare en $O \log(n)$ approximation af Minimum dominerende mængde og forbundet minimum dominerende mængde. Løsningerne er bevaret i køretid $O(\delta \cdot \text{polylog}\ (n))$.

Fully Dynamic Consistent Facility Location: Vi viser, hvordan man bevarer en konstant approximation for facilitets lokation i en general metrik med en køretid på $O\ (\log(n))$ og rekurs på $O\ (n)$

Detecting ghostwriters in high schools: Forfatter bekræftelse, altså problemstillingen om en given forfatter er den faktiske forfatter, er et kendt problem i NLP fællesskabet. Det nye i denne artikel er at benytte det til at afgøre, om en elev har fået en anden til at lave sin aflevering. Dette lykkes med 87.5 procent success på et balanceret data sæt.

Investigating Writing Style Development in High School: For hver studerende laves en profil af skrivestil, baseret på en sammenligning af de første afleveringer mod afleveringer over tid. Profilerne grupperes og analyseres. Derudover sammenligner vi, hvordan den gennemsitlige similiaritet udvikler sig over tid.

Sequence Modelling For Analysing Student Interaction with Educational Systems: Vi bruger log data fra EDULAB til at modellere studerendes opførsels som en distribution af Markov kæder. Vi analyserer kæderne, og finder, at ca 125.000 af dem viser suboptimal læring.

Tracking Behavioral Patterns among Students in an Online Educational System:: Vi laver en svag gruppering af studerendes aktivitet af perioder på en uge, baseret på log data for undervisningssystemet CLIO ONLINE. Baseret på det giver vi anbefalinger til, hvordan de studerendes læringskurve kan forbedres.

DABAI: A data driven project for e Learning in Denmark: I denne artikel giver vi overblik over projekterne involveret i DABAI, samt hvilke problemer firmaer i undervinsnings sektoren står over for, og som kan løses ved brug af data.

Denne afhandling er skrevet som manuskript med publicerede artikler og artikler i submission vedhæftet til sidst. Beskrivelsen af aftiklerne er baseret i stor stil på introduktions stykket i aftiklerne, med nogen dele tæt på identiske.

CONTENTS

0.3 PREFACE

This thesis is written as a synopsis based on the papers below, to comply with the general rules and guidelines of the University of Copenhagen, Faculty of Science, for the PhD program. The synopsis consists of a short description of the main results in each paper. The papers are attached at the end, and they might differ slightly from the version in the proceedings, but the results and conclusions remain in perfect alignment. Of the papers described in this thesis, seven have been accepted at peer reviewed venues and the remaining one in submission.

- Anders Aamand, Niklas Hjuler, Jacob Holm, and Eva Rotenberg, "One-way Trail Orientations" In: *Proc. 45th International Colloquium on Automata, Languages and Programming (ICALP). 2018*
- Niklas Hjuler, Giuseppe F. Italiano, Nikos Parotsidis, David Saulpic, "Dominating Sets and Connected Dominating Sets in Dynamic Graphs" In: *The 36th International Symposium on Theoretical Aspects of Computer Science (STACS'19)* Algorithm in $O(\sqrt{m})$ update time in submission
- Niklas Hjuler, Nikos Parotsidis, David Saulpic, "Fully Dynamic Consistent Facility Location", In: *Submission to NIPS 2019*
- Magnus Stavngaard, August Sørensen, Stephan Lorenzen, Niklas Hjuler, Stephen Alstrup, "Detecting Ghostwriters in High Schools", In: *European Symposium on Artificial Neural Networks (ESANN 2019)*
- Stephan Lorenzen, Niklas Hjuler, Stephen Alstrup "Investigating Writing Style Development in High School", *Proceedings of the 12th International Conference on Educational Data Mining (EDM 2019)*
- Christian Hansen, Casper Hansen, Niklas Hjuler, Stephen Alstrup, Christina Lioma, "Sequence Modelling For Analysing Student Interaction with Educational Systems", In: *Proceedings of the 10th International Conference on Educational Data Mining (EDM 2017)*
- Stephan Lorenzen, Niklas Hjuler, Stephen Alstrup "Tracking Behavioral Patterns among Students in an Online Educational System", In: *Proceedings of the 11th International Conference on Educational Data Mining (EDM 2018)*
- Stephen Alstrup, Casper Hansen, Christian Hansen, Niklas Hjuler, Stephan Lorenzen, Ninh Pham, "DABAI: A data driven project for e Learning in Denmark", In: *Proceedings of the 16th European Conference on e-Learning (ECEL 2017)*

0.4 INTRODUCTION

This thesis contains a short introduction of the papers I have written during my PhD studies. For each paper there is a summary of the main results of the paper, a description of related work, a brief description of possible applications, and where applicable, a discussion of ethically issues regarding the paper. The introductions are in no way mathematically thorough, but meant to give an overview. If a deeper curiosity is sparked, the entire paper can be consulted in the appendix. The papers in this thesis span a very broad scope, from pure theoretical computer science, over practical machine learning applications, to data analysis. In chapter 0.5 we go over the algorithmic papers, which includes one paper regarding strong orientations of graphs, one regarding probabilistic labeling schemes, two about maintaining a property in a graph (Dominating Set and Independent Set), and finally one where we maintain a constant approximation in different clustering problems. In the second chapter 0.6 we go through three articles about student profiling. One with focus on writing style, and two others based on log data. Finally we have a paper about detecting ghostwriting in high school assignments. In the last chapter 0.7 we go through one paper about digital challenges in the educational sector, furthermore discussing the cooperation with companies involved, and how they possibly could apply my research. This reflects the burden that was given to me as a PhD student at the University of Copenhagen in the DABAI project, where I was asked to contribute in all these fields.

Besides the papers: During my PhD I have supervised three bachelor projects, all in the subject of intelligent tutoring systems, supervised two students in a project about authorship verification, and supervised two master thesis students about Ghostwriting detection. Furthermore I have taught classes in both Machine learning and Algorithms. I would like to thank all the students I have supervised and taught for good cooperation and for doing a great job. I have taken 30 ECTS points in a wide range of classes, ranging from technical machine learning and graph theory classes to ethical and pedagogical classes. I like to thank all my teachers for improving my knowledge in various areas and aspects of research. In 2018 I had a four month stay at Tor Vergata University in Rome, Italy, with Professor Giuseppe F. Italiano as my host. During my stay at Tor Vergata I worked with dynamic graph problems such as: Maximal Independent Set, Dominating Set and clustering. I like to thank Giuseppe F. Italiano for welcoming me into his group with open arms, which also extends to two of my colleagues in Rome, David Saulpic and Nikos Parotsidis. I have attended numerous conferences and summer schools in Machine Learning, algorithms, and educational data analysis, and I would like to thank the people involved and the communities for

their work and great inspiration. My research has been cited more than a hundred times by media all over the world in multiple languages including Danish, English, German, and Spanish.[1] The work I did during my PhD was in close cooperation with three companies: MaCom, CLIO online, and Edulab. I like to thank the people involved in the cooperation for their valuable effort and assistance. I like to thank my supervisors Mikkel Abrahamsen, Christina Lioma, and Stephen Alstrup for their guidance. I like to thank my family and friends, and specially my girlfriend Emilie Brun for her love and support.[2] I like to thank for my father for his support and assistance in the making of this thesis.

1 A subset of the medias can be seen in the appendix

2 Which was no small feat at daunting times

0.5 ALGORITHMS

0.5.1 *Introduction to Algorithms*

In this section I will go through all my papers relating to algorithms. For all of them it can be said, that they are related to graph theory. The first four papers are clearly embedded into graph theory and the last paper, about fully dynamic clustering, is on the borderline related to graph theory. We see a graph as a set $G = (V, E)$ with nodes V and edges E. Unless otherwise specified the graph is not directed and unweighted.

0.5.2 *One-way trail orientations [AHHR18]*

Imagine a town, and the following question: When can you make all the streets of a town one-way, without losing the property that you can get from anywhere to anywhere. In 1939 Robbin [Rob39] showed, that given a graph G, every edge can be directed such that the resulting graph is strongly connected if and only if G was $2-$edge connected. In 1980 Bosch [BT80] showed, that given a mixed multigraph G, the remaining undirected edges can be directed, such that the resulting graph is strongly connected if and only if G is connected and the underlying graph is bridgeless. However; Turing award winner Professor Robert Tarjan naturally noticed, that a street in the real world rarely correspond to a single edge in the corresponding graph. Thus he asked the same questions, but now the edges of the graph are partitioned into trails, and the trails must be oriented. Part of our contribution is the following two theorems:

Theorem 0.5.1. *Let $G = (V, E)$ be an undirected multigraph with E partitioned into trails. An orientation of each trail such that the resulting directed graph is strongly connected exists if and only if G is 2-edge connected.*

which is a beautiful extension of Robbins [Rob39] original theorem. The proof is done by a simple and elementary induction on the number of edges in the graph. The proof also inspires how to find a strong orientation in polynomial time, which was the first of its kind. In the mixed case, things are still elementary but not simple:

Theorem 0.5.2. *Let $G = (V, E)$ be a strongly connected mixed multigraph. Then $G - e$ is strongly connected for all undirected $e \in E$ if and only if for any partition $\mathcal{P}$ of the undirected edges of G into trails, and any $T \in \mathcal{P}$, any orientation of T can be extended to a strong trail orientation of $(G, \mathcal{P})$.*

To understand this theorem, think of an undirected edge as forced, if there exists a cut, such that all the directed edges only go from one side to the other. This "forces" the direction of our undirected edge. Then the first part of the theorem states, that there are no

forced edges. This leads to a simple algorithm, where we direct the forced edges (and their corresponding trail), and if there is none, you can direct any. The theorem above guarantees, that this algorithm is successful if a strong orientation did exist. Understanding the mixed case is clearly not a resolved matter, and I strongly encourage further research to clear this up. To understand that the mixed case actually differ in this setting, take a look at the following graph.

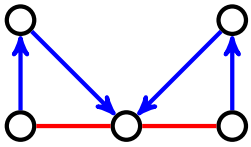


Figure 1: The graph is strongly connected, and the underlying graph is 2-edge connected, but irrespective of the choice of orientation of the red trail, the graph will no longer be strongly connected.

As described in the paper "one way trail orientations", we find a number of additional results. We show a linear time algorithm for solving the trail orientation problem in undirected graphs. For this, we make two essential observations. First, we show that there is an easy linear time reduction from general graphs or multigraphs to cubic multigraphs. Second, we show that in a cubic multigraph with n vertices, we can in linear time find and delete a set of edges, that are at the end of their trails, such that the resulting graph has $\Omega(n)$ 3-edge connected components.

We further show, that we can compute the required orientation recursively from an orientation of each 3-edge connected component together with the cactus graph of 3-edge connected components. Since the average size of these components is constant, we can compute the orientations of most of them in constant time individually, and thus in linear time taken together. The rest contains at most a constant fraction of the vertices, and so a simple geometric sum argument tells us, that the total time is also linear.

0.5.3 *Dominating Sets and Connected Dominating Sets in Dynamic Graphs [HIPS19]*

The study of dynamic graph algorithms is a classical area in algorithmic research, and has been thoroughly studied in the last few decades. Maintaining a solution of a graph problem in the case, where the underlying graph changes dynamically over time, is a big challenge in the design of efficient and practical algorithms. Indeed, in several applications, due to the dynamic nature of today's data, it is not sufficient to compute a solution to a graph problem only once and for all: Often, it is necessary to *maintain* a solution efficiently while the input graph is undergoing a sequence of dynamic updates. More precisely, a *dynamic graph* is a sequence of graphs $G_0, ..., G_M$ on n nodes and

such that G_{i+1} is obtained from G_i by adding or removing a single edge. The natural first barrier in the study of dynamic algorithms, is to design algorithms, that are able to maintain a solution for the problem at hand after each update faster than recomputing the solution from scratch. Many dynamic graph problems such as minimum spanning forests (see e.g. [HLT01; NSW17]), shortest paths [DI04], matching [BFH; NS16; Sol16], or coloring [BCHN18] have been extensively studied in the literature, and very efficient algorithms are known for those problems. Recently, a lot of attention has been devoted to the **Maximal Independent Set** problem (MIS). In this problem, one wishes to find a maximal set of vertices that do not share any edge ("maximal" meaning that it is not possible to add any vertex without violating this property). Until very recently, the best known update bound on the complexity to maintain a MIS, was a simple $O(\Delta)$ algorithm, where Δ is an upper bound on the degree of vertices in the graph. This bound was first broken by Assadi et al. [AOSS18], who gave an $O(m^{3/4})$ algorithm, then Gupta and Khan [GK18] improved the update bound to $O(m^{2/3})$. Using randomization, Assadi et al. [AOSS] presented an amortized fully-dynamic algorithm with an expected $\widetilde{O}(n^{1/2})$-time bound per update.

The MIS problem is closely related to the **Dominating Set** (DS) problem: Given a graph $G = (V, E)$ the DS problems is to find a subset of vertices, $\mathcal{D} \subseteq V$, such that every vertex in G is adjacent to $\mathcal{D}$ (or *dominated* by $\mathcal{D}$). Indeed, a MIS is also a Minimal DS: The fact that it is not possible to add a vertex without breaking the independence property implies, that every vertex is adjacent to the MIS, so this must also be a DS; at the same time, it is not possible to remove a vertex, since that vertex is no longer dominated. Thus, to find a Minimal DS, one can simply find a MIS: This immediately gives a deterministic $O(m^{2/3})$ [GK18] bound and a randomized $\widetilde{O}(n^{1/2})$ [AOSS] one. However, while it is known that is hard to approximate **Maximum Independent Set**[3] within a factor $n^{1-\epsilon}$ for every $\epsilon > 0$[Hås99], a simple greedy approach achieves a $O(\log n)$-approximation for Minimum DS [Chv79].

In recent years, there has been a lot of work done on designing dynamic graph algorithms for maintaining approximate solutions to several problems. A notable example is matching, where for different approximations there exist different algorithms (see e.g., [BFH; BS16; NS16; GP13; BHI18; Sol16]). This raises the natural question of whether there exists a dynamic algorithm capable of maintaining an approximation to Minimum DS, and even better a $O(\log n)$ approximation. In this paper, we answer this question affirmatively, by presenting an algorithm that achieves a $O(\log n)$ approximation, with a complexity matching the long standing $O(\Delta)$ bound for MIS.

3 It is not possible to find a polynomial-time algorithm that finds a $n^{1-\epsilon}$-approximation to **Maximum Independent Set** under the assumption NP $\neq$ ZPP

Moreover, if one is interested in finding a DS faster, we present a very simple *deterministic* $O(m^{1/2})$ algorithm to compute a Minimal DS, improving the $O(m^{2/3})$ bound coming from MIS. We believe these are important steps towards understanding the complexity of the problem. Those two results are stated below.

Theorem 0.5.3. *Starting from a graph with n vertices, a $O(\log n)$ approximation of* ***Minimum Dominating Set*** *can be maintained over any sequence of $\Omega(n)$ edge insertions and deletions in $O(\Delta \log n)$ amortized time per update, where Δ is the maximum degree of the graph over the sequence of updates.*

Theorem 0.5.4. *Starting from a graph with n (fixed) vertices, a* ***Minimal Dominating Set*** *can be deterministically maintained over any sequence of edge insertions and deletions in $O(\sqrt{m})$ amortized time per update, where m is an upper bound on the number of edges in the graph.*

We also study the **Minimum Connected Dominating Set** problem (MCDS), which adds the constraint, that the graph induced by the DS $\mathcal{D}$ must be connected. This problem was first introduced by Sampathkumar and Walikar [SW79] and arises in several applications. The most noteworthy is its use as a *backbone* in routing protocols: It allows us to limit the number of packet transmissions, by sending packets only along the backbone, rather than throughout the whole network. Du and Wan's book [DW12] summarizes the knowledge about MCDS. A special class of graphs are geometric graphs, where vertices are points in the plane, and two vertices are adjacent if they fall within a certain range (say, their distance is at most 1). This can model wifi transmissions, and the dynamic MCDS has been studied in this setting: A polynomial-time approximation scheme is known [CHL+03], and Guibas et al. [GMM13] show how to maintain a constant-factor approximation with polylogarithmic update time. While geometric graphs model problems are linked to wifi transmissions, the general graph setting can also be seen as a model for wired networks. However, no work about dynamic MCDS is known in this setting: The static case is well studied, with a greedy algorithm developed by Guha and Keller [GK98], that achieves an approximation factor $O(\ln \Delta)$. They also show a lower bound matching their complexity, together with their approximation factor. MCDS has also been thoroughly studied in the distributed setting. A heuristic to find a Minimal CDS can be seen in [BCOP04], another one that sends $O(\Delta n)$ messages and has a time complexity at each vertex $O(\Delta^2)$ [WL99] or a $3 \log n$ approximation that runs in $O(\gamma)$ rounds where γ is the size of the CDS found, with time complexity $O(\gamma\Delta^2 + n)$ and message complexity $O(n\Delta\gamma + m + n \log n)$ [BB97]. Despite all this work, no results are known in the dynamic graph setting. As another application of our approach, we contribute to filling this gap in the research line of MCDS. In particular, in this paper we show how our

algorithm for Minimum DS can be adapted in a non-trivial way to maintain a $O(\log n)$ approximation of the MCDS in general dynamic graphs.

Theorem 0.5.5. *Starting from a graph with n vertices, a $O(\log n)$ approximation of* ***Minimum Connected Dominating Set*** *can be maintained over any sequence of $\Omega(n)$ edge insertions and deletions in $\widetilde{O}(\Delta)$ amortized time per update.*

We further show how to maintain independently a Dominating Set $\mathcal{D}$ and a set of vertices $\mathcal{C}$, such that the induced subgraphs on the vertices $\mathcal{C} \cup \mathcal{D}$ are connected. The set $\mathcal{C}$ has the additional property that $|\mathcal{C}| \leqslant 2|\mathcal{D}|$, such that $|\mathcal{C} \cup \mathcal{D}| = O(|\mathcal{D}|)$. If $\mathcal{D}$ is a α-approximation of Minimum DS, this gives a $O(\alpha)$ approximation for MCDS.

FURTHER RELATED WORK. It is well known that finding a Minimum DS is NP-hard [GJ79]. It is therefore natural to look for approximation algorithms for this problem. Unfortunately, it is also NP-hard to find a $c \log n$ approximation, for any $0 < c < 1$ [Fei98]. This bound is tight, since there is a simple greedy algorithm matching this bound [Chv79]. Minimum DS had been studied extensively in distributed computing: An algorithm which runs in $O(\log n \log \Delta)$ rounds finds an $O(\log n)$ approximation with high probability [JRS02], and an algorithm with constant number of rounds achieves a non-trivial approximation[KW05].

The DS problem is closely related to the **Set Cover** problem: The two problems are equivalents under L-reduction [Kan92]. However, **Set Cover** was studied in the dynamic setting [GKKP17; AS18], but with different kinds of updates: Instead of edges being inserted or deleted (which would represent new elements in the sets according to the L-reduction), new elements are being added to the cover (which would be new vertices in DS).

0.5.4 *Fully Dynamic Consistent Facility Location [***clustering***]*

Clustering is a core procedure in unsupervised machine learning and data analysis. Due to the large number of applications, clustering problems have been extensively studied for several decades. The existing literature includes both very precise algorithms[ANSW17; Gon85; Li13], and very fast ones [MP04]. Due to the importance of the task, clustering problems have also been studied in several computing settings, such as the streaming model [COP03], and the sliding-window model [BLLM16], the distributed model [BLK17], the dynamic model [HGS18], and others.

Applications nowadays operate on dynamically evolving data, e.g., pictures are constantly added and deleted from picture repositories, purchases are continuously added into online shopping systems, reviews are added or being edited in retail systems, etc. Due to the

scale and the dynamic nature of the data at hand, conventional algorithms designed to operate on static inputs, become unable to handle the task for two main reasons. First, the running time of even the most efficient algorithms is too expensive to execute after every single change in the input data. Secondly, re-running a static algorithm after every update, might generate solutions that differ substantially between consecutive updates, which might be undesirable for the application at hand. The number of changes in the maintained solution between consecutive updates is called the *recourse* of the algorithm. Our study is motivated by these limitations of static algorithms or dynamic algorithms, that are effective on only one of the two objectives.

Most fundamental problems in computer science have been studied in the dynamic setting. At a very high-level, a dynamic algorithm computes a solution on the initial input data, and as the input undergoes insertions and/or deletions of elements, the algorithm updates the solution to reflect the current state of the data. A dynamic algorithm may allow only insertions or only deletions, or may support an intermixed sequence of insertions and deletions, in which case the algorithm is called fully-dynamic. The running time of a dynamic algorithm can either guarantee a worst-case update time after each update, or a bound on the average update time over a sequence of updates, which is called amortized update bound. A dynamic algorithm with worst-case update bounds is the most desirable, and often hard to obtain, but in several applications algorithms with amortized update bounds are sufficient.

In this paper, we study fully-dynamic algorithms for classic clustering problems. In particular, we consider the facility location, the k-means, and the k-median problems in the dynamic setting. In the static case, these problems are defined as follows. Let X be a set of n points, and $d : X \times X \to \mathbb{R}$ a distance function. We assume that d is symmetric and that (X, d) forms a metric space. For the (k, p)-clustering problem, the objective function that we seek to optimize is $C_p(X, S)$, where $S \subseteq X, |S| = k$. $p = 1$ gives the k-median objective, and $p = 2$ the k-means one. For the facility location problem the objective function is $C(X, S)$.

$$C(X, S) := \sum_{x \in X} \min_{c \in S} d(x, c) + f \cdot |S| \qquad C_p(X, S) := \sum_{x \in X} \min_{c \in S} d^p(x, c),$$

All these problems are NP-Hard, so our best hope is to design algorithms with provable approximation guarantees. In the dynamic setting, the goal is to efficiently maintain a good solution to the clustering problem at hand, as the set of points undergoes element insertions and deletions. The main criterion for designing a *good* dynamic algorithm for these problems, is the quality of the clustering, with respect to the optimum solution at any given time. However, in many

applications, it is equally important to maintain a *consistent* clustering, namely a clustering with bounded recourse. Lattanzi and Vassilvitskii [LV17] have recently considered consistent clustering problems in the online setting, where the points appear in sequence, and the objective is to maintain a constant factor approximate solution, while minimizing the total number of times the maintained solution changes over the whole sequence of points. Another criterion, that has been much less explored, but is of high importance when dealing with massive data, is update time. This criterion is the time it takes to update the solution after each insertion/deletion so that the solution remains a constant factor approximate solution.

Our Contribution

We present the first work, that studies fully-dynamic algorithms while considering all of the three aforementioned objectives at the same time: The approximation guarantee, consistency and update time. From an input perspective, we consider general metric spaces, an element of the input is thus a point in that metric space, which is defined by the distances to the other points of the metric. The contribution of our paper can be summarized as follows:

• We give a fully-dynamic algorithm for the facility location problem with a constant factor approximation, a constant number of changes to the clustering at each time step, and $O(n \log n)$ update time. We moreover show, that a constant number of changes per update is necessary for achieving a constant factor approximation.

• We extend the algorithm for facility location to the k-median and k-means problems. Here, our algorithm maintains a constant factor approximate solution with $\widetilde{O}(n + k^2)$[4] update time (Theorem 0.5.7). This is the first non-trivial result for these problems, as hitherto the only known solution was to recompute from scratch after each update: This requires time $\Omega(nk)$ for k-median and $\Omega(n^2)$ for facility location. Hence, our time bounds are significantly better than the naive approach for a large range of k.

EMPIRICAL ANALYSIS. We complement our study with an experimental analysis of our algorithm, on three real-world data sets, and show that it outperforms the standard approach which recomputes a solution from scratch after each update, using a fast static algorithm. Interestingly, we show that this barely impacts the approximation guarantee. At the same time, our algorithm outperforms by at least three orders of magnitude the simple-minded solutions, both in terms of running time and total number of changes made in the maintained solution throughout the update sequence.

4 $\widetilde{O}(\cdot)$ hides polylog factors.

Related Work

ONLINE AND CONSISTENT CLUSTERING Online algorithms for facility location were first proposed by Meyerson [Mey01] in his seminal paper. Fotakis [Fot08] later showed that the algorithm, has a competitive ratio of $O(\log n / \log\log n)$, which is also optimal. Additionally, the algorithm has a constant competitive ratio if the points arrive in random order [Mey01; Lan18]. There also exist $O(\log n)$ competitive deterministic algorithms, see [ABUH04; Fot07]. This was recently extended to the online model, that incorporates deletions [CCMS18].

Online algorithms for clustering, that are only allowed to place centers, cannot be competitive. This led to the consideration of the incremental model, which allows two clusters to be merged at any given time. Work in this area includes [CCFM04; Fot06]. The number of reassignments (commonly referred to as *recourse*) over the execution of an incremental algorithm, may be arbitrary. However, recently Lattanzi and Vassilvitskii [LV17] considered the online clustering problem with bounded total recourse. They showed a lower bound of $\Omega(k \log n)$ changes over an arbitrary sequence of updates, and presented an algorithm that can maintain a constant factor approximation, while limiting the total recourse to $O(k^2 \cdot \log^4 n)$. Their work differs to ours in that elements can only be added, and that they do not consider optimizing the running time. In the fully dynamic case, their bound on the recourse does not hold, and we moreover show, that constant recourse per update is unavoidable.

FULLY-DYNAMIC AND STREAMING ALGORITHMS. Streaming algorithms for clustering can be used to obtain fast dynamic algorithms, by recomputing a clustering after each update. Since streaming algorithms are highly memory compressed, and typically process updates in time linear in the memory requirement, the approach automatically yields good update times. Low-memory adaptations of Meyerson's algorithm [Mey01] turned out to be simple and particularly popular, see [BMO+11; Lan18; SWM11]. Another technique for designing clustering algorithms in the streaming models, is by maintaining coresets, see the following recent survey for an overview [MS18]. For fully dynamic data streams, the only known algorithms for maintaining coresets for k-means and k-median in Euclidean spaces using small space and update times, are due to Braverman et al. [BFL+17] and Frahling and Sohler [FS05]. There also exists some work on estimating the cost of Euclidean facility location in dynamic data streams, see [CLMS13; Ind04; LS08].

For more general metrics, the problem of maintaining clusterings dynamically, has been considered by Henzinger et al. [HLM17] and Goranci et al. [GHL18] who consider the facility location in bounded doubling dimension. The arguably most similar work previous to ours is due to Hubert-Chan et al. [HGS18]. They consider the k-center

problem in general metrics in the fully dynamic model. Here, they were able to maintain a constant factor approximation with update time $O(k \log n)$ [5].

Whether an algorithm in the fully dynamic model with low recourse and update times exists, was left as an open problem.

In this paper we consider the classical clustering problems: Facility location, K-means, K-median, but in a fully dynamic data stream setting. Three parameters are of importance in this context: (1) How good the solution is, (2) the time it takes to update the solution, and (3) how much the solution changes.

In this paper we focus on general metric space. Most of the focus is on the facility location problem. We give a simple algorithm, which maintains a constant factor approximation with update time $O(\log(n)\, n)$ and total recourse $O(n)$. This is better than the naive algorithm, which consists of recomputing at every timestamp, which can take up to $O(n^2)$ running time and a total recurse of $O(n^2)$. Our bounds are nearly optimal, since inserting a point in a general metric space takes $O(n)$ time, and we show a simple lower bound of $O(n)$ on total recourse. Moreover, we generalize this result for the k-medians and k-means problems: Our algorithms maintain a constant factor approximation in time $\widetilde{O}(n + k^2)$.

We complement our analysis with experiments, showing that the cost of the solution maintained by our algorithm at any time t, is very close to the cost of a solution obtained by quickly recomputing a solution from scratch at time t while having a much better running time.

A quick statement of the results

For facility location we get the following theorem.

Theorem 0.5.6. *There exist a randomized algorithm that, given a metric space undergoing insertion and deletions of points, maintains a set of center S^t such that :*

- *each update is processed in time $O(n^* \log(n^*))$ with probability $1 - 1/n^*$*
- *at any given time t, $C(X^t, S^t) = O(1)C(X^t, OPT^t)$ with probability $1 - 1/n^*$*
- *$\sum_{t=0}^{n} |S^t \Delta S^{t+1}| = O(n)$, i.e. the amortized recourse is $O(1)$ per step.*

and for k-means and k-median

5 Under the common assumption that the ratio longest distance / shortest distance of the metric is polynomially bounded.

Theorem 0.5.7. *There exists a randomized algorithm that, given a metric space undergoing insertions and deletions of points, maintains a set of centers S^t with $\widetilde{O}(n^* + k^2)$ update time such that, for any time t, $C_p(X^t, S^t) = O(1) \cdot C_p(X^t, OPT^t)$.*[6]

and the empirical results are

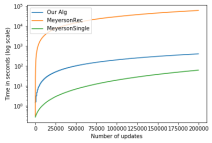


Figure 2: Running time, Twitter.

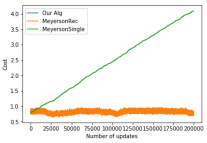


Figure 3: Cost, Twitter.

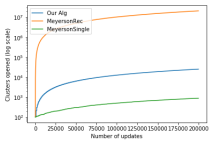


Figure 4: Recourse, Twitter.

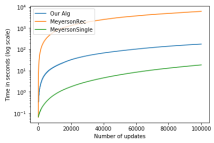


Figure 5: Running time, covertype.

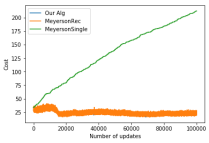


Figure 6: Cost, covertype.

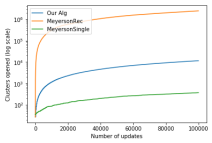


Figure 7: Recourse, covertype.

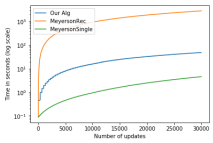


Figure 8: Running time, USCensus1990.

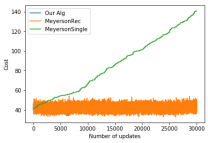


Figure 9: Cost, USCensus1990.

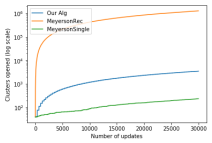


Figure 10: Recourse, USCensus1990.

Figure 11: A comparison of the algorithms we consider in terms of running time (left column), cost of the solution (middle column), and recourse (right column).

6 We assume (as in [LV17]) that the minimum distance in the metric is 1 and the maximum Δ is bounded by a polynomial in n^*. Alternatively, our bounds can be stated with $\log \Delta$ instead of $\log n^*$.

0.6 MACHINE LEARNING AND DATA ANALYSIS

0.6.1 *Introduction*

In this section we go through four papers related to machine learning and data analysis. The first paper: "Ghost Writing Detection in High Schools" is using a Siamese network to estimate how similar two assignments are, where two assignments from the same author is said to be similar. Then this network is used to guess, if a new assignment from a student is written by the student himself or by the use of a ghost writer. In "Investigating Writing Style Development in High School", the Siamese network from before is used on a student to track how similar his newer assignments are compared to previous work. This gives a profile of each student, which is then analyzed and the profiles are clustered. In both "Sequence Modelling For Analyzing Student Interaction with Educational Systems" and "Tracking Behavioral Patterns among Students in an Online Educational System" we do clustering of log data, though with different models and sources of data. In the first it is done with a model, based on a mixture of Markov chains, and in the second it is done with Non-Negative Matrix Factorization.

0.6.2 *Ghost Writing Detection in High schools [SSL+19b]*

In recent years, the possibility of hiring a ghost writer to do your academic work, has increased due to the rise in the number of services who offer to do this for you. Compared to plagiarism, this is a much harder task to detect, since the work is original, just not written by the student himself. The problem is of course only solveable, if you have a history of assignments from the student, and thus the problem is answering whether the new assignment is written by the same author as the earlier assignments. In the Natural language processing community, this problem is called authorship verification and is well studied [Sta09; Bag15; BDD+15].

However, in the case of detecting ghost writing in written work in high school, to the best of our knowledge, this was the first attempt.

The network we used was a Siamese neural network, which was inspired by [QHZ18a]. The network is trained, such that given two documents, it determines how likely they are to be written by the same author. The Siamese part of the network gives an encoding of each of the documents in the network, which are then compared with four dense layers of 500 neurons in the end. This network was able to predict with about 72 percent accuracy, if two documents were written by the same author on a balanced data set.

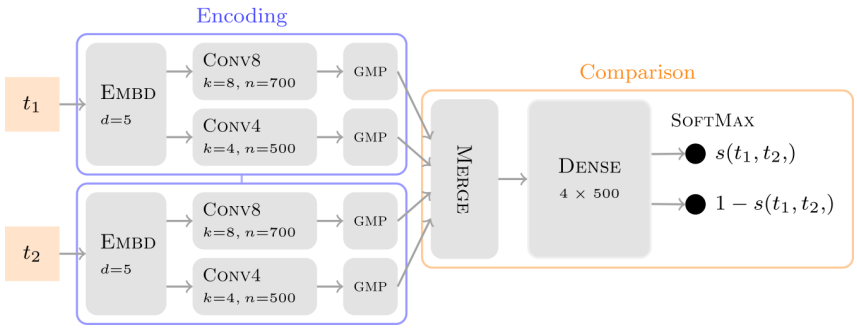


Figure 12: The architecture of the siamese network used

For the final prediction the student has a new assignment x and a history of assignments T_α. The final prediction is given by

$$C_s(T_\alpha, x) = \sum_{t \in_\alpha} e^{-\lambda \tau(t)} s(t, x)$$

where $\tau(t)$ is the number of months assignment t was written before the new assignment x, λ is a configurable parameter, and $s(x, t)$ is the similarity between the assignments as learned by the network. The new assignment x is said to be written by the student if $C_s(T_\alpha) \geqslant \delta$. The parameters λ, δ was found on a validation set. Using this prediction strategy, we got an accuracy of 87.5 percent and an AUC score of 0.947 on a balanced data set.

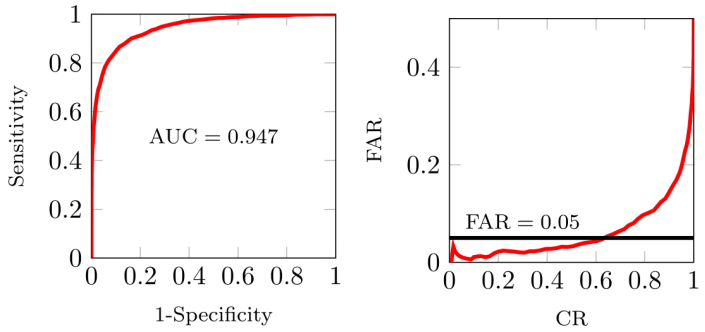


Figure 13: On the left we have the ROC curve, on the right is tradeoff between false accusation and catch rate

This project received lots of media attention, and has been mention by more than 60 danish media and more than 120 international media.[7] One of the reasons for the large media attention is at least partially due to it being controversial. There is a clear ethical dilemma here. On one side using ghostwriters can unfairly improve your grades, and in Denmark high school grades directly determines which further education programs you can pursue. Thus, the use of ghostwriters may push out more deserving students. Also, the use of ghost writers undermines the whole grade system, since students are given

7 see Appendix

grades based on other peoples work. On the other hand, falsely accusing a student of using a ghostwriter is a serious issue, and should in no way be taken lightly.

Confronted with the issue that the use of our research may lead to false accusations, our natural response is, that we are just a warning system. An alert should not lead to any consequences, before all traditional methods have been taken into account. In other words, the only difference our system make is "information" to the teacher/principal, and what he/she does with it is up to him/her. While such an answer is understandable, it does nothing but push the responsibility to the next level. We are in the usual situation, where we have to balance the false negatives (false accusation) and false positives (not detecting when a ghostwriter has been used). Many people like to cite Blackstone's principle: "It is better that ten guilty persons escape than that one innocent suffer" when I confronted with this dilemma. However my response "what about twenty?" either leaves them silent, or takes their sympathetic, but too idealistic, principle to the limit, concluding that no innocent may ever be accused.

Another ethical issue which is not specific to our paper, but to author verification in general, is the prospects of re identification based on text. For a long time there has been an underlying assumption, that text with no named author is anonymous. With the new GDPR rules, whether data is anonymous or not depends on whether the person, which the data belong to is identifiable or not. From recital 26 of EU GDPR we have the following quote:
"To determine whether a natural person is identifiable, account should be taken of all the means reasonably likely to be used, such as singling out, either by the controller or by another person to identify the natural person directly or indirectly.
To ascertain whether means are reasonably likely to be used to identify the natural person, account should be taken of all objective factors, such as the costs of and the amount of time required for identification, taking into consideration the available technology at the time of the processing and technological developments."

With authorship verification developing as it does, it changes which texts are anonymous and which are not. A famous, but not so devastating example, was when J.K Rowling was identified as the author of 'The Cuckoo's Calling', which she wrote under a pseudonym. I do not think though, it will be long, before we are faced with much more serious issues, such as identifying 'anonymous' dissidents.

0.6.3 *Investigating Writing Style Development in High School [LHA19]*

One of the most essential skills learned during the course of primary, secondary, and high school, is writing. While the main focus of primary school are on basic writing skills (such as grammar), secondary

or high school will be more focused on improving *the linguistic writing style* of a student, that is, the quality of the written text as perceived by the reader. With many jobs being highly dependent on producing relatively large amounts of well-written text, no justification is needed for why *good writing* is an essential skill.

The definition of quality in linguistic writing style is widely discussed [Spa01; B D74]. While correct grammar being a prerequisite, several other measures are also correlated to writing style being perceived as good, for instance use of vocabulary, sentence structure and readability [PN08]. Our focus in this work will mainly be on writing style *development* through the course of high school, while writing style quality will have a secondary role. We consider data from Danish high schools, consisting of Danish essays, and investigate the general development patterns among the students during the three years of study. The end goal is to be able to provide feedback to teachers about the development of their students' writing styles. We identify patterns among thousands of students across different classes and institutions, allowing us to provide teachers with new insights, which current data available to teachers might not show. For instance, insights about students, whose writing style development patterns may be unique within their own classes.

By itself, our method potentially allows for identifying students with deviating writing styles development (which might be good or bad), or students with sudden significant changes in writing style, which could be an indicator of cheating. However, we also consider several measures for the *quality* of writing. We investigate how these measures correlate with the different patterns of writing style development found, as a mean to detect optimal and suboptimal development profiles with respect to text quality. Information of this kind could be used to help teachers tailor their teaching style to specific groups of students, who may need training in specific areas challenging to their development profile.

Our Contribution

As mentioned, we concern ourselves with the development of linguistic writing style (as opposed to e.g. handwriting) during the course of high school. Specifically, we investigate the development of writing style in Danish essays handed-in by students in Danish high schools [8].

We are interested in determining general patterns of development, and to discuss which of the patterns are optimal, in the sense of improving writing style quality. In particular, we consider the following questions:

8 Note, that high school in Denmark usually consists of three years of study with students normally starting at age 15-17 and finishing at age 18-20.

- How does the writing style of a student develop, and what are the typical kinds of development in writing style?
- How does writing style changes correlate with measures of quality?
- How does writing style similarity between students behave, with respect to how far the students are in their education?

Our study is based on data from the company MaCom[9], which is behind the learning management system Lectio, a system used by 90% of Danish high schools. Students submit their written essays through Lectio, giving MaCom access to a huge corpus of Danish texts by high school students, marked with author and date of submission.

Our approach is based on methods from authorship verification; in order to learn a similarity measure for writing style, we consider examples of writing styles in texts from the same or different authors, similar to how it is done in verification tasks. We use a Siamese neural network for learning this similarity measure. While training, time is not taken into account. Assuming that writing style actually changes over time, this will lead to a suboptimal network. However, testing the network, we see clear patterns in how the "errors" distribute for a single author, indicating that the network simulates the best similarity measure possible, and the "errors" are actual changes in writing style. Using this method, writing style development profiles are generated and clustered for a large body of students. Analyzing the clusters, we see optimal and suboptimal types of development. In general, the average similarity is found to decay with time to a great extend, which corresponds well with the general perception, that writing style changes during high school, and also matches conclusions made in the literature [SSL+19a; HLLA14; CP04].

While this paper presents a case study of the data from MaCom, the methods used for analysis are of independent interest. They are not specific to the Danish language or high school, except for the neural network, which would at least require retraining in the given language. Considering other network architectures than the one used in this work, might also improve upon the analysis, see for instance [QHZ18b] for a network used with English.

Related Work

In one way or another, writing style analysis has been studied in the natural language community for many years. Typically, the analysis of writing style is used as a middle link for tasks such as *authorship verification* [SSL+19a; QHZ18b; Sta09], in which a text of unknown

9 The data set is proprietary and not publicly available

authorship is given, together with a set of texts by some known author, and we wish to verify, whether the given author is the author of the unknown text. Similarly, in *authorship attribution* the unknown text must be attributed to one of several known authors. Traditional methods for verification and attribution utilize both unsupervised methods from the field of outlier detection [Sta09], as well as standard supervised learning techniques, such as SVMs [HLLA14] and techniques based on neural networks [SSL+19a; QHZ18b].

Other uses of writing style analysis include distinguishing features of the writer (e.g. sex and age [SBSV13; SKA02; PDV11; AKFS03], demographics [AKPS09], or nationality [KSZ05]), using supervised learning algorithms such as SVMs, random forest, and neural networks. Other studies have investigated written conversations on online forums, trying to infer whether one person is trying to convince another [FBPW11].

Some studies investigate the quality of writing, for instance prediction of popularity of news articles [YCL+19], or the quality of scientific articles [LN13]. The former uses the popularity of an article on social media as a measure of quality, while the quality measure of scientific papers considered in the latter, is based on acceptance of a paper to "The Best American Science Writing", an anthology of popular science articles published in the United States on a yearly basis.

Few studies consider development of writing style as the main objective. [CS14] uses neural network models to track style of *handwriting* (i.e. not linguistic writing style) and investigate the development of handwriting among young students, and how similar it is when compared to different students, in the same/different grade level. [BD74] shows how students in higher grades get higher scores for their essays from teachers, in a blind experiment, where all student information is hidden from the grading teacher. [CP04] consider two famous Turkish writers, investigating their change in writing style over time, the most significant finding being average word length increasing with the age of the author.

Finally, several studies related to writing style have been conducted using the data available from MaCom. [HLLA14] investigates temporal aspects of authorship attribution, and concludes, that considering more recent essays improves authorship attribution algorithms, indicating that the writing style among high school students does indeed change with time. [SSL+19a] also uses the MaCom data for testing their neural network based authorship verification methods; their results also support these findings.

By using the Siamese network from above, we can tell how similar two essay are, or more exactly, how likely they are to be written by the same author. What was noticed when we did the "ghost writing detection", was that the further apart in time two assignments written

by the same author was, the less "similar" our network predicted them to be. If you think of the network as checking similar writing style, it is of no surprise that this happens, since students writing style develop a lot during high school. Though one needs to keep in mind, that the network is trained with plenty of pairs of assignments written by the same student but far apart in time, thus the conclusion is not trivial, but none the less the case.

We took the two first assignments of each student, and then calculated the similarity of these two assignments to all later assignments. Using these similarities, we created a profile of each student, which shows a graph of how the similar his later assignments was to his two first. Then we made a modified k++ algorithm and cluster the profiles, with the number of clustered determined by the elbow method. Then we analyzed all the clusters and evaluated which of them showed the best learning behavior by comparing to other metrics such as number of words, verbs per phrase and SMOG score. One of the clusters showed the similarity dropping so low, that after 30 months it could just as well have been another person writing it, when seen from the perspective of the algorithm.

We also looked at how the average similarity was between essays with different authors, depending on how far in high school they were when the assignment was written. This showed clearly, that the similarities decreased with time, an indication that students writing style develop through their high school years.

0.6.4 *Sequence Modelling For Analyzing Student Interaction with Educational Systems [HHH+17a]*

In this paper we used 1.08 million student sessions. We modeled each student as a distribution of different underlying behaviors, where the sequence of actions from each session is assumed to be generated by a single Markov chain. This means, that a student can act according to different types of behavior, but during one session will act only according to one Markov chain.

The method of the paper is as follows: For each session we extract a sequence of actions $A_1, ..., A_n$, and every sequence correspond to a path in the Markov chain model. We then generate priors $P_1, ...P_k$, which themselves are Markov chains, one for each cluster we want. The priors are generated by each edge given a random number from a uniform distribution between 0 and 1. Then every action sequence is assigned to the prior most likely to generate it, and we update each prior such that it maximizes the likelihood to generate the sequences assigned to it. Then we assign sequences again and update priors and this is done until less than 5 percent of the seqeunces change prior. Notice that this is just the k++ algorithm modified for this settting.

To decide the right number of clusters/priors, we used the elbow method, which means that you look for the elbow in the graph when the algorithm has been run for different number of clusters. In this case the graph is the probability that the priors would generate the sequences assigned to them. This should be increasing with the number of clusters, but there should be a sign once the number of clusters go above the "true" number. Using the elbow approach is very common, when you do not have expert domain knowledge. However, we did not see a clear break, but a strong indication that it was between 6-10, and we went with 6.

When we started analyzing the 6 chains, one of the chains showed strong signs of suboptimal behavior, where the student either experience questions that are too easy, too hard, or they never train what they learn in a lesson.

0.6.5 *Tracking Behavioral Patterns among Students in an Online Educational System [LHA18]*

How students act in educational systems is always an essential topic in educational data mining. Understanding this behavior in an educational systems can help us guide students in the direction of optimal learning, based on actual use of the system. This behaviour can be understood both through an explicit study [HMW+16], or, as in this paper, through the automatically generated log data of the system.

The analysis of log data is often done as an unsupervised clustering of students [FKD16; GRD+16; HHH+17b; KKSG16]. A popular approach is to extract action sequences, and transform them into an aggregated representation using Markov models [HHH+17b; KKSG16]. The Markov chains can then be clustered by different methods. Klingler et al. did student modeling with the use of explicit Markov chains and the clustering with different distance measures defined on the Markov chains [KKSG16]. Hansen et al. assumed the actions sequences to be generated by a mixture of Markov chains, and used an heuristic algorithm to find the generating Markov chains [HHH+17b]. Gelman et al. used non-negative matrix factorization to find clusters for different measures of activity, aggregated in weekly periods during a MOOC course. These clusters are then matched from week to week by cosine similarity.

Our work is similar to Gelman et al. [GRD+16] in that we also use *Non-negative Matrix Factorization* (NMF) to make a soft clustering at the student level in a given time period. However, our clustering is only made once, and we are looking at primary school data over a vastly longer period of time, (2 years compared to 14 weeks).

Our soft clustering by non-negative matrix factorization is based on log data from Clio Online. Clio Online is the largest provider of digital learning of all subjects in the Danish primary school (except

mathematics), having 90% of all primary schools in Denmark as customers.

Using NMF, we assume that the set of features chosen can be represented by a set of fewer underlying behaviors. These underlying behaviours would each be represented by a cluster in the non-negative matrix factorization. Each student will then get a number for each cluster in each time period, representing how much of that underlying behavior he has shown. Non-negativity gives the behaviors an additive structure, which is more natural than showing a negative amount of a given behavior. We reason that the soft clustering will show both the behaviors of individual students, as well as how the behaviors change over time, both individually and on a system-wide level.

In the paper, we consider two main questions: a) how does student activity in the system affect performance, and b) how does student activity distribute between different levels of Bloom's taxonomy in different subjects. Both questions are important in regards to optimizing learning; the first in relation to performance, the latter in relation to utilization of all taxonomy levels.

Several points can be taken from our analysis. We have identified three optimal and two sub-optimal behaviors in relation to subject and performance. One notably conclusion is that students using the Clio Online system during non-school hours (at home), do not seem to gain any significant boost to performance. We also see how taking quizzes seems to increase the performance of students in languages, more so than in other subjects where reading texts are of more importance. This fits the intuition that skills such as grammar need to be trained in order to be learned. We inform how exercises are used, depending both on their subject and their level in Bloom's taxonomy. And lastly we see that the average amount of time spent in the system is increasing both generally and for the individual students in all subjects, but especially for students working with languages. Furthermore, both experiments show how behaviors can have high correlation on a system-wide level, despite being uncorrelated on the individual student level. While the change of behavior for individual students was not directly analyzed in this paper (due to privacy concerns), our method allows for tracking such individual changes. Hopefully, this will help teachers encourage optimal student behavior, e.g. by recommending training quizzes for students working with languages, or making sure that students are allowed more time to use the system in school.

0.7 COMPANY APPLICATIONS

0.7.1 *Introduction*

While my PhD was not a traditional industrial PhD, it was in close cooperation with three companies. Edulab, who is responsible for "MatematikFessor", a Danish online math platform, where primary school students can practice and learn math. "MatematikFessor" is used by more than 1.500 primary school math teachers in Denmark, and more than a million questions is answered each day on the platform. CLIO online, which is another learning platform for primary schools, has the focus on all the other classes. Finally there was MaCom, who is behind the product Lectio, a Learning Management System for high schools, which is used by about 90 percent of high schools in Denmark. Working together with private companies presents itself with both challenges and opportunities. The opportunities relate to working with problems that the industry cares for and getting access to real world data sets, which might otherwise be unavailable. Working with real world data sets is a very messy job, and you have to take many decisions regarding the best way to clean your data. The right way to clean data is in no way obvious, and I encountered this issue several times during my PhD. An example was, when we wanted to predict the score of a student on a quiz before he took it, based on historical data of the student. In many cases the data indicated that the student was not doing his best, for example by using less than a minute on a test estimated to last an hour. Thus it became a question of predicting human behavior, instead of predicting how good the student would be at the quiz. While it was obvious that some attempts were not genuine, it was no easy task to decide where to draw the line.

0.7.2 *DABAI: A data driven project for e Learning in Denmark [**ECEL**]*

One of the first papers I wrote, was about the challenges faced by the companies we were cooperating with. One of them was student profiling. I have written three papers about student profiling during my PhD:Tracking Behavioral Patterns among Students in an Online Educational System, Sequence Modelling For Analyzing Student Interaction with Educational Systems, and Investigating Writing Style Development in High School. The common denominator in all these papers was, that suboptimal learning behavior was found, and alternatives suggested. The companies have incorporated those findings into the services to improve the learning experience of students.

Another problem which originates from MaCom, was authorship verification. This lead to the paper: Ghost Writer Detection in High

Schools. This paper received extraordinary media attention and has been cited by hundreds of media all over the world.

0.8 CONCLUSION

I have contributed with three papers to the algorithms community, one to the machine learning community, and four to the educational community. I have shown companies how to use their data for getting insights which can help improve their system, and also given them ways to solve problems, which was previously unheard of in their setting. I have spread our ideas and results, not just for the science community, but for people in general, by having our research described by hundreds of medias with reach to millions of people. Such impacts are unusual, but necessary, if you want more than just the community to be aware.

I have extended Robinsons original theorem by theorem 0.5.1.This led to the finding of a strong orientation in polynomial time.

Theorem 0.5.2 led to a simple algorithm, where the theorem guarantees the algorithm to be successful, if a strong orientation exists. Furthermore, I have shown, that there is a linear time algorithm for solving the trail orientation in undirected graphs.

Dominating Sets and Maximum Independent Sets: I have demonstrated, that there exists a dynamic algorithm capable of maintaining an $O(\log n)$ approximation, with a complexity matching the long standing $O(\delta)$ bound for MIS (theorem 0.5.3). Furthermore, I have shown a simple, deterministic algorithm to compute a minimal DS (theorem 0.5.4).

Minimum Connected Dominating Sets: I have contributed to filling the gap in MCDS research, by showing an algorithm for Minimum DS can be adapted in a non-trivial way to maintain an $O(\log n)$ approximation of the MCDS in general dynamic graphs (theorem 0.5.5).

Dynamic Clustering: I have presented the first work that studies fully-dynamic algorithms, which simultaneously takes approximation guarantee, consistency, and update time into consideration. This led to a fully-dynamic algorithm for the facility location problem, with a constant factor approximation, a constant number of changes to the clustering at each time step, and $O(\log n)$ update time. The algorithm was extended for facility location to the k-median and k-means problem (theorem 0.5.6). This is the first non-trivial solution to these problems, resulting in time bounds significantly better than the naive approach for a larger range of k. In an empirical analysis on three real-world data sets, our algorithm outperformed the simple-minded solutions by at least three orders of magnitude.

Machine Learning and Data Analysis

Authorship Verification: I present a Siamese Network to detect ghost writing in high school, which, to the best of my knowledge, is the

first of its kind. The model had an accuracy of 87.5 percent, and an AUC score of 0.947 on a balanced data set.

Writing Style Development in High School

Using a Siamese Network, I have shown the ability to identify patterns in "error" distribution for individual authors. This has the potential to be used by teachers to improve their teaching, as the model identifies optimal and suboptimal developments.

Sequence Modelling for Analyzing Student Interaction with Eudcational Systems

Assuming that each individual student session is generated by a single Markov chain, we analyzed 1.08 million student sessions. By repeatedly updating priors and reassigning sequences, we reached a level where less than 5 percent change prior (a modified k++ algorithm for this setting). By using the elbow method we found a strong indication, that the right numbers of clusters/priors were between 6 and ten. Analyzing six chains, we found one to be suboptimal, indicating that the students found questions to be either too easy, too hard, or that they lack training in the subject.

Tracking Behavioral Patterns among Students in an Online Educational System

Using Non-negative Matrix Factorization I studied primary school data from Clio Online over a period of two years. The clustering was only made once. I identified three optimal and two suboptimal behaviors related to subject and performance. This lead to the identification of the areas where Clio Online was a great benefit to the students (language, quizzes), and where it was not (working with Clio Online at home).

BIBLIOGRAPHY

[AHHR18] Anders Aamand, Niklas Hjuler, Jacob Holm, and Eva Rotenberg. "One-Way Trail Orientations." In: *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*. 2018, 6:1–6:13.

[AS18] Raghavendra Addanki and Barna Saha. "Fully Dynamic Set Cover–Improved and Simple." In: *arXiv preprint arXiv:1804.03197* (2018).

[ANSW17] S. Ahmadian, A. Norouzi-Fard, O. Svensson, and J. Ward. "Better Guarantees for k-Means and Euclidean k-Median by Primal-Dual Algorithms." In: *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*. Oct. 2017, pp. 61–72.

[ABUH04] Aris Anagnostopoulos, Russell Bent, Eli Upfal, and Pascal Van Hentenryck. "A simple and deterministic competitive algorithm for online facility location." In: *Inf. Comput.* 194.2 (2004), pp. 175–202.

[AKFS03] Shlomo Argamon, Moshe Koppel, Jonathan Fine, and Anat Rachel Shimoni. "Gender, genre, and writing style in formal written texts." In: *TEXT* 23 (2003), pp. 321–346.

[AKPS09] Shlomo Argamon, Moshe Koppel, James W. Pennebaker, and Jonathan Schler. "Automatically Profiling the Author of an Anonymous Text." In: *Commun. ACM* 52.2 (Feb. 2009), pp. 119–123.

[AOSS18] Sepehr Assadi, Krzysztof Onak, Baruch Schieber, and Shay Solomon. "Fully dynamic maximal independent set with sublinear update time." In: *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*. 2018.

[AOSS] Sepehr Assadi, Krzysztof Onak, Baruch Schieber, and Shay Solomon. "Fully Dynamic Maximal Independent Set with Sublinear in <italic>n</italic> Update Time." In: *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 1919–1936.

[B D74] Paul B. Diederich. "Measuring Growth in English." In: (Jan. 1974).

[BLK17] Olivier Bachem, Mario Lucic, and Andreas Krause. "Distributed and Provably Good Seedings for k-Means in Constant Rounds." In: *Proceedings of the 34th International Conference on Machine Learning*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. International Convention Centre, Sydney, Australia: PMLR, June 2017, pp. 292–300.

[Bag15] Douglas Bagnall. "Author Identification using multi-headed Recurrent Neural Networks." In: *CLEF 2015 Evaluation Labs and Workshop – Working Notes Papers*. CEUR-WS.org, Sept. 2015.

[BDD+15] Alberto Bartoli, Alex Dagri, Andrea De Lorenzo, Eric Medvet, and Fabiano Tarlao. "An author verification approach based on differential features." In: *CEUR WORKSHOP PROCEEDINGS*. Vol. 1391. CEUR. 2015.

[BFH] Aaron Bernstein, Sebastian Forster, and Monika Henzinger. "A Deamortization Approach for Dynamic Spanner and Dynamic Maximal Matching." In: *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 1899–1918.

[BS16] Aaron Bernstein and Cliff Stein. "Faster fully dynamic matchings with small approximation ratios." In: *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics. 2016, pp. 692–711.

[BB97] Sivakumar R Bevan Das and V Bharghavan. "Routing in ad-hoc networks using a virtual backbone." In: *Proceedings of the 6th International Conference on Computer Communications and Networks (IC3N'97)*. 1997, pp. 1–20.

[BCHN18] Sayan Bhattacharya, Deeparnab Chakrabarty, Monika Henzinger, and Danupon Nanongkai. "Dynamic algorithms for graph coloring." In: *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics. 2018, pp. 1–20.

[BHI18] Sayan Bhattacharya, Monika Henzinger, and Giuseppe F Italiano. "Deterministic fully dynamic data structures for vertex cover and matching." In: *SIAM Journal on Computing* 47.3 (2018), pp. 859–887.

[BT80] Frank Boesch and Ralph Tindell. "Robbins's Theorem for Mixed Multigraphs." In: *The American Mathematical Monthly* 87.9 (1980), pp. 716–719.

[BMO+11] V. Braverman, A. Meyerson, R. Ostrovsky, A. Roytman, M. Shindler, and B. Tagiku. "Streaming k-means on Well-Clusterable Data." In: *SODA*. 2011, pp. 26–40.

[BFL+17] Vladimir Braverman, Gereon Frahling, Harry Lang, Christian Sohler, and Lin F. Yang. "Clustering High Dimensional Dynamic Data Streams." In: *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*. 2017, pp. 576–585.

[BLLM16] Vladimir Braverman, Harry Lang, Keith Levin, and Morteza Monemizadeh. "Clustering problems on sliding windows." In: *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics. 2016, pp. 1374–1390.

[BCOP04] Sergiy Butenko, Xiuzhen Cheng, Carlos A Oliveira, and Panos M Pardalos. "A new heuristic for the minimum connected dominating set problem on ad hoc wireless networks." In: *Recent developments in cooperative control and optimization*. Springer, 2004, pp. 61–73.

[CP04] Fazli Can and Jon M. Patton. "Change of Writing Style with Time." In: *Computers and the Humanities* 38.1 (Feb. 2004), pp. 61–82.

[CCFM04] Moses Charikar, Chandra Chekuri, Tomás Feder, and Rajeev Motwani. "Incremental Clustering and Dynamic Information Retrieval." In: *SIAM J. Comput.* 33.6 (2004), pp. 1417–1440.

[COP03] Moses Charikar, Liadan O'Callaghan, and Rina Panigrahy. "Better Streaming Algorithms for Clustering Problems." In: *Proceedings of the Thirty-fifth Annual ACM Symposium on Theory of Computing*. STOC '03. San Diego, CA, USA: ACM, 2003, pp. 30–39.

[CHL+03] Xiuzhen Cheng, Xiao Huang, Deying Li, Weili Wu, and Ding-Zhu Du. "A polynomial-time approximation scheme for the minimum-connected dominating set in ad hoc wireless networks." In: *Networks: An International Journal* 42.4 (2003), pp. 202–208.

[CS14] Jun Chu and Sargur Srihari. "Writer Identification Using a Deep Neural Network." In: *Proceedings of the 2014 Indian Conference on Computer Vision Graphics and Image Processing*. ICVGIP '14. Bangalore, India: ACM, 2014, 31:1–31:7.

[Chv79] V. Chvatal. "A Greedy Heuristic for the Set-Covering Problem." In: *Mathematics of Operations Research* 4.3 (1979), pp. 233–235.

[CCMS18] Marek Cygan, Artur Czumaj, Marcin Mucha, and Piotr Sankowski. "Online Facility Location with Deletions." In: *26th Annual European Symposium on Algorithms, ESA 2018, August 20-22, 2018, Helsinki, Finland*. 2018, 21:1–21:15.

[CLMS13] Artur Czumaj, Christiane Lammersen, Morteza Monemizadeh, and Christian Sohler. "(1+ ϵ)-approximation for facility location in data streams." In: *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*. 2013, pp. 1710–1728.

[DI04] Camil Demetrescu and Giuseppe F. Italiano. "A New Approach to Dynamic All Pairs Shortest Paths." In: *J. ACM* 51.6 (2004), pp. 968–992.

[DW12] Ding-Zhu Du and Peng-Jun Wan. *Connected dominating set: theory and applications*. Vol. 77. Springer Science & Business Media, 2012.

[FKD16] Louis Faucon, Lukasz Kidzinski, and Pierre Dillenbourg. "Semi-Markov model for simulating MOOC students." In: *Proceedings of the 9th International Conference on Educational Data Mining (EDM)*. International Educational Data Mining Society (IEDMS), 2016, pp. 358–363.

[Fei98] Uriel Feige. "A threshold of ln n for approximating set cover." In: *Journal of the ACM (JACM)* 45.4 (1998), pp. 634–652.

[Fot06] Dimitris Fotakis. "Incremental algorithms for Facility Location and *k*-Median." In: *Theor. Comput. Sci.* 361.2-3 (2006), pp. 275–313.

[Fot07] Dimitris Fotakis. "A primal-dual algorithm for online non-uniform facility location." In: *J. Discrete Algorithms* 5.1 (2007), pp. 141–148.

[Fot08] Dimitris Fotakis. "On the Competitive Ratio for Online Facility Location." In: *Algorithmica* 50.1 (2008), pp. 1–57.

[FS05] Gereon Frahling and Christian Sohler. "Coresets in dynamic geometric data streams." In: *Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC)*. 2005, pp. 209–217.

[FBPW11] Marjorie Freedman, Alex Baron, Vasin Punyakanok, and Ralph Weischedel. "Language Use: What Can It Tell Us?" In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: Short Papers - Volume 2*. HLT '11. Portland, Oregon: Association for Computational Linguistics, 2011, pp. 341–345.

[GJ79] M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[GRD+16] Ben U. Gelman, Matt Revelle, Carlotta Domeniconi, Kalyan Veeramachaneni, and Aditya Johri. "Acting the Same Differently: A Cross-Course Comparison of User Behavior in MOOCs." In: *Proceedings of the 9th International Conference on Educational Data Mining (EDM)*. International Educational Data Mining Society (IEDMS), 2016, pp. 376–381.

[Gon85] Teofilo F. Gonzalez. "Clustering to minimize the maximum intercluster distance." In: *Theoretical Computer Science* 38 (1985), pp. 293–306.

[GHL18] Gramoz Goranci, Monika Henzinger, and Dariusz Leniowski. "A Tree Structure For Dynamic Facility Location." In: *26th Annual European Symposium on Algorithms, ESA 2018, August 20-22, 2018, Helsinki, Finland*. 2018, 39:1–39:13.

[GK98] Sudipto Guha and Samir Khuller. "Approximation algorithms for connected dominating sets." In: *Algorithmica* 20.4 (1998), pp. 374–387.

[GMM13] Leonidas Guibas, Nikola Milosavljević, and Arik Motskin. "Connected dominating sets on dynamic geometric graphs." In: *Computational Geometry* 46.2 (2013), pp. 160–172.

[GKKP17] Anupam Gupta, Ravishankar Krishnaswamy, Amit Kumar, and Debmalya Panigrahi. "Online and dynamic algorithms for set cover." In: *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*. ACM. 2017, pp. 537–550.

[GK18] Manoj Gupta and Shahbaz Khan. "Simple dynamic algorithms for Maximal Independent Set and other problems." In: *arXiv preprint arXiv:1804.01823* (2018).

[GP13] Manoj Gupta and Richard Peng. "Fully dynamic (1+ e)-approximate matchings." In: *Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on*. IEEE. 2013, pp. 548–557.

[HHH+17a] Christian Hansen, Casper Hansen, Niklas Hjuler, Stephen Alstrup, and Christina Lioma. "Sequence Modelling For Analysing Student Interaction with Educational Systems." In: *Proceedings of the 10th International Conference on Educational Data Mining, EDM 2017, Wuhan, Hubei, China, June 25-28, 2017*. 2017.

[HHH+17b] Christian Hansen, Casper Hansen, Niklas Hjuler, Stephen Alstrup, and Christina Lioma. "Sequence Modelling For Analysing Student Interaction with Educational Systems." In: *Proceedings of the 10th International Conference on Educational Data Mining (EDM)*. International Educational Data Mining Society (IEDMS), 2017, pp. 232–237.

[HLLA14] Niels Dalum Hansen, Christina Lioma, Birger Larsen, and Stephen Alstrup. "Temporal context for authorship attribution: a study of Danish secondary schools." In: *Multidisciplinary information retrieval*. Springer, 2014, pp. 22–40.

[Hås99] Johan Håstad. "Clique is hard to approximate withinn 1- ε." In: *Acta Mathematica* 182.1 (1999), pp. 105–142.

[HLM17] Monika Henzinger, Dariusz Leniowski, and Claire Mathieu. "Dynamic Clustering to Minimize the Sum of Radii." In: *25th Annual European Symposium on Algorithms, ESA 2017, September 4-6, 2017, Vienna, Austria*. 2017, 48:1–48:10.

[HIPS19] Niklas Hjuler, Giuseppe F. Italiano, Nikos Parotsidis, and David Saulpic. "Dominating Sets and Connected Dominating Sets in Dynamic Graphs." In: *36th International Symposium on Theoretical Aspects of Computer Science (STACS 2019)*. Ed. by Rolf Niedermeier and Christophe Paul. Vol. 126. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019, 35:1–35:17.

[HLT01] Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. "Poly-logarithmic Deterministic Fully-dynamic Algorithms for Connectivity, Minimum Spanning Tree, 2-edge, and Biconnectivity." In: *J. ACM* 48.4 (2001), pp. 723–760.

[HGS18] T.-H. Hubert Chan, Arnaud Guerqin, and Mauro Sozio. "Fully Dynamic k-Center Clustering." In: *Proceedings of the 2018 World Wide Web Conference on World Wide Web, WWW 2018, Lyon, France, April 23-27, 2018*. 2018, pp. 579–587.

[HMW+16] Stephen Hutt, Caitlin Mills, Shelby White, Patrick J. Donnelly, and Sidney K. D'Mello. "The Eyes Have It: Gaze-based Detection of Mind Wandering during Learning with an Intelligent Tutoring System." In: *Proceedings of the 9th International Conference on Educational Data Mining (EDM)*. International Educational Data Mining Society (IEDMS), 2016, pp. 86–93.

[Ind04] Piotr Indyk. "Algorithms for dynamic geometric problems over data streams." In: *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*. 2004, pp. 373–380.

[JRS02] Lujun Jia, Rajmohan Rajaraman, and Torsten Suel. "An efficient distributed algorithm for constructing small dominating sets." In: *Distributed Computing* 15.4 (2002), pp. 193–205.

[Kan92] Viggo Kann. "On the approximability of NP-complete optimization problems." PhD thesis. Royal Institute of Technology Stockholm, 1992.

[KKSG16] Severin Klingler, Tanja Käser, Barbara Solenthaler, and Markus Gross. "Temporally Coherent Clustering of Student Data." In: *Proceedings of the 9th International Conference on Educational Data Mining (EDM)*. International Educational Data Mining Society (IEDMS), 2016, pp. 102–109.

[KSZ05] Moshe Koppel, Jonathan Schler, and Kfir Zigdon. "Determining an Author's Native Language by Mining a Text for Errors." In: *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*. KDD '05. Chicago, Illinois, USA: ACM, 2005, pp. 624–628.

[KW05] Fabian Kuhn and Roger Wattenhofer. "Constant-time distributed dominating set approximation." In: *Distributed Computing* 17.4 (2005), pp. 303–310.

[LS08] Christiane Lammersen and Christian Sohler. "Facility Location in Dynamic Geometric Data Streams." In: *Algorithms - ESA 2008, 16th Annual European Symposium, Karlsruhe, Germany, September 15-17, 2008. Proceedings*. 2008, pp. 660–671.

[Lan18] Harry Lang. "Online Facility Location against a t-Bounded Adversary." In: *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*. 2018, pp. 1002–1014.

[LV17] Silvio Lattanzi and Sergei Vassilvitskii. "Consistent k-Clustering." In: *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*. 2017, pp. 1975–1984.

[Li13] Shi Li. "A 1.488 approximation algorithm for the uncapacitated facility location problem." In: *Information and Computation* 222 (2013). 38th International Colloquium

on Automata, Languages and Programming (ICALP 2011), pp. 45–58.

[LHA18] Stephan Lorenzen, Niklas Hjuler, and Stephen Alstrup. "Tracking Behavioral Patterns among Students in an Online Educational System." In: *Proceedings of the 11th International Conference on Educational Data Mining, EDM 2018, Buffalo, NY, USA, July 15-18, 2018*. 2018.

[LHA19] Stephan Lorenzen, Niklas Hjuler, and Stephen Alstrup. "Investigating Writing Style Development in High School." In: *Proceedings of the 12th International Conference on Educational Data Mining, EDM 2019, Montréal, Canada, July 2-5, 2019*. 2019.

[LN13] Annie Louis and Ani Nenkova. "What Makes Writing Great? First Experiments on Article Quality Prediction in the Science Journalism Domain." In: *Transactions of the Association for Computational Linguistics* 1 (2013), pp. 341–352.

[MP04] Ramgopal R. Mettu and C. Greg Plaxton. "Optimal Time Bounds for Approximate Clustering." In: *Machine Learning* 56.1 (July 2004), pp. 35–60.

[Mey01] Adam Meyerson. "Online Facility Location." In: *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*. 2001, pp. 426–431.

[MS18] Alexander Munteanu and Chris Schwiegelshohn. "Coresets-Methods and History: A Theoreticians Design Pattern for Approximation and Streaming Algorithms." In: *KI* 32.1 (2018), pp. 37–53.

[NSW17] D. Nanongkai, T. Saranurak, and C. Wulff-Nilsen. "Dynamic Minimum Spanning Forest with Subpolynomial Worst-Case Update Time." In: *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*. Oct. 2017, pp. 950–961.

[NS16] Ofer Neiman and Shay Solomon. "Simple deterministic algorithms for fully dynamic maximal matching." In: *ACM Transactions on Algorithms (TALG)* 12.1 (2016), p. 7.

[PDV11] Claudia Peersman, Walter Daelemans, and Leona Van Vaerenbergh. "Predicting Age and Gender in Online Social Networks." In: *Proceedings of the 3rd International Workshop on Search and Mining User-generated Contents*. SMUC '11. Glasgow, Scotland, UK: ACM, 2011, pp. 37–44.

[PN08] Emily Pitler and Ani Nenkova. "Revisiting Readability: A Unified Framework for Predicting Text Quality." In: *EMNLP 2008*. 2008.

[QHZ18a] Chen Qian, Tianchang He, and Rao Zhang. "Deep Learning based Authorship Identification." In: (2018). report, Stanford University.

[QHZ18b] Chen Qian, Tianchang He, and Rao Zhang. *Deep Learning based Authorship Identification*. Accessed April 2019. Stanford University, 2018.

[Rob39] H. E. Robbins. "A Theorem on Graphs, with an Application to a Problem of Traffic Control." In: *The American Mathematical Monthly* 46.5 (1939), pp. 281–283.

[SW79] E Sampathkumar and HB Walikar. "The connected domination number of a graph." In: *J. Math. Phys* (1979).

[SBSV13] Kosgi Santosh, Romil Bansal, Mihir Shekhar, and Vasudeva Varma. "Author Profiling: Predicting Age and Gender from Blogs - Notebook for PAN at CLEF 2013." In: *CLEF*. 2013, p. 10.

[SKA02] Anat Rachel Shimoni, Moshe Koppel, and Shlomo Argamon. "Automatically Categorizing Written Texts by Author Gender." In: *Literary and Linguistic Computing* 17.4 (Nov. 2002), pp. 401–412. eprint: http://oup.prod.sis.lan/dsh/article-pdf/17/4/401/3345463/170401.pdf.

[SWM11] M. Shindler, A. Wong, and A. Meyerson. "Fast and Accurate k-means For Large Datasets." In: *NIPS*. 2011, pp. 2375–2383.

[Sol16] S. Solomon. "Fully Dynamic Maximal Matching in Constant Update Time." In: *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*. Oct. 2016, pp. 325–334.

[Spa01] V. Spandel. *Creating Writers: Through 6-trait Writing Assessment and Instruction*. Longman, 2001.

[Sta09] Efstathios Stamatatos. "A Survey of Modern Authorship Attribution Methods." In: *J. Am. Soc. Inf. Sci. Technol.* 60.3 (Mar. 2009), pp. 538–556.

[SSL+19a] Magnus Stavngaard, August Sørensen, Stephan Lorenzen, Niklas Hjuler, and Stephen Alstrup. "Detecting Ghostwriters in High Schools." In: *27th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*. 2019.

[SSL+19b] Magnus Stavngaard, August Sørensen, Stephan Lorenzen, Niklas Hjuler, and Stephen Alstrup. "Detecting Ghostwriters in High Schools." In: *CoRR* abs/1906.01635 (2019). arXiv: 1906.01635.

[WL99] Jie Wu and Hailan Li. "On calculating connected dominating set for efficient routing in ad hoc wireless networks." In: *Proceedings of the 3rd international workshop on Discrete algorithms and methods for mobile computing and communications*. ACM. 1999, pp. 7–14.

[YCL+19] Yuting Yang, Juan Cao, Mingyan Lu, Jintao Li, and Chia-Wen Lin. "How to Write High-quality News on Social Network? Predicting News Quality by Mining Writing Style." In: *CoRR* abs/1902.00750 (2019). arXiv: 1902.00750.

Dominating Sets and Connected Dominating Sets in Dynamic Graphs

Niklas Hjuler
University of Copenhagen, Denmark
hjuler@di.ku.dk

Giuseppe F. Italiano
LUISS University, Rome, Italy
gitaliano@luiss.it

Nikos Parotsidis
University of Rome Tor Vergata, Italy
nikos.parotsidis@uniroma2.it

David Saulpic
ENS Paris, France
david.saulpic@ens.fr

Abstract

In this paper we study the dynamic versions of two basic graph problems: Minimum Dominating Set and its variant Minimum Connected Dominating Set. For those two problems, we present algorithms that maintain a solution under edge insertions and edge deletions in time $O(\Delta \cdot \text{polylog } n)$ per update, where Δ is the maximum vertex degree in the graph. In both cases, we achieve an approximation ratio of $O(\log n)$, which is optimal up to a constant factor (under the assumption that $P \neq NP$). Although those two problems have been widely studied in the static and in the distributed settings, to the best of our knowledge we are the first to present efficient algorithms in the dynamic setting.

As a further application of our approach, we also present an algorithm that maintains a Minimal Dominating Set in $O(min(\Delta, \sqrt{m}))$ per update.

2012 ACM Subject Classification Theory of computation → Dynamic graph algorithms

Keywords and phrases Dominating Set, Connected Dominating Set, Dynamic Graph Algorithms

Digital Object Identifier 10.4230/LIPIcs.STACS.2019.35

Funding This work was done while Niklas Hjuler and David Saulpic were visiting University of Rome Tor Vergata.

1 Introduction

The study of dynamic graph algorithms is a classical area in algorithmic research and has been thoroughly investigated in the past decades. Maintaining a solution of a graph problem in the case where the underlying graph changes dynamically over time is a big challenge in the design of efficient and practical algorithms. Indeed, in several applications, due to the dynamic nature of today's data, it is not sufficient to compute a solution to a graph problem only once and for all: often, it is necessary to *maintain* a solution efficiently while the input graph is undergoing a sequence of dynamic updates. More precisely, a *dynamic graph* is a sequence of graphs $G_0, ..., G_M$ on n nodes and such that G_{i+1} is obtained from G_i by adding or removing a single edge. The natural first barrier, in the study of dynamic algorithms, is to design algorithms that are able to maintain a solution for the problem at hand after each update faster than recomputing the solution from scratch. Many dynamic graph problems such as minimum spanning forests (see e.g. [22, 26]), shortest paths [12], matching [4, 27, 30] or coloring [7] have been extensively studied in the literature, and very efficient algorithms are known for those problems. Recently, a lot of attention has been devoted to the Maximal

Independent Set problem (MIS). In this problem, one wishes to find a maximal set of vertices that do not share any edge ("maximal" meaning that it is not possible to add any vertex without violating this property). Until very recently, the best known update bound on the complexity to maintain a MIS was a simple $O(\Delta)$ algorithm, where Δ is an upper bound on the degree of vertices in the graph. This bound was first broken by Assadi et al. [2] who gave a $O(m^{3/4})$ algorithm, then by Gupta and Khan [19] improved the update bound to $O(m^{2/3})$. Very recently, using randomization, Assadi et al. [3] presented an amortized fully-dynamic algorithm with an expected $\widetilde{O}(n^{1/2})$-time bound per update.

The MIS problem is closely related to the Dominating Set (DS) problem: given a graph $G = (V, E)$ the DS problems is to find a subset of vertices $\mathcal{D} \subseteq V$ such that every vertex in G is adjacent to $\mathcal{D}$ (or *dominated* by $\mathcal{D}$). Indeed, a MIS is also a Minimal DS: the fact that it is not possible to add a vertex without breaking the independence property implies that every vertex is adjacent to the MIS, so this must be also a DS; at the same time, it is not possible to remove a vertex since that vertex is no longer dominated. Thus, to find a Minimal DS one can simply find a MIS: this gives immediately a deterministic $O(m^{2/3})$ [19] bound and a randomized $\widetilde{O}(n^{1/2})$ [3] one. However, while it is known that is hard to approximate Maximum Independent Set[1] within a factor $n^{1-\epsilon}$ for every $\epsilon > 0$[21], a simple greedy approach achieves a $O(\log n)$-approximation for Minimum DS [11].

In recent years, there has been a lot of work on designing dynamic graph algorithms for maintaining approximate solutions to several problems. A notable example is matching, where for different approximations there exist different algorithms (see e.g., [4, 5, 27, 20, 8, 30]). This raises the natural question on whether there exists a dynamic algorithm capable of maintaining an approximation to Minimum DS, and even better a $O(\log n)$ approximation. In this paper, we answer this question affirmatively by presenting an algorithm that achieves a $O(\log n)$ approximation, with a complexity matching the long standing $O(\Delta)$ bound for MIS. Moreover, if one is interested in finding a DS faster, we present a very simple *deterministic* $O(m^{1/2})$ algorithm to compute a Minimal DS, improving the $O(m^{2/3})$ bound coming from MIS. We believe these are important steps towards understanding the complexity of the problem. Those two results are stated below.

▶ **Theorem 1.** *Starting from a graph with n vertices, a $O(\log n)$ approximation of* Minimum Dominating Set *can be maintained over any sequence of $\Omega(n)$ edge insertions and deletions in $O(\Delta \log n)$ amortized time per update, where Δ is the maximum degree of the graph over the sequence of updates.*

▶ **Theorem 2.** *Starting from a graph with n (fixed) vertices, a* Minimal Dominating Set *can be deterministically maintained over any sequence of edge insertions and deletions in $O(\sqrt{m})$ amortized time per update, where m is an upper bound on the number of edges in the graph.*

We also study the Minimum Connected Dominating Set problem (MCDS), which adds the constraint that the graph induced by the DS $\mathcal{D}$ must be connected. This problem was first introduced by Sampathkumar and Walikar [28] and arises in several applications. The most noteworthy is its use as a *backbone* in routing protocols: it allows to limit the number of packet transmissions, by sending packets only along the backbone rather than throughout the whole network. Du and Wan's book [13] summarizes the knowledge about

[1] It is not possible to find a polynomial-time algorithm that finds a $n^{1-\epsilon}$-approximation to Maximum Independent Set under the assumption NP $\neq$ ZPP

MCDS. A special class of graphs is geometric graphs, where vertices are points in the plane, and two vertices are adjacent if they fall within a certain range (say, their distance is at most 1). This can model wifi transmissions, and the dynamic MCDS had been studied in this setting: a polynomial-time approximation scheme is known [10], and Guibas et al. [17] show how to maintain a constant-factor approximation with polylogarithmic update time. While geometric graphs model problems linked to wifi transmissions, the general graph setting can be also seen as a model for wired networks. However, no work about dynamic MCDS is known in this setting: the static case is well studied, with a greedy algorithm developed by Guha and Keller [16] that achieves an approximation factor $O(\ln \Delta)$. They also show a lower bound matching their complexity, together with their approximation factor. MCDS had also been thoroughly studied in the distributed setting (see e.g. a heuristic to find a Minimal CDS in [9], another one that sends $O(\Delta n)$ messages and has a time complexity at each vertex $O(\Delta^2)$ [31] or a $3 \log n$ approximation that runs in $O(\gamma)$ rounds where γ is the size of the CDS found, with time complexity $O(\gamma\Delta^2 + n)$ and message complexity $O(n\Delta\gamma + m + n \log n)$ [6]). Despite all this work, no results are known in the dynamic graph setting. As another application of our approach, we contribute to filling this gap in the research line of MCDS. In particular, in this paper we show how our algorithm for Minimum DS can be adapted in a non-trivial way to maintain a $O(\log n)$ approximation of the MCDS in general dynamic graphs.

▶ **Theorem 3.** *Starting from a graph with n vertices, a $O(\log n)$ approximation of* Minimum Connected Dominating Set *can be maintained over any sequence of $\Omega(n)$ edge insertions and deletions in $\widetilde{O}(\Delta)$ amortized time per update.*

We further show how to maintain independently a Dominating Set $\mathcal{D}$ and a set of vertices $\mathcal{C}$ such that the induced subgraphs on the vertices $\mathcal{C} \cup \mathcal{D}$ is connected. The set $\mathcal{C}$ has the additional property that $|\mathcal{C}| \leq 2|\mathcal{D}|$, such that $|\mathcal{C} \cup \mathcal{D}| = O(|\mathcal{D}|)$. If $\mathcal{D}$ is a α-approximation of Minimum DS, this gives a $O(\alpha)$ approximation for MCDS.

Further Related Work

It is well known that finding a Minimum DS is NP-hard [15]. It is therefore natural to look for approximation algorithms for this problem. Unfortunately, it is also NP-hard to find a $c \log n$ approximation, for any $0 < c < 1$ [14]. This bound is tight, since there is a simple greedy algorithm matching this bound [11]. Minimum DS had been studied extensively in distributed computing: an algorithm which runs in $O(\log n \log \Delta)$ rounds finds a $O(\log n)$ approximation with high probability [23] and an algorithm with constant number of rounds achieves a non-trivial approximation[25].

The DS problem is closely related to the Set Cover problem: the two problems are equivalents under L-reduction [24]. However, Set Cover was studied in the dynamic setting [18, 1], but with different kinds of updates: instead of edges being inserted or deleted (which would represent new elements in the sets according to the L-reduction), new elements are being added to the cover (which would be new vertices in DS).

Outline. The rest of the paper is organized as follows. First, we present an algorithm for Minimum DS, which will be used later on also for MCDS: we start by a $\widetilde{O}(n)$ algorithm, and then show how to overcome its bottleneck in order to achieve a $\widetilde{O}(\Delta)$ complexity. Finally, we present our $O(\sqrt{m})$ algorithm for Minimal DS.

2 A $O(\log n)$ approximation of Minimum Dominating Set in $O(\Delta \log n)$ time per update

This section aims at proving Theorem 1. Following a reduction from Set Cover, minimum DS is NP-hard to approximate within a factor $\log n$ [14]. Here we present a matching upper bound (up to a constant factor), in the dynamic setting. Our algorithm relies heavily on the clever set cover algorithm by Gupta et al. [18]. While in the static setting Set Cover is equivalent to minimum DS, in the dynamic setting these two problems are different. More precisely, in the dynamic Set Cover problem one is asked to cover a set of points S (called the universe) with a given family of sets F, while the set S is changing dynamically. To draw the parallel with DS, in the latter the set S is the set of vertices of the graph (which does not change) and for every vertex the set of its neighbors is in F. The dynamic part concerns therefore F, and not the universe S.

Gupta et al. present an $O(\log n)$-approximation for dynamic Set Cover problem: in what follows, we show how to adapt their algorithm to the DS case, with an update time of $O(\Delta \log n)$. As in [18], the approach easily adapts to the weighted case. Unfortunately, this cannot be generalized to MCDS, therefore we do not consider this property of the algorithm. The following definitions are partly adapted from [18].

2.1 Preliminaries

For a vertex v, let $N(v)$ be the set of its neighbors, including v. The algorithm maintains a solution S_t at time t such that an element of S_t is a pair composed of

- a dominant vertex v
- a set $\mathrm{Dom}(v) \subseteq N(v)$, which are the vertices that are dominated by v. We call $|\mathrm{Dom}(v)|$ the *cardinality* of the pair.

We call a *dominating pair* an element of S_t. The algorithm requires that multiple copies of a vertex can appear as the dominant vertex of a pair. However, each vertex is exactly in one $\mathrm{Dom}(v)$. The solution to the DS problem is composed of all vertices that appear as dominant vertices of a pair. Since each vertex is in exactly one $\mathrm{Dom}(v)$, each vertex is dominated and therefore the set of dominants is a valid solution to the DS problem.

The dominating pairs are placed into *levels* according to their cardinality: the level l is defined by a range $R_l := [2^{l-10}, 2^l]$, and each pair $(v, \mathrm{Dom}(v))$ is placed at an appropriate level l such that $|\mathrm{Dom}(v)| \in R_l$. In that case, elements of $\mathrm{Dom}(v)$ are said to be dominated at level l; we denote by V_l the set of all vertices dominated at level l. We say that an assignment of levels is valid if it respects the constraint $|\mathrm{Dom}(v)| \in R_l$. This allows us to define the notion of *Stability*:

- *stable solution*: A solution S_t is stable if there is no vertex v and level l such that $|N(v) \cap V_l| > 2^l$; in other words, it is not possible to introduce a new vertex in the solution to dominate some vertices at level l such that the resulting dominating pair could be at level strictly greater that l.

The algorithm will dynamically maintain a stable solution S_t, with a valid assignment of levels. Note that the ranges R_l overlap: this gives some slack to the algorithm, which allows enough flexibility to prevent too many changes while our algorithm maintains a valid solution.

2.2 The algorithm

The main part of the algorithm is the function STABILIZE, which restores the stability at the end of every update. The function is the following (see [18]):

> STABILIZE. As long as a vertex v violates the stability condition at level l, do the following: Add the pair $(v, N(v) \cap V_l)$ to the *lowest* possible level j (i.e., the lowest level such that $|N(v) \cap V_l| \in R_j$); Remove the elements of $N(v) \cap V_l$ from the set of their former covering pair: if it gets empty, remove the pair from the solution. Otherwise, if the cardinality of such a pair goes below 2^{l-10}, put it at the *highest* possible level.

Edge addition. When a new edge (u, v) is added to the graph, one just need to ensure that the solution remains stable, and thus the algorithm runs STABILIZE.

Edge deletion. When an edge (u, v) is removed from the graph, we proceed as follows. If neither u nor v dominates the other endpoint, the solution remains valid and stable, and nothing needs to be done. Otherwise, assume without loss of generality that v dominates u. Then:

- Remove u from $\mathrm{Dom}(v)$
- Add the pair $(u, \mathrm{Dom}_u = \{u\})$ to the solution with level 1
- Run STABILIZE

Correctness. All the nodes of the graph are dominated at every time. Indeed, STABILIZE does not make any node undominated and if a vertex is not dominated after an edge removal, the algorithm simply adds it to the solution. Therefore, the solution S_t maintained by the algorithm is a valid one.

2.3 Analysis

Approximation ratio. We use the following lemma by Gupta et al. [18] to bound the cost of a stable solution.

▶ **Lemma 4** (Lemma 2.1 in [18]). *The number of sets at one level in any stable solution is at most* $2^{10} \cdot OPT$.

Since for every dominating pair $(v, \mathrm{Dom}(v))$ we have that $1 \leq |\mathrm{Dom}(v)| \leq n$, there are only $\log n$ levels that can contain a set. The total cost of a stable solution is therefore $O(\log n \cdot \mathrm{OPT})$.

A token scheme to bound the number of updates. Unfortunately, the analysis of Gupta et al. cannot be applied directly to the case of DS, due to the different nature of the updates. However, we can build upon their analysis, as follows. We first bound the number of vertices that change level, and then explain how to implement a level change so that it costs $O(\Delta)$. We prove the following lemma by using a token argument.

▶ **Lemma 5.** *After k updates of the algorithm, at most $O(k \log n + n \log n)$ elements have changed levels.*

Proof. We use the following token scheme, where each vertex pays one token for each level change. In the beginning, we give $2\log n$ tokens to every vertex. If a vertex is undominated after an edge removal, we give $2\log n$ new tokens to this vertex. Since at most one vertex gets undominated for each edge deletion, the total number of tokens given after k updates is $O(k\log n + n\log n)$. To prove the lemma, we need to show that at any time each vertex has always a positive amount of tokens. We adapt the proof of Gupta et al. to show the following invariant:

▶ **Invariant 1.** *Every vertex at level l has more than $2(\log n - l)$ tokens.*

When a vertex is moved to a higher level, it pays one token for the cost of moving. It also saves one token, and gives it to an "emergency fund" of its former covering pair. Each pair has therefore a fund of tokens that can be used when the pair has to be moved to a lower level.

When the pair $(v, \mathrm{Dom}(v))$ has to be moved from level l to level $l-j$, it means that a lot of vertices have left $\mathrm{Dom}(v)$ and that the tokens they gave to the pair can be used to pay for the operation. Formally, we want to pay one token for every vertex in $\mathrm{Dom}(v)$ for its level change, but we also want to restore the invariant. We need therefore $2j+1$ tokens for each vertex of $\mathrm{Dom}(v)$. Since the pair can be moved to level $l-j$, this means that $|\mathrm{Dom}(v)| < 2^{l-j}$. Since a new pair is moved to the lowest possible level, this pair could not be at level $l-1$, which implies that $|\mathrm{Dom}_{init}(v)| > 2^{l-1}$ where $\mathrm{Dom}_{init}(v)$ is the set $\mathrm{Dom}(v)$ at the time where it was created. Moreover, each of the vertices that left gave one token: the amount of tokens usable is therefore bigger than $2^{l-1} - 2^{l-j}$. Thus we want to prove that $2^{l-1} - 2^{l-j} \geq (2j+1)\cdot|\mathrm{Dom}(v)|$. It is enough to have $2^{l-1} - 2^{l-j} \geq 3\cdot(2j+1)2^{l-j}$, i.e. to have $2^{j-1} - 1 \geq 3(2j+1)$. But since the pair was moved to level $l-j$, it means that $|\mathrm{Dom}(v)| > 2^{l-j-1}$ and $|\mathrm{Dom}(v)| < 2^{l-10}$: putting these two equations together gives $j > 9$, which ensures that $2^{j-1} - 1 \geq 3(2j+1)$ and concludes the proof. ◀

As the following corollary shows, we can bound the number of changes to $\mathcal{D}$ to $O(\log n)$ amortized. This property will be useful in Section 3.

▶ **Corollary 6.** *After k updates of the algorithm, at most $O(k\log n + n\log n)$ vertices can be added to or removed from $\mathcal{D}$.*

Proof. Whenever a vertex is added to or removed from $\mathcal{D}$, its level is changed. Lemma 5 gives the corresponding bound. ◀

We now turn to the implementation of the function Stabilize. As shown in the next lemma, we implemented so that its cost is $O(\Delta)$ for each element that changes level.

▶ **Lemma 7.** *A stable solution can be maintained in $O(\Delta\log n)$ amortized time per update.*

Proof. For all vertices v and all levels l, the algorithm maintains the set $N(v)\cap V_l$ and its cardinality. Every time a vertex changes its level, it has to inform all its neighbors: this can be done in $O(\Delta)$. When an edge (u,v) is added to or removed from the solution, the algorithm updates the sets $N(v)\cap V_{l_u}$ and $N(u)\cap V_{l_v}$, where l_u and l_v are the levels of u and v, respectively.

During a call to Stabilize, the algorithm maintains also a list of vertices that may have to be added to restore the stability: for a vertex v and level l, every time that $N(v)\cap V_l$ changes, if the new cardinality violates the stability, we add v to this list in constant time. The algorithm processes the list vertex by vertex: it checks that the current vertex still needs to be added to the solution, and add it if necessary.

Since we pay $O(\Delta)$ per level change and there are $O(\log n)$ amortized changes, the amortized complexity of each update is $O(\Delta\log n)$. ◀

Since a stable solution gives a $O(\log n)$ approximation to minimum DS, Lemmas 4 and 7 yield the proof of Theorem 1: a $O(\log n)$ approximation of Minimum Dominating Set can be maintained in $O(\Delta \log n)$ amortized time per update.

3 A $O(\log n)$ Approximation for Minimum Connected Dominating Set in $\widetilde{O}(n)$ per update

A possible way to compute a Connected DS is simply to find a DS and add a set of vertices to make it connected. Section 2 gives an algorithm to maintain an approximation of the Minimum DS: we will use it as a black box (and refer to it as the "black box"), and show how to make its solution connected without losing the approximation guarantee. If the original graph is not connected, the algorithm finds a CDS in every connected component: we focus in the following on a single of these components. Let $\mathcal{D}$ be the DS maintained, and $\mathcal{C}$ be a set of vertices such that $\mathcal{C} \cup \mathcal{D}$ is connected and $\mathcal{C}$ is minimal for that property. The minimality of $\mathcal{C}$ will ensure that $|\mathcal{C}| \leq 2|\mathcal{D}|$: since $\mathcal{D}$ is a $O(\log n)$ approximation of MDS, this leads to a $O(\log n)$ approximation for MCDS. Note that the vertices of $\mathcal{C}$ are not used for domination: $\mathcal{C} \cup \mathcal{D}$ is therefore not minimal, but still an approximation of minimum.

Overall, we will apply the following charging scheme to amortize the total running time. The main observation is that although a lot of vertices can be deleted to restore the minimality of $\mathcal{C}$, only a few can be added at every step. We thus give enough potential to a vertex whenever it is added into $\mathcal{C}$ and whenever its neighborhood changes, so that at the time of its removal from $\mathcal{C}$ it has accumulated enough potential for scanning its entire neighborhood. After an edge deletion we might have to restore the connectivity requirement. We do that by adding at most 2 new vertices in $\mathcal{C}$: this is crucial for our amortization argument.

Outline. The set $\mathcal{C}$ may have to be updated for two reasons:

- Restore the connectivity: if an edge gets deleted from the graph, or if the black box removes some vertices from $\mathcal{D}$, it may be necessary to add some vertices to $\mathcal{C}$ in order to restore the connectivity of $\mathcal{C} \cup \mathcal{D}$.
- Restore the minimality of $\mathcal{C}$: when an edge is added to the graph, or when a vertex is added to $\mathcal{C} \cup \mathcal{D}$ (either by the black box or in order to restore the connectivity), some vertices of $\mathcal{C}$ may become useless and therefore need to be removed.

We now address those two points. All our bounds are expressed in term of the total number of changes in $\mathcal{C} \cup \mathcal{D}$: let therefore k be this number of changes. We will show later that, after t updates to the graph, $k = O(t \log n)$.

The first phase of the algorithm is to restore the connectivity. We explain in the following how to decide which vertices should be added to $\mathcal{C}$ for that purpose.

Restore the connectivity after an edge deletion

To monitor the connectivity requirement, we use the following idea. The algorithm maintains a minimum spanning tree (MST) of the graph G where a weight 1 is assigned to the edges between vertices in $\mathcal{C} \cup \mathcal{D}$ (called from now on $\tilde{\mathcal{D}}$), and weight m is assigned to all other edges. These weights ensure that, as long as $\tilde{\mathcal{D}}$ is connected, the MST induces a tree on $\tilde{\mathcal{D}}$. When $G[\tilde{\mathcal{D}}]$ gets disconnected by an update, the MST uses a vertex of $V \setminus \tilde{\mathcal{D}}$ as an internal vertex: in that case, our algorithm adds this vertex to $\mathcal{C}$, to restore the connectivity. We give more details in the next section.

The edge weights are updated as the graph undergoes edge insertions and deletions and vertices enter or leave $\tilde{\mathcal{D}}$. The MST of the weighted version of the graph has the following properties.

- If $\tilde{\mathcal{D}}$ is a connected DS, then the MST has weight $(|\tilde{\mathcal{D}}| - 1) + m \cdot |V \setminus \tilde{\mathcal{D}}|$ (Kruskal's algorithm on this graph would use $|\tilde{\mathcal{D}}| - 1$ edges of weight 1 to construct a spanning tree on $\tilde{\mathcal{D}}$, then $|V \setminus \tilde{\mathcal{D}}|$ edges of weight m to span the entire graph).
- If $\tilde{\mathcal{D}}$ is a DS but $G[\tilde{\mathcal{D}}]$ is not connected, then the weight of the MST has larger value.

The two properties stem from the fact that a MST can be produced by finding a minimum spanning forest on $\tilde{\mathcal{D}}$ and extend it to a MST on V. Kruskal's algorithm ensures that this leads to a MST. In the case where $\tilde{\mathcal{D}}$ is connected, the first step yields a tree of weight $\tilde{\mathcal{D}} - 1$, and since the graph is connected the second step yields a cost $m \cdot |V \setminus \tilde{\mathcal{D}}|$. However, if $\tilde{\mathcal{D}}$ is not connected, the second step adds strictly more that $|V \setminus \tilde{\mathcal{D}}|$ edges, therefore yielding a cost bigger than $m \cdot (1 + |V \setminus \tilde{\mathcal{D}}|)$. This is more than $(|\tilde{\mathcal{D}}| - 1) + m \cdot |V \setminus \tilde{\mathcal{D}}|$, as claimed.

Furthermore, if $G[\tilde{\mathcal{D}}]$ has two connected components C_1, C_2, then the shortest of all paths between vertices $u, v,\ u \in C_1, v \in C_2$ is the minimum number of vertices whose insertion into $\mathcal{C}$ restores the connectivity requirement. Note that the shortest of all such paths must have length at most 2 (otherwise, there must be a vertex not adjacent to any vertex in $\mathcal{D}$, which contradicts the fact that $\mathcal{D}$ is a DS).

After an edge deletion, it may happen that $\tilde{\mathcal{D}}$ becomes disconnected and that the MST includes some internal vertices (at most 2, by the previous discussion) not in $\tilde{\mathcal{D}}$: in that case, we add them to $\mathcal{C}$. This turns out to be enough to ensure the connectivity.

To maintain the MST of the weighted version of the input graph we use the $O(\log^4 n)$ update time fully-dynamic MST algorithm from [22]. Since the weights of the edges incident to the vertices that enter or leave $\tilde{\mathcal{D}}$ are also updated, the algorithm runs in time $\widetilde{O}(\Delta)$ for each change in $\tilde{\mathcal{D}}$, i.e. in time $k \cdot \widetilde{O}(\Delta)$

Restore the connectivity when a vertex is deleted by the black box. When a vertex v is deleted from $\mathcal{D}$ by the black box DS algorithm, we need to be more careful: updating the edge weights and finding the new MST may add a lot of vertices to $\mathcal{C}$ (as many as Δ, one per edge of the MST incident to v). However, if the removal of v disconnects $G[\tilde{\mathcal{D}}]$, it is enough to add v to $\mathcal{C}$ to restore the connectivity. If its removal does not disconnect $G[\tilde{\mathcal{D}}]$, nothing needs to be done. It is possible to know if the graph $G[\tilde{\mathcal{D}}]$ gets disconnected using the properties of the MST, by only looking at the weight of the MST. The complexity of this step is therefore $\widetilde{O}(\Delta)$, the time needed to update the weights of the MST.

Restore the minimality. The second phase of the algorithm is to restore the minimality of $\mathcal{C}$. We explain next how to find the vertices of $\mathcal{C}$ that need to be removed to accomplish this task. This minimality condition is equivalent to the condition that all vertices in $\mathcal{C}$ are *articulation points* in the graph induced by $\mathcal{C} \cup D$. (An articulation point is a vertex such that its removal increases the number of connected components.) This turns out to be useful in order to identify which vertices need to be removed to restore the minimality of $\mathcal{C}$.

To restore the connectivity requirement, new vertices were added into $\mathcal{C}$, and the black box added some vertices to $\mathcal{D}$: this might result in some vertices in $\mathcal{C}$ not being articulation points of $G[\tilde{\mathcal{D}}]$ anymore. As observed before, these are the vertices that need to be removed. We need to identify a maximal set of such vertices that can be removed from $\mathcal{C}$ without violating the connectivity requirement. To do this, the algorithm queries in an arbitrary order one-by-one all the vertices $v \in \mathcal{C}$ to determine whether $G[\tilde{\mathcal{D}} \setminus v]$ is connected. This can be done using a data structure from Holm et al. [22] that requires $\widetilde{O}(1)$ per query. Whenever

the algorithm identifies a vertex such that $G[\tilde{\mathcal{D}} \setminus v]$ is connected, it can safely remove it from $\mathcal{C}$. The complexity of this step is therefore $\widetilde{O}(n)$ to find all articulation points, and an extra $\widetilde{O}(\Delta)$ for each of the vertices we remove from $\mathcal{C}$.

The following three lemmas conclude the proof: the first shows that the algorithm is correct, the second the $\widetilde{O}(n)$ time bound and the third the $O(\log n)$ approximation ratio.

▶ **Lemma 8.** *The algorithm that first restores the connectivity of $\mathcal{C} \cup \mathcal{D}$ and then the minimality of $\mathcal{C}$ is correct: it gives a minimal set $\mathcal{C}$ such that $\mathcal{C} \cup \mathcal{D}$ is connected.*

Proof. After restoring the connectivity requirement the algorithm maintains a spanning tree of $\tilde{\mathcal{D}}$, so $G[\tilde{\mathcal{D}}]$ is indeed connected. In the following steps, before the algorithm removes a vertex v from $\mathcal{C}$, it first verifies that $G[\tilde{\mathcal{D}} \setminus v]$ remains connected, which guarantees that $G[\tilde{\mathcal{D}}]$ is connected at the end of the update procedure. Since the black box ensures that $\mathcal{D}$ is a DS, $\tilde{\mathcal{D}}$ is a DS too: hence at the end, $\tilde{\mathcal{D}}$ satisfies both the domination and the connectivity requirements. It remains to show that $\mathcal{C}$ is minimal, i.e., that all vertices in $\mathcal{C}$ are articulation points in $G[\tilde{\mathcal{D}}]$. Since during the second step the algorithm only removes vertices from $\mathcal{C}$, a vertex that was not an articulation point cannot become one, and therefore the loop to find the articulation points is correct. The set $\mathcal{C}$ is therefore a minimal set such that $\mathcal{C} \cup \mathcal{D}$ is connected. ◀

▶ **Lemma 9.** *The amortized complexity of the algorithm is $\widetilde{O}(n)$ per update.*

Proof. The amortized cost of the black box to compute $\mathcal{D}$ is $\widetilde{O}(\Delta)$. We analyze now the additional cost of maintaining $\tilde{\mathcal{D}}$. As shown in this section, the cost to add or delete a vertex from $\tilde{\mathcal{D}}$ is $\widetilde{O}(\Delta)$. To prove the lemma, we bound the number of changes in $\tilde{\mathcal{D}}$. For that, we count the number of vertices *added* to $\tilde{\mathcal{D}}$: in an amortized sense this bounds the number of changes too. Formally, we pay a budget $deg(v)$ when v is added to $\tilde{\mathcal{D}}$. Following insertions and deletions of edges adjacent to v, we update this budget (with a constant cost), so that when v gets deleted from $\tilde{\mathcal{D}}$ a budget equal to its degree is available to spend.

From Corollary 6, the black box makes at most $\widetilde{O}(1)$ changes to $\mathcal{D}$ per update (in an amortized sense). If it removes a vertex from $\mathcal{D}$, we showed previously that no new vertex is added to $\tilde{\mathcal{D}}$. The number of additions to $\tilde{\mathcal{D}}$ is therefore $\widetilde{O}(1)$. Moreover, in the case of an edge deletion, at most two vertices are added to $\tilde{\mathcal{D}}$ to maintain the connectivity. Since restoring the minimality requires only to delete vertices, the total number of additions into $\tilde{\mathcal{D}}$ is $\widetilde{O}(1)$. As the cost for any of these additions is $\widetilde{O}(\Delta)$, the total cost of this algorithm is upper bounded by the loop to find the articulation points, which is $\widetilde{O}(n)$. ◀

▶ **Lemma 10.** *The algorithm maintains a $O(\log n)$ approximation for MCDS, i.e. $|\mathcal{C} \cup \mathcal{D}| = O(\log n) \cdot OPT$*

Proof. We first prove that $|\mathcal{C}| \leq 2|\mathcal{D}|$, using the minimality of $\mathcal{C}$. Each vertex of $\mathcal{C}$ is there to connect some components of $\mathcal{D}$. Consider the graph (W, F) where vertices W are either connected components of $\mathcal{D}$ or vertices of $\mathcal{C}$, and the set F of edges is constructed as follows. Start with a graph containing one vertex for each connected component of $\mathcal{D}$, and add vertices of $\mathcal{C}$ one by one. When the vertex v is added, identify a node u in $\mathcal{D}$ adjacent to v such that adding the edge (u, v) to F does not create a cycle: add to F an edge between v and the node corresponding to the connected component containing u. It is always possible to find such a vertex u, otherwise v would not be necessary for the connectivity, which would contradict the minimality of $\mathcal{C}$. This process gives a forest such that every node of $\mathcal{C}$ is adjacent to a connected component of $\mathcal{D}$. Since $\mathcal{C} \cup \mathcal{D}$ is connected, it is possible to complete F to make it a tree, adding some other edges. This tree has the two following properties.

1. The leaves are vertices that correspond to connected component of $\mathcal{D}$: indeed, if a vertex of $\mathcal{C}$ was a leaf in this tree, it could be removed without losing the connecting of $\mathcal{C} \cup \mathcal{D}$, which would contradict the minimality of $\mathcal{C}$.
2. Any vertex of $\mathcal{C}$ is adjacent to a connected component of $\mathcal{D}$, by construction of the forest.

These properties ensure that for every subtree rooted at a vertex of $\mathcal{C}$, there is a $\mathcal{D}$ vertex at distance at most 2 from the root: otherwise, the vertices at distance 1 from it would be from $\mathcal{C}$ and adjacent only to $\mathcal{C}$ vertices. Moreover, since a $\mathcal{C}$ vertex is not a leaf, it has necessarily some descendant and the reasoning applies. Therefore, by rooting the tree at an arbitrary vertex of $\mathcal{C}$, we can charge every $\mathcal{C}$ vertex to a $\mathcal{D}$ descendant at distance at most 2. As a $\mathcal{D}$ vertex can be charged only by an ancestor at most two levels above it, it is charged at most twice. This ensures that $|\mathcal{C}| \leq 2|\mathcal{D}|$.

Moreover, since $\mathcal{D}$ is a $O(\log n)$ approximation of MDS, $|\mathcal{D}| = O(\log n) \cdot \mathrm{OPT}$. Putting things together, we have $|\mathcal{C} \cup \mathcal{D}| = |\mathcal{C}| + |\mathcal{D}| = O(\log n) \cdot \mathrm{OPT}$. ◀

Combining Lemmas 8, 9 and 10 proves our claim: there is a $\widetilde{O}(n)$ algorithm to maintain a $O(\log n)$ approximation of the Minimum Connected Dominating Set. The main bottleneck of this approach is the time spent by the algorithm in the second phase to query all vertices in $\mathcal{C}$ in order to identify the vertices that are no longer articulation points. In the next section we present an algorithm that overcomes this limitation and is able to identify the necessary vertices more efficiently.

4 A more intricate $\widetilde{O}(\Delta)$ algorithm to restore the minimality of $\mathcal{C}$

In this section we present a more sophisticated algorithm for implementing the phase that guarantees the minimality of the maintained connected dominating set. This gives a proof of Theorem 3. We focus on a single edge update: indeed, when a vertex is added to (or removed from) $\tilde{\mathcal{D}}$, one can simply add (or remove) all its edges one by one. As in the analysis of the complexity in Lemma 9, the amortized number of changes in $\tilde{\mathcal{D}}$ is $\widetilde{O}(1)$. We aim now at proving that the time required for handling a single change is $\widetilde{O}(\Delta)$: for that, we treat edge insertions and deletions to $\tilde{\mathcal{D}}$ one by one, and prove that any edge update can be done in $\widetilde{O}(1)$, which would prove the claimed bound. Our algorithm maintains another spanning forest F of $G[\tilde{\mathcal{D}}]$ (unweighted) using the algorithm from [22].

▶ **Lemma 11.** *The vertices of $\mathcal{C}$ that are not articulation points after the insertion of the edge (v, w) all lie on the tree path $v...w$ of F. Moreover, the removal of any of these vertices results in the other vertices being articulation points again.*

Proof. Let G_b be the graph before the insertion of (v, w), and G_a be the one after. Let u be a vertex that is an articulation point in $G_b[\tilde{\mathcal{D}}]$ but not in $G_a[\tilde{\mathcal{D}}]$. Suppose by contradiction that u is not on the tree path $v...w$: that means that v and w are connected in $G_b[\tilde{\mathcal{D}}] \setminus \{u\}$. Since u is an articulation point in $G_b[\tilde{\mathcal{D}}]$, v is not connected to some vertex x in $G_b[\tilde{\mathcal{D}}] \setminus \{u\}$. But as v and w are connected in $G_b[\tilde{\mathcal{D}}] \setminus \{u\}$, adding the edge (v, w) does not connect v and x and therefore u is still an articulation point after the insertion of the edge. Therefore, all the articulation points that can be removed are in the cycle $v...w, v$. Since they are not articulation points in $G_a[\tilde{\mathcal{D}}]$, they separate $G_b[\tilde{\mathcal{D}}]$ in only two components: one with v, the other with w. Therefore, $v...w, v$ is the only cycle containing v and w, and removing any vertex from it make the articulation points of $G_b[\tilde{\mathcal{D}}]$ be articulations point in $G_a[\tilde{\mathcal{D}}]$, because they disconnect v and w again. ◀

Lemma 11 allows us to focus on the following problem: find a vertex in $\mathcal{C}$ that is no longer an articulation point in $G[\tilde{\mathcal{D}}]$ after the insertion of the edge (v, w). To achieve this, the algorithm maintains for each vertex $v \in \mathcal{C}$ the number $nc(v)$ of connected component of $G[\mathcal{D} \setminus v]$. For $v \notin \mathcal{C}$ we set for convenience $nc(v)$ to be the number of connected component in $G[\mathcal{D} \setminus v]$ plus n. This information can be used as follows: when an edge (v, w) is added, if for one vertex $u \in \mathcal{C}$ it holds $nc(u) = 1$ then u is removed from $\mathcal{C}$ (because it is no longer an articulation point). To identify such a vertex, the algorithm queries for the minimal value along the path $v...w$ in T: if the minimum value is 1, the corresponding vertex is removed from $\mathcal{C}$. This removal makes all the other vertices of the set $\mathcal{C}$ articulation points again: by Lemma 11, the cycle created by the insertion of (v, w) is broken by the deletion of u from $G[\tilde{\mathcal{D}}]$.

Notice that we are only interested in the $nc(v)$ values of the vertices in $\mathcal{C}$, as $nc(v) > n$ for $v \notin \mathcal{C}$. Since we compute a minimum and the values relevant are smaller than n, this is equivalent to ignoring v. The advantage of this offset is that when v becomes part of $\mathcal{C}$, it is sufficient to decrease its value by n to make it consistent. We now show how to keep this value up to date after adding or removing an edge.

Maintaining the $nc(v)$ values in a top-tree. For this purpose, we use the biconnectivity data structure from [22] (called *top-tree*) on the subgraph $G[\tilde{\mathcal{D}}]$. To avoid cumbersome notation, we pretend that we execute the algorithm on G, although the underlying graph on which we execute the algorithm is $G[\tilde{\mathcal{D}}]$. We also assume that the number of vertices remains n throughout the execution, which is simply implemented by removing from G all incident edges from the vertices with no incident edges in $G[\tilde{\mathcal{D}}]$.

We now briefly describe the approach of [22]. The algorithm maintains a spanning forest F of G and assigns a level $\ell(e)$ to each edge e of the graph. Let G_i be the graph composed of F and all edges of level at least i. The levels are attributed such that the following invariant is maintained:

▶ **Invariant 2.** *The maximal number of vertices in a biconnected component of G_i is $\lceil n/2^i \rceil$.*

Therefore the algorithm needs only to consider $\lceil \log_2 n \rceil$ levels. Whenever an edge (v, w) is deleted, one needs to find which vertices in the path $v...w$ in F are still biconnected. We use the following notion to describe the algorithm.

▶ **Definition 12.** *A vertex u is* covered *by a nontree edge (x, y) if it is contained in a tree cycle induced by (x, y). We say that a path $v...w$ is covered at level i if every of its node is in a tree cycle induced by an edge at level greater than i.*

Mark that all the vertices that are covered by a given edge are in the same biconnected component.

When a non-tree edge (v, w) is removed, it may affect the 2-edge connected components along the tree-path $v...w$ in T. To find which vertices are affected, the following algorithm is used in [22]. It first marks the vertices in $v...w$ as no longer covered at level $\ell(v, w)$. Then, it iterates over edges (x, y) that could cover $v...w$, i.e., the ones such that the intersection between $x...y$ and $v...w$ is not empty, and marks the vertices in this intersection as covered. This step is explained in the following function, which is called for all level i from $\ell(v, w)$ down to 0. $meet(v, w, x)$ is the intersection of the tree paths $v...w$, $v...x$ and $x...w$.

Recover(v, w, i). Set $u := v$, and iterate over the vertices of $v...w$ towards w. For each value of u, consider each nontree edge (q, r) with $meet(q, v, w) = u$ and such that $u...q$ is covered at level i. If it is possible without breaking Invariant 2, increase the level of (q, r)

to $i+1$ and mark the edges of $q...r$ covered at level $i+1$. Otherwise, mark them covered at level i and stop. If the phase stopped, start a second symmetric phase with $u = w$ and iterating on $w...v$ towards v.

As shown in [22], this is correct and runs in $O(\log^4)$ amortized time.

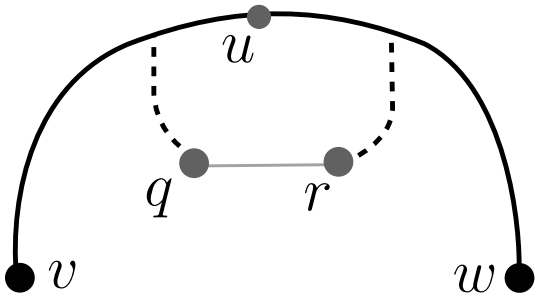


Figure 1 The edge (q, r) covers some node u on the path $v...w$.

In our case, we are interested in the vertices u whose value $nc(u)$ changes. They are exactly those that are still marked as not covered at the end of the process. Indeed, if an edge (q, r) covers a vertex u (see Figure 1), then v and w are still connected in $G[D \setminus u]$, hence the connected component of $G[D \setminus u]$ do not change. However, if u is not covered by any edge, then v and w gets disconnected in $G[D \setminus u]$, thus $nc(u)$ must be updated.

We maintain the $nc(\cdot)$ values in a top-tree, as follows. We call a *segment* a subpath of $v...w$. The idea is to maintain the non-covered segments and decrease the nc values along these at the end of the process. The top-trees allow us to alter the value of a segment of a path in $O(\text{polylog}n)$ time.

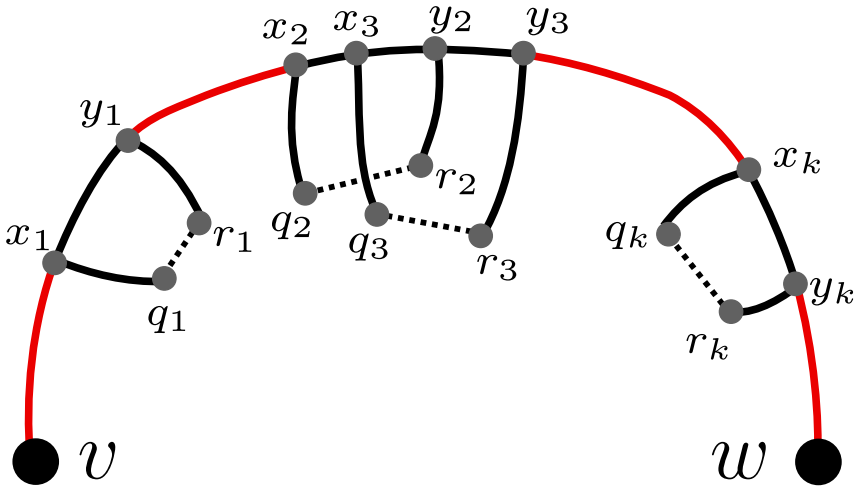


Figure 2 The black segments are covered by edges (q_i, r_i). The red segments are uncovered.

Computing the list of uncovered segments. To find the uncovered segments (in red on Figure 2), we sort the covered ones and take the complementary. Let $(q_1, r_1), ..., (q_k, r_k)$ be the nontree edges considered in the execution of Recover, and let $x_i = LCA(v, q_i)$ and $y_i = LCA(v, r_i)$ (where $LCA(u, v)$ is the lowest common ancestor of u and v in the tree). The covered segments are exactly the (x_i, y_i). Using lowest common ancestor queries, it is

possible to sort those segments according to the position of x_i along the path $v...w$. Given the segments in order, it is then possible to determine the uncovered segments in linear time: they correspond to the complementary of those segments. Answering a lowest common ancestor query on a dynamic tree can be done in $O(\log n)$ (see [29]), hence it is possible to sort the covered segments in time $O(k \log^2 n)$ and to find the uncovered segments with the same complexity.

Since k is the number of edges that move to a higher level during a call to Recover, and the maximum level is $\log n$, the total complexity of computing the uncovered segments is at most $\log^3 n$ per edges. Hence the overall complexity is $O(\log^4 n)$, which is the cost of the function Recover.

Adding an edge. To add an edge, two things are required: first decrease some nc value, and then query if a vertex has a nc value 1. We have to decrease the nc value of a vertex y if and only if its predecessor and its successor along the tree path $v...w$ were not connected in $\mathcal{D} \setminus \{y\}$ before the insertion of (v, w). This turns out to be equivalent to saying that y is not covered: thus, the algorithm needs to compute the list of segments along $v...w$ that were uncovered before the insertion of (v, w). It then must decrease the nc values along these segments, because they become connected. This is analogous to the case of an edge deletion: the latter can be used the following way. First add the edge (v, w) (and make updates to the data structure according to [22]), then delete it using the algorithm from the previous section, with the only difference that, instead of increasing the nc values along the uncovered segments, the algorithm decrease them.

It is then easy to find the minimum nc value along the path $v...w$, using the top-tree. If this value is 1, we can remove the corresponding vertex from $\mathcal{C}$. To remove it, we remove its incident edges one by one, each time updating the nc values of the remaining vertices.

The results of this section are summarized in the following lemma.

▶ **Lemma 13.** *After these updates, $\mathcal{C}$ is minimal. Moreover, the algorithm runs in amortized time $\widetilde{O}(1)$ for a single edge update.*

A direct corollary of this lemma and Lemma 9 is Theorem 3.

▶ **Corollary 14** (Theorem 3)**.** *The whole algorithm to maintain the Connected DS is correct and runs in time $\widetilde{O}(\Delta)$*

Proof. The correctness follows from Lemma 13 and from the correctness of the $\widetilde{O}(n)$ algorithm. As for the running time, the only difference from Lemma 9 is the search for articulation points: this takes $\widetilde{O}(1)$ for each edge added or removed from $\tilde{\mathcal{D}}$, and consequently $\widetilde{O}(\Delta)$ for each node added to or removed from $\tilde{\mathcal{D}}$. This yields that the algorithm takes $\widetilde{O}(\Delta)$ amortized time per update. ◀

5 A $O(\min(\Delta, \sqrt{m}))$ amortized algorithm for Minimal Dominating Set

This section presents a faster algorithm if one is only interested in finding a *Minimal* DS. This is a DS in which it is not possible to remove a vertex, but it can be arbitrarily big. For instance, in a star, the Minimum DS is only one vertex (the center), but its complementary is another minimal DS and has size $n - 1$. This result highlights the difference between MIS and Minimal DS: the best known *deterministic* complexity for MIS is $O(m^{2/3})$, whereas we present here a $O(\sqrt{m})$ algorithm for Minimal DS.

Key idea. When one needs to add a new vertex to the dominating set in order to dominate a vertex v, he can choose a vertex with degree $O(\sqrt{m})$, either v or one of its neighbors (a similar idea appears in Neiman et al. [27]). We present an algorithm with complexity proportional to the degree of the vertex added to the DS: this will give a $O(\min(\Delta, \sqrt{m}))$ algorithm. To analyze the complexity, we follow an argument similar to the one for CDS. At most one vertex is added to the DS at every step, even though several can be removed. Therefore we can pay for the (future) deletion of a vertex at the time it enters the DS.

For a vertex v, $N(v)$ is the set of its neighbors, including v. Let $\mathcal{D}$ be the dominating set maintained by the algorithm. If $v \in \mathcal{D}$ and $u \in N(v)$, we say that v *dominates* u.

For each vertex v, the algorithm keeps this sets up-to-date:

- let $N_{\mathcal{D}}(v)$ be the set of neighbors of v that are in the dominating set $\mathcal{D}$, i.e., $N_{\mathcal{D}}(v) = \mathcal{D} \cap N(v)$
- if $v \in \mathcal{D}$, let OnlyBy(v) be the set of neighbors of v that are dominated only by v, i.e., $\text{OnlyBy}(v) = \{u \in N(v) \mid |N_{\mathcal{D}}(u)| = 1\}$

Note that $N_{\mathcal{D}}(v)$ and OnlyBy(v) are useful to check, throughout any sequence of updates, whether a vertex v must be added to or removed from the current dominating set. In particular, if $N_{\mathcal{D}}(v) = \emptyset$ then v is not dominated by any other vertex, and thus it must be included in the dominating set. On the other hand, if $\text{OnlyBy}(v) = \emptyset$, all the neighbors of v (v included) are already dominated by some other vertex, and thus v could be removed from the dominating set.

5.1 The algorithm

We now show how to maintain a minimal dominating set $\mathcal{D}$ and the sets $N_{\mathcal{D}}(v)$ and OnlyBy(v), for each vertex v, under arbitrary sequences of edge insertions and deletions. We first describe two basic primitives, which will be used by our insertion and deletion algorithms: adding a vertex to and deleting a vertex from a dominating set $\mathcal{D}$.

Adding a vertex v to $\mathcal{D}$. Following some edge insertion or deletion, it may be necessary to add a vertex v to the current dominating set $\mathcal{D}$. In this case, we scan all its neighbors u and add v to the sets $N_{\mathcal{D}}(u)$. If before the update $N_{\mathcal{D}}(u)$ consisted of a single vertex, say w, we also have to remove u from the set OnlyBy(w), since now u is dominated by both v and w. If OnlyBy(w) becomes empty after this update, we remove w from $\mathcal{D}$ since it is no longer necessary in the dominating set.

Removing a vertex v from $\mathcal{D}$. When a vertex v is removed from the dominating set, we have to remove v from all the sets $N_{\mathcal{D}}(u)$ such that $u \in N(v)$. If after this update $N_{\mathcal{D}}(u)$ consists of a single vertex, say w, we add u to OnlyBy(w).

Edge insertion. Let (u, v) be an edge to be inserted in the graph. We distinguish three cases depending on whether u and v are in the dominating set $\mathcal{D}$ before the insertion. If neither of them is in the dominating set (i.e., $u \notin \mathcal{D}$ and $v \notin \mathcal{D}$), then nothing needs to be done. If both are in the dominating set (i.e., $u \in \mathcal{D}$ and $v \in \mathcal{D}$), then we start by adding v to the set $N_{\mathcal{D}}(u)$. If u was only necessary to dominate itself, we remove u from $\mathcal{D}$. Otherwise, we add u to $N_{\mathcal{D}}(v)$ and perform the same check on v.

If only one of them is in the dominating set (say, $u \notin \mathcal{D}$ and $v \in \mathcal{D}$), we have to add v to the set $N_{\mathcal{D}}(u)$. As in the case of adding a vertex to $\mathcal{D}$, this may cause the removal of another vertex from the dominating set. This can happen only if before the insertion, $N_{\mathcal{D}}(u) = \{w\}$

for some vertex w and $\text{OnlyBy}(w) = \{u\}$: in other terms, u was dominated only by w, and w was in the dominating set only to dominate u. Since after the addition of the edge (u, v) u is also dominated by v, w can be removed from the dominating set.

Edge deletion. Let (u, v) be the edge being deleted from the graph. We distinguish again the same three cases as before. If $u \notin \mathcal{D}$ and $v \notin \mathcal{D}$, nothing needs to be done. If both $u \in \mathcal{D}$ and $v \in \mathcal{D}$, we just have to remove u (resp. v) from the sets $N_{\mathcal{D}}(u)$ and $\text{OnlyBy}(u)$ (resp. $N_{\mathcal{D}}(v)$ and $\text{OnlyBy}(v)$).

If only one of them is in the dominating set, say $u \notin \mathcal{D}$ and $v \in \mathcal{D}$, then we have to remove v from $N_{\mathcal{D}}(u)$. Now, there are two different subcases:

- If $N_{\mathcal{D}}(u) \neq \{v\}$ before the deletion, then nothing needs to be done.
- Otherwise, we have to remove u from $\text{OnlyBy}(v)$: if $\text{OnlyBy}(v) = \emptyset$ after this operation, then we can safely remove v from $\mathcal{D}$. The algorithm must find a new vertex to dominate u: we simply add u to the dominating set.

5.2 Running time

Adding or removing a vertex v from the dominating set can be done in time $O(deg(v))$, where $deg(v)$ is the degree of v in the current graph. While several vertices can be removed from $\mathcal{D}$ at every step, only one can be added (following an edge deletion): the amortized complexity of the algorithm is therefore $O(\Delta)$, where Δ is an upper bound on the degree of the nodes.

Nevertheless, it is possible to chose the vertex to be added to the dominating set more carefully. When the algorithm must find a new vertex to dominate vertex u, it does the following:

- If $deg(u) \leq 2\sqrt{m} + 1$, the algorithm simply adds u to $\mathcal{D}$.
- Otherwise, $deg(u) > 2\sqrt{m} + 1$. The algorithms finds a vertex $w \in N(u)$ with $deg(w) \leq \sqrt{m}$ and adds w to $\mathcal{D}$. Note that such a vertex w can be found by simply scanning only $2\sqrt{m} + 1$ neighbors of u, since (by averaging) at least one of them must have degree smaller than $\sqrt{m}$.

In both cases, the insertion takes time $O(\min(\Delta, \sqrt{m}))$.

When a vertex v is deleted from the dominating set, its degree can be potentially larger than $2\sqrt{m}$. However, when v was added to the dominating set its degree must have been $O(\sqrt{m})$: this implies that many edges were added to v, and we can amortize the work over those edges. More precisely, when a vertex v enters the dominating set, we put a budget $deg(v)$ on it. Every time an edge incident to v is added to the graph, we increase by one this budget, so that when v has to be removed from $\mathcal{D}$, v has a budget larger than $deg(v)$ that can be used for the operation.

References

1 Raghavendra Addanki and Barna Saha. Fully Dynamic Set Cover–Improved and Simple. *arXiv preprint*, 2018. arXiv:1804.03197.

2 Sepehr Assadi, Krzysztof Onak, Baruch Schieber, and Shay Solomon. Fully dynamic maximal independent set with sublinear update time. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, 2018.

3 Sepehr Assadi, Krzysztof Onak, Baruch Schieber, and Shay Solomon. *Fully Dynamic Maximal Independent Set with Sublinear in n Update Time*, pages 1919–1936. SIAM, 2019. doi:10.1137/1.9781611975482.116.

4 Aaron Bernstein, Sebastian Forster, and Monika Henzinger. A Deamortization Approach for Dynamic Spanner and Dynamic Maximal Matching. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1899–1918. SIAM, 2019. doi:10.1137/1.9781611975482.115.

5 Aaron Bernstein and Cliff Stein. Faster fully dynamic matchings with small approximation ratios. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms*, pages 692–711. Society for Industrial and Applied Mathematics, 2016.

6 Sivakumar R Bevan Das and V Bharghavan. Routing in ad-hoc networks using a virtual backbone. In *Proceedings of the 6th International Conference on Computer Communications and Networks (IC3N'97)*, pages 1–20, 1997.

7 Sayan Bhattacharya, Deeparnab Chakrabarty, Monika Henzinger, and Danupon Nanongkai. Dynamic algorithms for graph coloring. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1–20. Society for Industrial and Applied Mathematics, 2018.

8 Sayan Bhattacharya, Monika Henzinger, and Giuseppe F Italiano. Deterministic fully dynamic data structures for vertex cover and matching. *SIAM Journal on Computing*, 47(3):859–887, 2018.

9 Sergiy Butenko, Xiuzhen Cheng, Carlos A Oliveira, and Panos M Pardalos. A new heuristic for the minimum connected dominating set problem on ad hoc wireless networks. In *Recent developments in cooperative control and optimization*, pages 61–73. Springer, 2004.

10 Xiuzhen Cheng, Xiao Huang, Deying Li, Weili Wu, and Ding-Zhu Du. A polynomial-time approximation scheme for the minimum-connected dominating set in ad hoc wireless networks. *Networks: An International Journal*, 42(4):202–208, 2003.

11 V. Chvatal. A Greedy Heuristic for the Set-Covering Problem. *Mathematics of Operations Research*, 4(3):233–235, 1979. URL: http://www.jstor.org/stable/3689577.

12 Camil Demetrescu and Giuseppe F. Italiano. A New Approach to Dynamic All Pairs Shortest Paths. *J. ACM*, 51(6):968–992, 2004.

13 Ding-Zhu Du and Peng-Jun Wan. *Connected dominating set: theory and applications*, volume 77. Springer Science & Business Media, 2012.

14 Uriel Feige. A threshold of ln n for approximating set cover. *Journal of the ACM (JACM)*, 45(4):634–652, 1998.

15 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

16 Sudipto Guha and Samir Khuller. Approximation algorithms for connected dominating sets. *Algorithmica*, 20(4):374–387, 1998.

17 Leonidas Guibas, Nikola Milosavljević, and Arik Motskin. Connected dominating sets on dynamic geometric graphs. *Computational Geometry*, 46(2):160–172, 2013.

18 Anupam Gupta, Ravishankar Krishnaswamy, Amit Kumar, and Debmalya Panigrahi. Online and dynamic algorithms for set cover. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 537–550. ACM, 2017.

19 Manoj Gupta and Shahbaz Khan. Simple dynamic algorithms for Maximal Independent Set and other problems. *arXiv preprint*, 2018. arXiv:1804.01823.

20 Manoj Gupta and Richard Peng. Fully dynamic (1+ e)-approximate matchings. In *Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on*, pages 548–557. IEEE, 2013.

21 Johan Håstad. Clique is hard to approximate withinn 1- ε. *Acta Mathematica*, 182(1):105–142, 1999.

22 Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. Poly-logarithmic Deterministic Fully-dynamic Algorithms for Connectivity, Minimum Spanning Tree, 2-edge, and Biconnectivity. *J. ACM*, 48(4):723–760, 2001.

23 Lujun Jia, Rajmohan Rajaraman, and Torsten Suel. An efficient distributed algorithm for constructing small dominating sets. *Distributed Computing*, 15(4):193–205, 2002.

24 Viggo Kann. *On the approximability of NP-complete optimization problems*. PhD thesis, Royal Institute of Technology Stockholm, 1992.

25 Fabian Kuhn and Roger Wattenhofer. Constant-time distributed dominating set approximation. *Distributed Computing*, 17(4):303–310, 2005.

26 D. Nanongkai, T. Saranurak, and C. Wulff-Nilsen. Dynamic Minimum Spanning Forest with Subpolynomial Worst-Case Update Time. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 950–961, October 2017.

27 Ofer Neiman and Shay Solomon. Simple deterministic algorithms for fully dynamic maximal matching. *ACM Transactions on Algorithms (TALG)*, 12(1):7, 2016.

28 E Sampathkumar and HB Walikar. The connected domination number of a graph. *J. Math. Phys*, 1979.

29 Daniel D Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *Journal of computer and system sciences*, 26(3):362–391, 1983.

30 S. Solomon. Fully Dynamic Maximal Matching in Constant Update Time. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 325–334, October 2016. doi:10.1109/FOCS.2016.43.

31 Jie Wu and Hailan Li. On calculating connected dominating set for efficient routing in ad hoc wireless networks. In *Proceedings of the 3rd international workshop on Discrete algorithms and methods for mobile computing and communications*, pages 7–14. ACM, 1999.

Detecting Ghostwriters in High Schools

Magnus Stavngaard August Sørensen Stephan Lorenzen†
Niklas Hjuler Stephen Alstrup *

University of Copenhagen - Department of Computer Science
Universitetsparken 3, Copenhagen, Denmark
† Corresponding author, e-mail: lorenzen@di.ku.dk

Abstract. Students hiring *ghostwriters* to write their assignments is an increasing problem in educational institutions all over the world, with companies selling these services as a product. In this work, we develop automatic techniques with special focus on detecting such ghostwriting in high school assignments. This is done by training deep neural networks on an unprecedented large amount of data supplied by the Danish company MaCom, which covers 90% of Danish high schools. We achieve an accuracy of 0.875 and a AUC score of 0.947 on an evenly split data set.

1 Introduction

The number of Danish high school students using ghostwriters for their assignments has been rising at an alarming rate due to the emergence of several new online services, allowing students to hire others to write their assignments[1].

We consider in this paper the problem of detecting such ghostwriting, or as it is more commonly known: *authorship verification*. Authorship verification is a common task in natural language processing [2, 3, 4]: Given author α with known texts $t \in T_\alpha$ and unknown text x, determine whether α is the author of x. Often, a set of texts $\overline{T_\alpha} = T \setminus T_\alpha$ (T denoting the complete set of available texts) not written by α is also available, which can be utilized as examples of different writing styles, when training a model. Note however, that $\overline{T_\alpha}$ is unlikely to contain examples written by the true author of x, unlike in the related *authorship identification* problem, in which the task is to determine the exact author of x, given a set of candidate authors and their texts [5, 6].

In this paper, we focus on the problem in high schools. We have access to a large data set consisting of 130K Danish essays, written by more than 10K high school students[1]. Thus we have access to a lot of different authors, each with a large amount of text. We suggest a *generalizing* technique for authorship verification (as opposed to *author specific* models); using a Siamese network working at character level (an approach inspired by [5]), writing style representations are learned and compared, in order to compute the style similarity between two texts. Using the similarity measure provided by this network, x are compared to previous works $t \in T_\alpha$, and a final answer is given by a weighted combination of the individual similarities. The data used is supplied by MaCom, the company behind Lectio, the largest learning management system in Denmark.

*Supported by the Innovation Fund Denmark through the Danish Center for Big Data Analytics Driven Innovation (DABAI). The authors would like to thank MaCom.

[1]The data set is proprietary and not publicly available.

Many previous approaches for authorship verification/identification are based on excessive feature selection [7, 2], but neural network approaches have also been considered, for instance [3] who utilize recurrent neural networks for identification. Previous work on Danish high school essays have used author specific models for verification/identification [6], but this work is the first neural network based approach used on this data (and, to our knowledge, in this setting).

2 Method

As mentioned, we solve the authorship verification problem in two steps. First, we solve the problem of computing the writing style similarity between two texts by learning the similarity function $s : T \times T \rightarrow [0, 1]$ using a Siamese network (Section 2.1). Second, we solve the authorship verification problem for author α by combining similarities computed between the unknown text x and the known texts $t \in T_\alpha$. We consider several different ways to combine these similarities, based on their value and relevant meta data. (Section 2.2).

2.1 Network

Several different architectures are considered, using different input channels (e.g. char, word, POS-tags), and evaluated on a validation set. The architecture of our best performing network is shown in Figure 1.

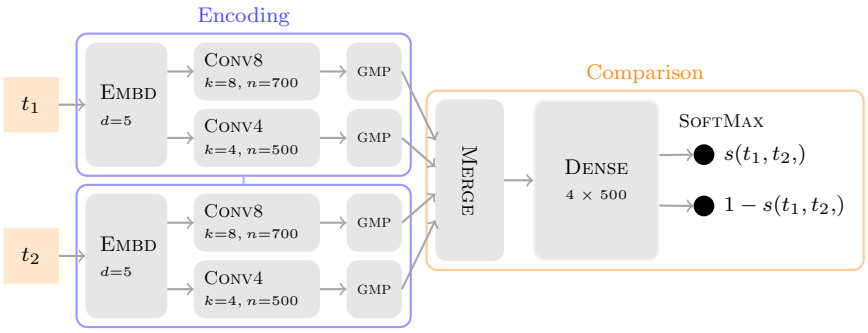


Fig. 1: Network architecture.

The Siamese network can be considered in two parts: *encoding* and *comparison*, the main idea being to learn an encoding of writing style, that the network is then able to distinguish. Our network uses only character level inputs.

The **encoding part** consists of a character embedding (Embd), followed by two different convolutional layers: Conv8 using kernel size $k = 8$ and $n = 700$ filters, and Conv4 using $k = 4$ and $n = 500$. Each convolutional layer is followed by a global max pooling layer (GMP). The weights of Embd and Conv8/Conv4 are shared between encoding t_1 and t_2.

In the **comparison part**, we first compute the absolute difference between the encodings in the Merge layer. Afterwards, 4 dense layers with 500 neurons

each are applied (DENSE), and finally, the output is normalized by use of a softmax layer with two outputs.

2.2 Combining similarities

Having a good estimate of $s(t_1, t_2)$ for any two texts, we consider different ways to combine these similarities, in order to give the final answer to an authorship verification query. More specifically, we consider functions $C_s : \mathcal{P}(T) \times T \rightarrow [0, 1]$, such that, given x and T_α, we will answer the query positively (i.e. α is the author of x) if:

$$C_s(T_\alpha, x) \geq \delta$$

where δ is a configurable threshold, which describes how likely we are to answer positively. In the experiments, we consider several different ways to combine similarities, for instance using weighted sums, the min/max similarity or majority vote, while utilizing meta data such as time stamps and text length. From the experiments, we found that the optimal strategy was a weighted sum with weights decaying exponentially with time:

$$C_s(T_\alpha, x) = \sum_{t \in T_\alpha} e^{-\lambda \tau(t)} s(t, x) \tag{1}$$

where $\tau(t)$ denotes the time in months since t was written, and λ is a configurable parameter, which is determined experimentally.

3 Experiment

This section describes our experiments performed on the MaCom data. Section 3.1 will describe the preprocessing and partitioning of data. Baselines will be described in Section 3.2. Finally, Section 3.3 lists and discusses the final results. We use accuracy, *false accusation rate*, $\mathrm{FAR} = \mathrm{FN}/(\mathrm{TN} + \mathrm{FN})$, and *catch rate*, $\mathrm{CR} = \mathrm{TN}/(\mathrm{TN} + \mathrm{FP})$ as performance metrics.

3.1 Data

The data is partitioned into three sets: T_{train} used for training, T_{val} used for early stopping and selecting C_s, and T_{test} used only for estimating the metrics of the final models. The three sets are author disjoint, meaning no author will appear in more than one of the sets. In an effort to remove invalid data (blank hand-ins, etc.), we clean the data by filtering according to length (keeping texts with lengths between 400 and 30,000 characters). Furthermore, some texts were found to include author revealing information (such as name, address); hence we removed all proper pronouns from the texts, as well as the first 200 characters. Finally, authors with less than 5 texts were removed.

After cleaning, the data set contains a total of 131,095 Danish essays, written by 10095 authors, with an average 13.0 texts per author, and an average text length of 5894.8 characters.

For each data set, we construct two types of problem instances: SIM and AV, used for training the network and selecting the combination strategy respectively. The data set has no labelled ghostwriters, so we assume all authors to be correct[2], and construct balanced (50/50) data sets as follows:

A SIM instance simply consists of two texts t_1, t_2 and a label indicating whether the texts are by the same author. Positive samples are generated by using $t_1, t_2 \in T_\alpha$, while negative samples are generated by using $t_1 \in T_\alpha$ and $t_2 \in \overline{T_\alpha}$. An AV instance consists of a set of known texts T'_α, an unknown text x, and a label indicating whether α is (positive) or is not (negative) the author of x. Letting t_{last} denote the most recent text of T_α, samples are generated using $T'_\alpha = T_\alpha \setminus \{t_{last}\}$ with $x = t_{last}$ for the positive sample, and $x \in \overline{T_\alpha}$ chosen at random for the negative sample.

Table 1 provides an overview of the data after partitioning and preprocessing.

Data set	#authors	#texts	#SIM	#AV
T_{train}	5418	70432	934720	10836
T_{val}	989	12997	173536	1978
T_{test}	3688	47666	627744	7376

Table 1: Data set overview.

3.2 Baselines

We will compare our method to Burrows's Delta method and author specific SVMs:

Burrows's Delta method (BURROWS) [7] is a method for authorship identification based on the l_1-distance between the z-scores of word frequencies in x and in the corpus for each of the candidate authors $\beta_1, ..., \beta_k$. We adapt it for verification by sampling a set of 'wrong' authors, $\beta_2, ...\beta_k$, and querying with x and $\beta_1 = \alpha, \beta_2, ..., \beta_k$. answering positively, if x is attributed to α. The top 150 word frequencies are considered. The optimal k is determined using T_{train}.

An author specific SVM [6, 2] is trained for each author in order to recognize T_α from $\overline{T_\alpha}$. Hyper parameters and features are selected using cross validation. Forward feature selection is used, considering char, word and POS-tag n-grams for varying n. The SVM will be trained on a balanced set, meaning that only a limited amount of data is available for each SVM. However, they have previously been shown to work well in this data set [6].

3.3 Results

Methods were trained and validated on T_{train} and T_{val}. For BURROWS, we found $k = 4$ to give the best results, while the parameters $C = 10, \gamma = 10^3$ were found optimal for the RBF kernel SVM. The optimal combination strategy C_s was

[2]An undoubtedly false assumption, which will be discussed in Section 3.3

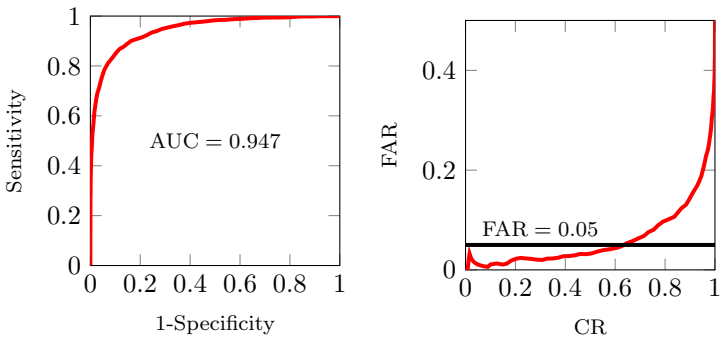


Fig. 2: ROC (left) and plot of false accusation rate/catch rate (right) on T_{test}.

found to be exponentially decaying weights (see (1)) with $\lambda = 0.1$. Furthermore, $\delta = 0.57$ was found to be optimal. Using these parameters, the baselines and our method were evaluated on T_{test}; Table 2 presents the results, while Figure 2 shows the ROC/AUC and a plot of false accusation/catch rate for our method. As it can be seen, our method clearly outperforms the baselines, on all metrics.

Method	Accuracy	FAR	CR
Burrows	0.677	0.357	0.806
SVM	0.720	0.266	0.689
Our method	0.875	0.141	0.896

Table 2: Results obtained on T_{test}

The false accusation rate is especially important considering the use case: when trying to detect ghostwriting in high schools, making false accusation can be especially devastating, as students found guilty of cheating could risk severe punishment and maybe even be expelled. Using this metric, our method performs very well, as illustrated in Figure 2 (right), a fairly low FAR can be obtained, while still catching a lot of ghostwriters. Optimizing the method on T_{val} while restricting FAR < 0.1, we achieved an accuracy of 0.864, FAR $= 0.106$ and CR $= 0.825$ on T_{test} (with exponential weighting and parameter $\lambda = 0.16$). However, even if these results are promising, the system should only be used as a warning system for the teacher, who should always have the final say.

An interesting aspect to note about the combination strategy C_s, is that it takes time into account with $\lambda = 0.1$, weighing recent assignments more than older ones. Since $\tau\,(t)$ measures in months, this means that a recent assignment gets $e^{12 \cdot 0.1} \approx 3.3$ times the weight of a one year old assignment. This corresponds well with the idea that high school students writing style changes over time, as also observed in [6].

When looking at the low false accusation rates of Figure 2 (right), one have to consider two things before translating them into practice: a) T_{test} is balanced,

while in reality much less than half of assignments are written by a ghostwriter, and b) ghostwriting does happen, also in our data set, and thus most likely some of our labels are wrong. A possible remedy for the second point could be to adjust FN to $\mathrm{FN} - \frac{\mathrm{TN}}{\mathrm{TN+FP}}\gamma\mathrm{T}$ (where γ is the estimated fraction of ghostwriters and $\mathrm{T} = \mathrm{TP}+\mathrm{FN}$), and similar for TP, under the assumption that a negative sample and a corrupted positive sample are indistinguishable. Adjusting for this would obviously lead to improved accuracy and false accusation rate, but requires a good estimate of γ.

4 Conclusion

We achieved an accuracy of 0.875, with a false accusation rate of 0.141 and a catch rate of 0.896. We show how false accusation rate can be improved at the cost of catch rate and accuracy. Results are good enough for practical use, and even with a slightly lower catch rate, the system is still expected to have a preventive effect. However, one has to keep in mind that, in practice, the data set is not 50/50 balanced, which obviously will affect the results. Making a split imitating the real world is hard for two reasons: one needs a good approximation of the actual fraction of ghostwriters, and even if this fraction is known, the number of corrupt labels would be approximately the same as the number of negatives, making it impossible to beat a false accusation rate of 0.5, even for a perfect classifier. Finding a clean data set or establishing ground truth would alleviate these problems, and could be interesting prospects for future work.

Another interesting direction is to analyze writing style changes over time more in depth, motivated by the chosen combination strategy and preliminary experiments, which show how two texts written within a shorter time span have higher similarity on average.

References

[1] Politisk flertal vil gøre salg af eksamensopgaver ulovligt. `http://nyheder.tv2.dk/politik/2017-06-21-politisk-flertal-vil-gore-salg-af-eksamensopgaver-ulovligt`. Accessed: 2018-11-25.

[2] Efstathios Stamatatos. A survey of modern authorship attribution methods. *J. Am. Soc. Inf. Sci. Technol.*, 60(3):538–556, March 2009.

[3] Douglas Bagnall. Author Identification using multi-headed Recurrent Neural Networks. In *CLEF 2015 Evaluation Labs and Workshop – Working Notes Papers*. CEUR-WS.org, September 2015.

[4] Alberto Bartoli, Alex Dagri, Andrea De Lorenzo, Eric Medvet, and Fabiano Tarlao. An author verification approach based on differential features. In *CEUR WORKSHOP PROCEEDINGS*, volume 1391. CEUR, 2015.

[5] Chen Qian, Tianchang He, and Rao Zhang. Deep learning based authorship identification. 2018. report, Stanford University.

[6] Niels Dalum Hansen, Christina Lioma, Birger Larsen, and Stephen Alstrup. Temporal context for authorship attribution: a study of Danish secondary schools. In *Multidisciplinary information retrieval*, pages 22–40. Springer, 2014.

[7] John Burrows. 'Delta': a Measure of Stylistic Difference and a Guide to Likely Authorship. *Literary and Linguistic Computing*, 17(3):267–287, 2002.

DABAI: A data driven project for e-Learning in Denmark

Stephen Alstrup, Casper Hansen, Christian Hansen, Niklas Hjuler, Stephan Lorenzen, and Ninh Pham

Department of Computer Science, University of Copenhagen, Denmark
s.alstrup@di.ku.dk
bnq@di.ku.dk
chrh@di.ku.dk
hjuler@di.ku.dk
lorenzen@di.ku.dk
pham@di.ku.dk

Abstract: A new Big Data research team called DABAI have been launched in Denmark, which aims at integrating cutting edge computer science research from machine learning, algorithms and visualization into the education sector. The educational part of the DABAI project is a cooperation between Danish universities and multiple enterprises providing e-Learning solutions for the Danish market. The companies' services cover over 90% of the Danish schools, with more than one million students, who on a daily basis do millions of exercises and interactions using the involved companies' solutions. The study presented in this paper is an initial investigation of the needs of the three largest companies in e-Education in Denmark directly involved in DABAI, as well as other companies, with the goal being to continue providing novel and high-demand features for their customers. The three companies are MaCom, Clio Online, and Edulab. Clio Online together with Edulab provide an online platform for teaching material and exercises for the primary school level covering all subjects. MaCom provides a lecture management system used by most Danish high schools. Overall the study shows that the problems encountered at the different companies are varied, but can be categorized into three general sub categories: Student Profiling, Content Profiling, and Content Recommendation. Some problem types fall into multiple sub categories, and in general to accomplish the goal of providing e-Learning of the highest quality, research into all of them is necessary. This paper presents the fundamental problems in e-Learning. For each encountered problem, we describe its objectives and challenges in detail, followed by the current state of the art for solving it.

Keywords: e-Learning, e-Learning challenge categorization, Big Data

1. Introduction

A new Big Data research team called Danish Center for Big Data Analytics driven Innovation (DABAI) have been launched in Denmark, which aims at integrating cutting edge computer science research from machine learning, algorithms, and visualization into the domains encompassing Food Supply Chain Data, Societal Data, and Education Data. DABAI is funded by Innovation Fund Denmark, with a total funding of 17M euro. In this paper we will present our initials finding for problems in the educational sector. The educational part of the DABAI project is a cooperation between Danish universities and multiple enterprises providing e-Learning solutions for the Danish market. The companies' services cover over 90% of the Danish schools, with more than one million students, who daily do millions of exercises using the involved companies' solutions. In Denmark there is a strong political focus and funding to ensure the education system being more data driven, and e-Learning tools developed being integrated quickly. Such a hard political push on the software level is now possible since it has been ensured that the schools have the necessary hardware level over the past years.

This work is an initial investigation of the needs of the three largest companies in e-Education in Denmark directly involved in DABAI, with the goal being to continue providing novel and high-demand features for their customers. The three companies are MaCom, Clio Online, and Edulab. Clio Online together with Edulab provide an online platform for teaching materials and exercises for the primary school level covering all topics. MaCom provides a lecture and assignment management system used by most Danish high schools.

Overall our study shows that the problems encountered at the different companies are very diverse, but can be categorized into three general sub categories: Student Profiling, Content Profiling, and Content Recommendation. Some problem types fall into multiple sub categories, and in general to accomplish the goal of providing e-Learning of the highest quality, research into all of them is necessary.

In this paper we present the fundamental problems in e-Learning, and for each encountered problem we describe its objectives and challenges in detail, followed by the current state of the art for solving it. We strongly believe that even though this work is done in Danish context, then the encountered problems and solutions generalizes worldwide.

1.1 Introduction of companies

In this section we will briefly introduce the three companies we are working with, such that the challenges they are facing can be better understand in the setting they are going to be used in.

Clio Online is a complete digital learning platform for the primary school level, and aims at offering everything a teacher and student need. They offer all subjects except mathematics, which Edulab provides. The material is tailored to the national targets for education, such that the material always follows the national curriculum for each subject. They want the teachers to be able to focus more on teaching, and less on preparing and finding the right material.

Edulab is a complete digital learning platform for mathematics, with the goal of making every child in the world better at mathematics. Edulab's products offers many different ways of learning mathematics, ranging from question heavy workloads to video and text lessons, as well as other activities depending on if the student is in class or at home.

MaCom provides *Lectio*, which is a lecture management system used in high schools. They provide a tool to manage the lessons carried out, and in a way that is transparent to the students and the parents. They also provide a tool to manage homework and for handing in assignments, such that they have a big repository of student work.
Together these companies services over 90% of all schools in Denmark.

1.2 Categorization of e-Learning challenges

One of the end goals for educational software is obtaining differentiable teaching materials fitted to each student's needs, such that the overall progression of the students is as high as possible. The current generation of educational software offer personalization options, but not as in depth compared to what a real human personal tutor could provide, and bridging this gap is one of the major challenges for current educational software providers. However, this task is complex and can be split in many subtasks, which also independently can provide great value. Our cooperation with the three largest e-Learning companies in Denmark have revealed that their needs can be put into three categories:

1. Student Profiling
2. Content Profiling
3. Content Recommendation

Student profiling is the broadest of the categories and cover modelling the students' skills, knowledge, and/or behaviour. These can be modelled by how the students interact with the educational systems, and how they perform in these systems. We present two cases on student profiling, one from modelling how students interact with the website and material, and another on modelling a student's writing style.

Content profiling is the task of understanding the material provided by the educational system, the understanding can be in the case of finding material that students generally perform much worse on than expected, or finding relations between material necessary for building solid material recommender systems. We present a case on content profiling with the task of finding similarities among a large number of quizzes.

Content Recommendation is in its essence the combination of student and content profiling, where the task is to recommend the right learning material to the student. A detailed profiling of both students and content allows to create optimal learning paths for the individual students. We present three cases on this category from improving personalized learning, to predicting student performance on quizzes based on other students' performance, and a system to train specific parts of a subject curriculum.

2. Company faced e-Learning challenges

In this section we will present our initial work and related literature on six e-Learning challenges faced by the companies.

2.1 Optimize e-Learning personalization

E-Learning personalization refers to an online educational approach that provides differentiated instruction to support the individual student's need. Currently, EduLab is deploying and developing an online learning platform called **"***SuperTrainer***"** to optimize the e-Learning personalization for individual student on learning Mathematics. In a nutshell, SuperTrainer can be seen as a specific recommender system that recommends *"the best question at the right time"* to students to optimize their mathematical learning process in primary school. Such adaptive recommender system provides a sequence of math questions in order to maximize student's learning gain while taking into account the limited amount of time and number of questions. One traditional mechanism is that students will be assigned easier/harder questions if they answer incorrectly/correctly on previous questions. This is due to the argument in psychology (Berlyne, 1960) and neuroscience (Gottlieb et al., 2013) that the human brain shows intrinsic pleasure in activities of optimal difficulty, i.e. balance between interest (not too difficult) and challenge (not too easy).

While traditional recommender systems are focusing on "one-shot" recommendations, e-Learning personalization investigates a *"personal path"* through the questions and adjusts this path depending on how students progress in learning. Therefore, it is essential to have a systematic measurement of individual student's performance when using personalized products. Currently, to the best of our knowledge, there is no measurement of student's performance on using e-Learning personalization. This means that we do not know how well we could do with e-Learning personalized platforms. Furthermore, there is no efficient mechanism with guarantees to recommend "*the best question at the right time"* to students.

To handle recent problems of e-Learning personalization, we aim to introduce and formalize important progress factors to measure students' performance, and then exploit such measurements to evaluate and improve an e-Learning personalization mechanism. In particular, we decompose student's progress into three factors as follows:

- **Ability of understanding:** Given a fixed number of questions, for each student we want to *maximize* the correctness/difficulty level of the given questions.
- **Learning speed:** Given a fixed amount of questions, we want to *minimize* the total amount of time that each student has to spend on solving them.
- **Balance of challenge and interest:** Given a fixed amount of questions, we want to maintain a well-defined correct ratio to balance between the *challenge* (i.e., student feels difficult to answer correctly) and the *interest* (i.e., student feels happy when answering correctly).

After formalizing student's performance measurements, or goal is to model the e-Learning personalization problem, i.e., recommending the best question at the right time to students, as an optimization problem. Following very recent machine learning approaches (Lopes et al., 2015; Tekin, Braun, and van der Schaar, 2015), we are investigating the *"Multi-Arm bandit models"* for our optimization. Assume that an e-Learning personalization has to recommend a sequence of questions to each individual student. Each question will give the student a reward value that highly depends on the student's level at that time. Note that such reward value must express the three progress factors above so that we are able to measure the student's progress. Then, we develop the optimization algorithms to maximize the sum of rewards earned by each student through a sequence of recommended questions. Our collaboration with EduLab will deploy a scalable and efficient algorithm for optimizing their e-Learning personalization platform – SuperTrainer – since EduLab is now dealing with very large-scale data, i.e., millions of questions on every day.

2.2 Student behavior modelling

Most learning platforms allow a student to access the material through many different paths. An example could be a video lesson which could be given as homework, it could come up as recommended material to an exercise if the student needs help solving it, or the student could simply find the video lesson themselves because they want to review the material without being asked to do it. Edulab is interested in the paths users

take for two primary reasons: 1) as for every other company providing a web solution, knowledge of user behavior can be used directly when planning further extensions of the system. 2) Educational systems are built based on knowledge of didactics which guide the usage of the system, and unintended usage of the system can therefore in some cases correspond to sessions with suboptimal learning for the user.

The number of daily user sessions is too large to make any meaningful qualitative study of the individual user sessions, where session in this context is defined as the interactions a user does from logging into the system to the time they log out. It is therefore necessary to do some clustering of sessions and analyze the resulting clusters, to extract general trends. A common approach (Köck, M. and Paramythis, A. 2011) for modelling the sessions, is to consider them as a sequence over an action space. A simple action space is that the user either answers a quiz (Q) or watches a video (V). A user session would then consist of entering the system, a sequence of actions, e.g. QQQVVQ, and then logging off. A clustering over such a simple space, would still allow us to ask questions like, does students in general start with videos or quizzes, and are the user interactions very binary such that they either answer questions or watch lessons? This is a very trivial example of an action space, but it can be made arbitrarily complex according to the system we wish to model, with the limitation being that larger action spaces requires more data to find meaningful insights. The method of considering user sessions as sequence over an action space have been used with success in deriving insights from educational systems (Klingler, S. et al. 2016; Faucon, L., Kidzinski, L., and Dillenbourg, P. 2016).

We are currently in the process of clustering all sessions done in Edulab's system on multiple different state spaces with the focus on finding "unproductive" sessions, where the students use the system in an unintended way. An initial study on a subset of the available data have been done (Hansen, C. et al. 2017), where the clustering lead to insight into a significant number of sessions where students had very binary behavior, meaning they either always watched lessons, answered questions correctly or answered wrongly. This study used an action space of considering video lesson, if a question was correctly and incorrectly answered, and if the student switched to material of different topics. The overall goal of this work is to provide Edulab with a new tool for future development of their system.

2.3 Predict student performance

Prediction of student performance, estimating the unknown score of a given task, is one of the important problems in e-Learning. The scores given by an individual student reflect how a student understands and applies the knowledge conveyed in class. A reliable performance prediction of such scores enables teachers to provide remedial support for weak students, recommend appropriate tasks to excellent students, generate adaptive hints, and improve the learning of students. This section focuses on predicting the score of students in the quiz system of the Clio Online learning platform, where a student has answered only a subset of the quizzes.

Currently, Clio Online learning platform is maintaining a quiz system that contains thousands of quizzes with several types, including multiple choice quizzes, gap-filling quizzes, etc., spanning several different elementary school subjects. We study the performance of students on the quiz data; specifically we will focus on predicting how a student performs on an unseen quiz. More formally, given n students $S = \{s_1, \ldots, s_n\}$, and a set of m quizzes $Q = \{q_1, \ldots, q_m\}$, any student s_i will answer some quizzes in Q and we need to predict the score of the other quizzes that he has not finished yet. In other words, given an incomplete matrix X of size $n \times m$ reflecting the scores of n students over m quizzes, our task is to efficiently complete such matrix X given its partial known values. Since we can view a student/quiz as a user/item, we view our prediction task as the *collaborative filtering problem* (Collaborative Filtering, 2017) and investigate state-of-the-art techniques for solving it.

In particular, we have studied the *matrix factorization techniques* (Koren, Bell, and Volinsky, 2009; Lee and Seung, 2000) for improving our prediction. This is due to the fact that we can assume that there are a small number of latent features, e.g., skill sets, revealing the students and tasks preferences. Such assumption is natural and has been used widely in research and application work in educational data (Barnes, 2005; Desmarais, 2011). It is also worth noting that these solutions require no knowledge of students and tasks, and therefore avoid the need for extensive data collection. Prior work using matrix factorization for predicting student performance (Elbadrawy et al., 2016; Thai-Nghe et al., 2010) indicates that there is sufficient information in the historical student-task score data to make the prediction feasible.

2.4 Similarity among quizzes

Given a par of quizzes, Quiz 1 and Quiz 2, how similar are they? There exist many qualitative ways of addressing this question and some quantitative like Pearson correlation measures and cosine similarity. Deary et al (2007) research correlation among intelligence tests and how well kids perform in different subjects. Even with the high number of answers in the Clio Online setting (millions of quiz answers), the thousands of quizzes results in many pairs of quizzes which have only a few or even no students who have taken both quizzes. This results in very high variance for the usual techniques. One could combat this by densifying the data with prediction values for the quizzes which would reduce the similarity problem to predicting student performance. The other option is to infer similarity in the sparse quizzes going through a third quiz taking advantage of the "transitive" property of objects being similar.

Formally we solve the problem of making the best prediction on normalized quiz scores (0 mean and unit variance for each quiz). The prediction we make is a weighted average of the other quizzes taken and the weights are the similarities. We require the similarities to be symmetric and non-negative, but other than that there is no restriction. This allow us to both train and check different overfitting measures to capture the "transitive property" of objects being similar.

The knowledge of similarities would give insight to the teachers. If for example a student suddenly does worse on a writing test than he usually does, one can find the similarities of more specific quizzes (like grammar quizzes) and thus pinpoint why this student has trouble with the specific writing quiz. Furthermore, taking advantage of similarities is likely to improve prediction (Khajah, Lindsey, and Mozer 2016). Chen, Gonxáles-Brenes and Tian (2016) suggest that spurious similarities could be due to the quiz requiring multiple skills.

2.5 Authorship verification

The competition for high grades in Danish secondary education (high school) is tougher than ever. This has led to an increase in fraud in written assignments. While efficient techniques for detecting simple copy-paste plagiarism exist and are deployed in Denmark (Frølich and Hansen, 2012), there have recently been an increase in students resorting to ghost-writing, i.e. handing in assignments written by someone else (e.g. a teacher, college student, or other professional). Normal plagiarism control does not detect this kind of fraud, since the assignment is original work. The problem has been highlighted recently by the emergence of so-called *paper-mills* in Denmark; online services providing academic ghost-writing (e.g. www.fixminopgave.dk), a phenomenon already seen in other pars of the world, e.g. the U.S., where paper mills have existed for several years (Tomar, 2014). MaCom wishes to combat this emerging trend by deploying a system for detecting cases of ghost-writing in Lectio. With data for more than 150,000 Danish students, including more than 15 million handed-in assignments, there is large potential.

Formally, we can define the problem of detecting ghost-writing as the *Authorship Verification Problem*: Given an author a, a set of texts $S = \{s_1, s_2, \ldots, s_n\}$ written by a and a text x, determine whether a is the author of x. This problem (and the related problem of Authorship Attribution) has been considered in the literature (Koppel, Schler and Bonchek-Dokow, 2007; van Halteren, 2004; Stamatatos 2009), and is in general considered challenging, since only limited data about the author is available. Most approaches take inspiration from the study of *stylometrics* and employ techniques from natural language processing and machine learning. Approaches usually follow either the *profile-based* paradigm (x is compared to a constructed profile of a) or the *instance-based* paradigm (x is compared to all instances s_1, s_2,...); in either case, the problem can be seen as a case of *Student Profiling*.

For the case of authorship verification in Danish high schools, there are several important considerations to be made. Most notably the fact that:

1. Students are still learning, and their writing style may change over time.
2. We have access to a large corpus of text consisting of the 15 million assignments, which may be utilized in order to improve verification.

These considerations are already investigated in two studies using the data from MaCom: (Hansen et al., 2014) considers authorship attribution with the temporal aspect, and concludes that, using SVMs, good accuracy can

be obtained even when *S* only contains the most recent assignments for a student, while (Aalykke, 2016) utilizes the corpus by performing verification through a profile-based authorship attribution approach.

We hope to improve upon the previous studies by utilizing both (1) and (2), coupled with comprehensive feature selection and state-of-the-art techniques for authorship verification and outlier detection. Furthermore, we hope that our student profiling may also aid in measuring progress of the student in terms of writing style.

2.6 Curriculum trainer

Every week teachers all over the world plan lessons to teach their kids. Much of this work is overlapping in between teachers since what the students need to learn is in many cases the same. Now in the era of "sharing economy" it is possible to utilize the community of teachers via large lecture management systems, such as Lectio, which is made by MaCom. Lectio is used by almost all high schools in Denmark (90%).

Curriculum trainer is a small step in the utilization of taking advantage of "sharing economy". It is a system made for high school students. They can use it to prepare them for their exams. The system is to be implemented in Lectio.

The system is made such that all high school teachers in Denmark (who use Lectio) can pose questions directly into the system. The teacher only has to state the question, the correct answer, some wrong answers and in which subject the question is related to. When the students use the system it adaptively estimates the difficulty of the question and the skill of the student. This is done by the ELO rating from chess (Elo, 1978), where the game is between a question and the student, similar to (Pelanek, 2016) and (Antal, 2013). The student wins if the question is answered correctly and is thus rewarded with an increase in his skill score. The difficulty index of the question is then decreased. How much depend on both the skill level of the student and the difficulty index of the question. The opposite happens if the student answers incorrectly. However unlike in chess the update rules can be different for students and questions, since the skill level of a student is expected to change whereas the difficulty level of a question is expected to be static at least over shorter periods of time, like a school year. The ELO ratings are also used to make sure that students get question at the level which is optimal for their learning. The initialisation of a student's skill level is based on his/her grades in the subject. In the future, log files could be studied to personalize how difficult question should be for a student to keep motivation high. The ELO rating is preferred for its simplicity since we need to minimize the amount of information the teacher needs to give.

The possibility of using "sharing economy" in the teaching scene has huge potential for generating teaching material of high quality, since all teachers can contribute. There are of course some barriers, the most often mentioned is the fear of being judged for their work. Hence starting at a small level and share questions should help overcome this barrier and in this case anonymity could easily be granted.

3. Conclusion

The overall goal of the study was to find a wide array of interesting problems that companies providing e-Education in Denmark have, and present early work in the direction we plan to go to solve them. The uncovered problems are interesting from an educational point of view, but they also raise interesting problems from a computer science perspective, showing the importance of the cooperation between the fields for e-Learning.

None of the presented problems were novel for the e-Learning community, and there are varying levels of existing work that potentially can be used directly by the companies without further improvement. This demonstrate the potential value between industry and university cooperation in the e-Learning community, where a cooperation will help reduce the gap between current state-of-the-art research in academia, and what solutions are currently available for students.

The goal of the educational part of DABAI is to go one step beyond bridging the gap between industry and academia. With access to data generated from more than a million students, the goal is to push the state-of-

the-art in cooperation with the companies, with the advantage of having access to extremely large amounts of real-world data to verify the methods against.

4. References

Aalykke, A. H. (2016), Computational Authorship Attribution in Danish High Schools. Master Thesis. DTU.

Antal, M., (2013) On the use of elo rating for adaptive assessment. Studia Universitatis Babes-Bolyai, Informatica 58

Barnes, T. (2005) The Q-matrix Method – Mining Student Response Data for Knowledge. In American Association for Artificial Intelligence 2005 Educational Data Mining Workshop, pp 1—8.

Berlyne, D. (1960) Conflict, arousal, and curiosity, McGraw-Hill Book Company.

Chen, Y. ,González-Brenes, J. , Tian, J. (2016) Joint Discovery of Skill Prerequisite Graphs and Student Models , Proceedings of Educational Data Mining (EDM), pp 46-53.

Collaborative Filtering 2017, Wikiversity, wiki, 22 May 2017. Available from https://en.wikipedia.org/wiki/Collaborative_filtering. [22 May 2017]

Deary, I., Strand , S., Smith, P., Fernandes, C. (2007) Intelligence and educational, Intelligence 35 (2007) pp 13–21.

Desmarais, M. (2011) Conditions for Effectively Deriving a Q-matrix from Data with Non-negative Matrix Factorization, In Proceedings of the International Conference on Educational Data Mining, pp 41—50.

Elbadrawy, A., Polyzou, A., Ren, Z., Sweeney, M., Karypus, G., and Rangwala, H. (2016) Predicting Student Performance Using Personalized Analytics, Computer Vol. 49, No. 4, pp 61—69.

Elo, A. E. (1978), The rating of chess players, past and present. B.T. Batsford, Ltd., London

Faucon, L.; Kidzinski, L. & Dillenbourg, P. (2016), Semi-Markov model for simulating MOOC students., in Proceedings of Educational Data Mining (EDM), pp. 358-363 .

Frølich, M. and Hansen, K. (2012), Efficient Plagiarism Detection. Master Thesis. DTU.

Gottlieb, J., Oudeyer, P.-Y., Lopes, M., and Baranes, A. (2013) Information-seeking, curiosity, and attention: computational and neural mechanisms, Trends in Cognitive Sciences 17, 11, pp 585–593.

Hansen, C., Hansen, C., Hjuler, N., Alstrup, S., and Lioma, C. (2017), Sequence Modelling For Analysing Student Interaction with Educational Systems. To appear in Proceedings of Educational Data Mining (EDM).

Hansen, N., Lioma, C., Larsen, B. and Alstrup, S. (2014), Temporal context for authorship attribution: a study of Danish secondary schools, in Proceedings of the 7th Information Retrieval Facility Conference (IRFC), pp 22-40.

Khajah, M., Lindsey, R., Mozer, M. (2016) How Deep is Knowledge Tracing? Proceedings of the 9th International Conference on Educational Data Mining pp 94-101

Klingler, S., Käser, T., Solenthaler, B. and Gross. M. (2016), Temporally Coherent Clustering of Student Data., in Proceedings of Educational Data Mining (EDM), pp. 102-109 .

Köck, M. & Paramythis, A. (2011), Activity sequence modelling and dynamic clustering for personalized e-learning., in User Model. User-Adapt. Interact. 21 (1-2), pp 51-97.

Koppel, M., Schler, J. and Bonchek-Dokow, E. (2007), Measuring Differentiability: Unmasking Pseudonymous Authors, Journal of Machine Learning Research Vol. 8, pp 1261-1276.

Koren, Y., Bell, R.-M., and Volinsky, C. (2009) Matrix Factorization Techniques for Recommender System, IEEE Computer, Vol. 42, No. 8, pp 30 – 37.

Lee, D.-D., and Seung H.-S. (2000) Algorithms for Non-negative Matrix Factorization. In Advances in Neural Information Processing System (NIPS), pp 556—562.

Lopes, M., Clement, B., Roy, D., and Oudeyer, P.-Y. (2015) Multi-Armed Bandits for Intelligent Tutoring Systems, Journal of Educational Data Mining (JEDM), Vol. 7, No. 2, pp 20–48.

Pelanek, R. (2016). Applications of the Elo Rating System in Adaptive Educational Systems. In Computers and Education , pages 169-179

Stamatatos, E. (2009), A survey of modern authorship attribution methods, Journal of the American Society for Information Science and Technology Vol. 60, No. 3, pp 538-556.

Tekin, C., Braun, J, and van der Schaar, M. (2015) eTutor: Online Learning for Personalized Education, In: Proceeding of International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp 5545—5549.

Thai-Nghe, N., Drumond, L., Krohn-Gimberghe, A., and Schimidt-Thieme, L. (2010) Recommender System for Predicting Student Performance, In Proceedings of the Workshop on Recommender System of Technology Enhanced Elearning, pp 2811—2819.

Tomar, D. A. (2014), The Ghostwriting Business: Trade Standards, Practices, and Secrets. [online] The Best Schools. Available at: http://www.thebestschools.org/resources/ghostwriting-business-trade-standards-practices-secrets/ [Accessed 24th May 2017]

van Halteren, H. (2004), Linguistic profiling for author recognition and verification, in Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics (ACL), Article No. 199.

Investigating Writing Style Development in High School

Stephan Lorenzen
University of Copenhagen
lorenzen@di.ku.dk

Niklas Hjuler
University of Copenhagen
Hjuler@di.ku.dk

Stephen Alstrup
University of Copenhagen
alstrup@di.ku.dk

ABSTRACT

In this paper we do the first large scale analysis of writing style development among Danish high school students. More than 10K students with more than 100K essays are analyzed. Writing style itself is often studied in the natural language processing community, but usually with the goal of verifying authorship, assessing quality or popularity, or other kinds of predictions.

In this work, we analyze writing style changes over time, with the goal of detecting global development trends among students, and identifying at-risk students. We train a Siamese neural network to compute the similarity between two texts. Using this similarity measure, a student's newer essays are compared to their first essays, and a writing style development profile is constructed for the student. We cluster these student profiles and analyze the resulting clusters in order to detect general development patterns. We evaluate clusters with respect to writing style quality indicators, and identify optimal clusters, showing significant improvement in writing style, while also observing suboptimal clusters, exhibiting periods of limited development and even setbacks.

Furthermore, we identify general development trends between high school students, showing that as students progress through high school, their writing style deviates, leaving students less similar when they finish high school, than when they start.

Keywords

Student clustering, Writing style analysis, Siamese Neural Network, Educational Systems

1. INTRODUCTION

One of the most essential skills, learned during the course of primary, secondary and high school, is writing. While the main focus of primary school are on basic writing skills (such as grammar), secondary or high school will be more focused on improving *the linguistic writing style* of a student, that is, the quality of the written text as perceived by the reader. With many jobs being highly dependent on producing relatively large amounts of well-written text, no justification is needed for why *good writing* is an essential skill.

The definition of quality in linguistic writing style is widely discussed [3, 23]. While correct grammar being a prerequisite, several other measures are also correlated to writing style being perceived as good, for instance use of vocabulary, sentence structure and readability [18]. Our focus in this work will mainly be writing style *development* through the course of high school, while writing style quality will have a secondary role. We consider data from Danish high schools, consisting of Danish essays, and investigate the general development patterns among the students during the three years of study. The end goal is to be able to provide feedback to teachers about the development of their students' writing styles. We identify patterns among thousands of students across different classes and institutions, allowing us to provide teachers with new insights, which the data available to the teacher might not show. For instance insights about students, whose writing style development patterns may be unique within their own classes.

By itself, our method potentially allows for identifying students with deviating writing styles development (which might be good or bad), or students with sudden significant changes in writing style, which could be an indicator of cheating. However, we also consider several measures for the *quality* of writing. We investigate how these measures correlate with the different patterns of writing style development found, as a mean to detect optimal and suboptimal development profiles with respect to text quality. Information of this kind could be used to help teachers tailor their teaching style to specific groups of students, who may need training in specific areas challenging to their development profile.

1.1 Our Contribution

As mentioned, we concern ourselves with the development of linguistic writing style (as opposed to e.g. handwriting) during the course of high school. Specifically, we investigate the development of writing style in Danish essays handed-in by students in Danish high schools [1].

[1]Note, that high school in Denmark usually consists of three years of study with students normally starting at age 15-17 and finishing at age 18-20.

We are interested in determining general patterns of development, and to discuss which of the patterns are optimal, in the sense of improving writing style quality. In particular, we consider the following questions:

- How does the writing style of a student develop, and what are the typical kinds of development in writing style?
- How does writing style changes correlate with measures of quality?
- How does writing style similarity between students behave, with respect to how far the students are in their education?

Our study is based on data from the company MaCom[2], who is behind the learning management system Lectio, a system used by 90% of Danish high schools. Students submit their written essays through Lectio, giving MaCom access to a huge corpus of Danish texts by high school students, marked with author and date of submission.

Our approach is based on methods from authorship verification; in order to learn a similarity measure for writing style, we consider examples of writing styles in texts from the same or different authors, similar to how it is done in verification tasks. We use a Siamese neural network for learning this similarity measure. While training, time is not taken into account. Assuming that writing style actually changes over time, this will lead to a suboptimal network. However, testing the network, we see clear patterns in how the "errors" distribute for a single author, indicating that the network simulates the best similarity measure possible, and the "errors" are actual changes in writing style. Using this method, writing style development profiles are generated and clustered for a large set of students. Analyzing the clusters, we see optimal and suboptimal types of development. In general, the average similarity is found to decay with time to a great extend, which corresponds well with the general perception, that writing style changes during high school, and also matches conclusions made in the literature [4, 9, 25].

While this paper presents a case study of the data from MaCom, the methods used for analysis are of independent interest, and not specific to the Danish language or high school, except for the neural network, which would at least require retraining in the given language. Considering other network architectures than the one used in this work, might also improve upon the analysis, see for instance [19] for a network used with English.

1.2 Related Work

Writing style analysis, in one way or another, has been studied in the natural language community for many years. Typically, the analysis of writing style is used as a middle link for tasks such as *authorship verification* [19, 24, 25], in which a text of unknown authorship is given, together with a set of texts by some known author, and we wish to verify, whether the given author is the author of the unknown text. Similarly, in *authorship attribution* the unknown text must be attributed to one of several known authors. Traditional methods for verification and attribution utilize both unsupervised methods from the field of outlier detection [24], as well as standard supervised learning techniques, such as SVMs [9] and techniques based on neural networks [19, 25].

Other uses of writing style analysis include distinguishing features of the writer (e.g. sex and age [1, 17, 21, 22], demographics [2], or nationality [12]), using supervised learning algorithms such as SVMs, random forest, and neural networks. Other studies have investigated written conversations on online forums, trying to infer whether one person is trying to convince another [8].

Some studies investigate the quality of writing, for instance prediction of popularity of news articles [27], or the quality of scientific articles [14]. The former uses the popularity of an article on social media as a measure of quality, while the quality measure of scientific papers considered in the latter is based on acceptance of a paper to "The Best American Science Writing", an anthology of popular science articles published in the United States on a yearly basis.

Few studies consider development of writing style as the main objective. [5] uses neural network models to track style of *handwriting* (i.e. not linguistic writing style) and investigate the development of handwriting among young students, and how similar it is when compared to different students, in the same/different grade level. [3] shows how students in higher grades get higher scores for their essays from teachers, in a blind experiment, where all student information is hidden from the grading teacher. [4] considers two famous Turkish writers, investigating their change in writing style over time, the most significant finding being average word length increasing with the age of the author.

Finally, several studies related to writing style have been conducted using the data available from MaCom. [9] investigates temporal aspects of authorship attribution, and concludes that considering more recent essays improves authorship attribution algorithms, indicating that the writing style among high school students does indeed change with time. [25] also uses the MaCom data for testing their neural network based authorship verification methods; their results also support these findings.

2. METHODS AND SETUP

This section describes our experimental setup and methods. We start by giving some basic notation.

We consider a set of students $\mathcal{A}$, and let $\alpha \in \mathcal{A}$ denote a single student with texts $t \in T_\alpha$. Furthermore, let $T = \cup_{\alpha \in \mathcal{A}} T_\alpha$ denote the entire corpus of texts.

Since our main focus is how the writing style of a student develops during the time they spend in high school, we are interested in computing a similarity function $s : T \times T \to [0, 1]$, allowing us to compare the writing style between two texts. As mentioned, we utilize a Siamese neural network to compute s; this approach is widely used for computing writing style similarity [7, 19, 25], specifically, our network

[2]The data set is proprietary and not publicly available

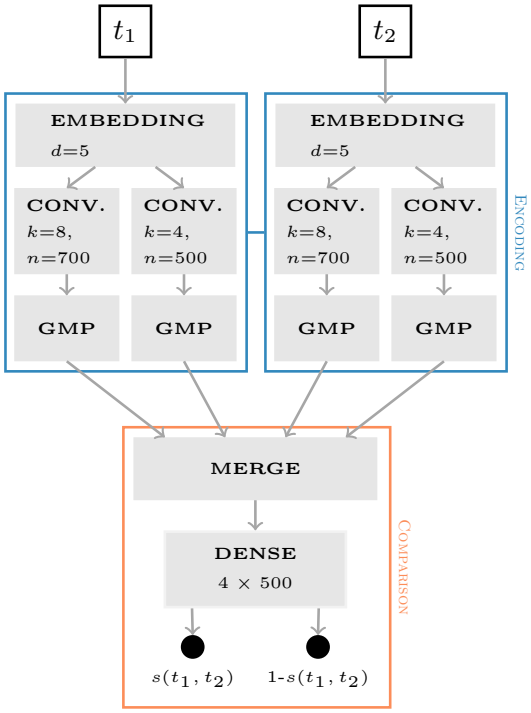


Figure 1: Network architecture.

will be similar to that of [25]. Section 2.1 will describe our network in detail.

The similarity measure s found will then be utilized for writing style analysis. Primarily, we will focus on determining development patterns by generating a *writing style development profile* P_α for each student α. These profiles are then clustered and analyzed with respect to different measures for text quality. The profile generation and clustering are described in more detail in Section 2.2.

Finally, we also explore how the similarity between random students change depending on their current progress through high school. This is done by sampling random pairs of texts $t_1 \in T_\alpha, t_2 \in T_\beta$ and computing their similarity. We then consider how the similarity changes depending on if α and β are in the same grade or not.

2.1 Text Similarity using a Siamese Neural Network

As mentioned, we use a Siamese neural network for computing the similarity $s(t_1, t_2)$ between two texts t_1 and t_2. We considered several different architectures, using different input channels (e.g. char, word, part of speech tags). These architectures were evaluated using a validation set (see Section 3.1), and the best architecture was selected, as shown in Figure 1. The network relies only on character level inputs.

The basic philosophy behind the network is to a) *encode* the two texts in some space using a replicated encoder network with shared weights, and b) *compare* the two texts in this space.

- The encoder network in the **encoding** part of our network consists of a character embedding (using ReLu activation functions), followed by two different convolutional layers (**CONV**): one using kernel size $k = 8$ and $n = 700$ filters, and one using $k = 4$ and $n = 500$, each followed by global max pooling layers (**GMP**).
- In the **comparison** part of the network, the **MERGE** layer first computes the absolute difference between the outputs of the two encoder networks. Afterwards, four dense layers (**DENSE**) with 500 neurons each are applied, using ReLu for activation function and with a dropout of 0.3. Finally a two neuron softmax layer is used to normalize the output.

Using the convolutional layers, the network extracts character n-grams. Specifically, it compares 8- and 4-grams. Character n-grams have been shown to be an important feature in writing style analysis tasks such as authorship attribution [24]. We did also consider architectures using recurrent networks, however none of them performed as well as convolutional networks.

2.2 Student Profiling and Clustering

As mentioned, we construct writing style development profiles for the students, in order to analyze the general development patterns. The profile P_α for student α is constructed by first determining their initial writing style. The natural way to do so, and indeed our approach, is to consider their early work. One or more texts may be used to represent the initial writing style, as a trade off between the amount of data available for the profile and the robustness of the initial writing style estimation. P_α then consists of a chronologically ordered sequence of similarities, between any $t \in T_\alpha$ and this initial writing style. More specifically, if $t_1, t_2, ..., t_{|T_\alpha|}$, $t_i \in T_\alpha$ is a chronologically ordering of T_α, we compute the similarity p_i between t_i and the initial writing style by:

$$p_i = \frac{1}{m} \sum_{j=1}^{m} s(t_i, t_j),$$

where m is the number of texts used for representing the initial writing style. Since the first m texts are part of the initial writing style, $p_1, p_2, ..., p_m$ are not independent, and thus we exclude the first $m-1$ texts, and re-index such that $p_j = p_{i-m+1}$. Furthermore, for each text, we let τ_j denote the time in months since t_m was written, i.e. the time since p_0, with $\tau_0 = 0$. Now, the final profile becomes the sequence consisting of pairs (τ_j, p_j) of length $|T_\alpha| - m + 1$. Note that the profile now describes a curve.

These profiles are now clustered using a slightly modified k-means clustering. Before clustering, for each profile P_α, an approximate profile $\hat{P}_\alpha$ is constructed by interpolating values between any two consecutive pairs (τ_j, p_j) and (τ_{j+1}, p_{j+1}), in intervals of 0.05 months. Thus $\hat{P}_\alpha$ becomes a vector $\hat{P}_\alpha \in [0, 1]^{\ell_\alpha}$ consisting of similarities for every 0.05 month, with length ℓ_α.

These approximate profiles are then clustered. The clustering is complicated by profiles having variable length: $\hat{P}_\alpha$ has length ℓ_α depending on $\tau_{|T_\alpha|-m+1}$ (the time span between

t_m and t_{T_α}), specific to α. Hence, distance computation used in the clustering algorithm is modified slightly; we compute the distance $dist(\hat{P}_\alpha, \hat{P}_\beta)$ between two profiles $\hat{P}_\alpha$ and $\hat{P}_\beta$ by computing the Euclidean distance between the prefixes of length $\ell = \min\{\ell_\alpha, \ell_\beta\}$ of the two profiles:

$$dist(\hat{P}_\alpha, \hat{P}_\beta) = dist_E(\hat{P}_\alpha[1...\ell], \hat{P}_\beta[1...\ell]),$$

where $dist_E$ denotes the Euclidean distance, and $v[1...n]$ denotes the prefix of length n of vector v.

Similarly, when computing centroid C_r for cluster $\mathcal{C}_r$, profile $\hat{P}_\alpha$ contributes only to the ℓ_α first entries of C_r. Thus, with $\mathcal{C}_r^j = \left\{\hat{P}_\alpha | \hat{P}_\alpha \in \mathcal{C}_r, \ell_\alpha \leq j\right\}$, the j'th entry of C_r is then computed as:

$$C_r[j] = \frac{1}{|\mathcal{C}_r^j|} \sum_{\hat{P}_\alpha \in \mathcal{C}_r^j} \hat{P}_\alpha[j]$$

where $v[j]$ denotes the j'th entry of vector v.

The clustering is initiated by selecting k profiles at random as the initial clusters, and then continually reassigning profiles and recomputing centroids for clusters. Having reassigned the profiles, the $E_\mathcal{C}$ is computed:

$$E_\mathcal{C} = \frac{1}{|\mathcal{A}|} \sum_{r=1}^{k} \sum_{\alpha \in \mathcal{C}_r} dist(\hat{P}_\alpha, C_r)$$

The algorithm iterates until the change in cluster error $E_\mathcal{C}$ is sufficiently small ($E_\mathcal{C} \leq 10^{-6}$), or until a set number of maximum iterations (100) is reached.

Selecting the number of clusters k is an inherent problem in all unsupervised learning task. One approach is to base the decision on domain knowledge, and select the "right" number of clusters. We will instead make use of the so called *elbow heuristic* which relies on looking at how the error decreases with the number of cluster and pick at the "elbow" in the resulting curve [26].

Having determined the parameter k and found k clusters, we compute a few statistics and writing quality indicators for each cluster. Specifically, we will compute the average *noun and verb phrases*, defined as the ratio between nouns and sentences, and the ratio between main verbs and sentences respectively. These measures, especially verb phrases, have been shown to correlate well with readability, which correlates with text quality [18]. Furthermore, we compute the *simple measure of Gobbledygook* (SMOG) grade [16], a measure estimating the grade level required for understanding the text. The SMOG grade is computed as:

$$\text{SMOG} = 1.0430\sqrt{\frac{30 n_{w*}}{n_s}} + 3.1291,$$

where n_{w*} is the number of words of 3 or more syllables, and n_s is the number of sentences [16].

Note that the study showing correlation between noun and verb phrases, and readability, is done on English texts. The SMOG grade as well is defined with the purpose of evaluating English texts. Hence, one must be careful when basing conclusion on these measures when used on Danish. However, we believe they can still provide information about the development, even if the exact computed value might be hard to interpret.

3. EXPERIMENTS AND RESULTS

In this section, we present the data, the experimental setup, and the results obtained. Section 3.1 presents the data, how it is preprocessed and split for training and analysis and some basic statistics. Section 3.2 describes the training of the Siamese neural network, while Section 3.3 describes the clustering and shows the resulting clusters.

3.1 Data

The full data set made available to us by MaCom contains around 130K essays by approximately 10K students, with an average length of about 6K characters. The data set was cleaned by removing very short (≤ 400) and very long ($\geq 30,000$) texts, in order to get rid of outliers/invalid essays (blank hand-ins, garbled texts, etc.). Furthermore, proper pronouns were substituted with placeholder tokens and the first 200 characters of each text were removed, in an effort to remove any data identifying the real author of the text, as such clues could be picked up by the neural network and lead to overfitting. Finally, authors with less than 5 texts were removed. Following this cleaning, the data set contains a total of 131,095 Danish essays, written by 10095 authors, with an average 13.0 texts per author, and an average text length of 5894.8 characters.

We partition the clean data into two author disjoint sets: $T_{network}$ used for training the neural network, and $T_{analyze}$, which we analyze using the trained similarity function. $T_{network}$ is further split into a training set T_{train} and a validation set T_{val} (also author disjoint), used for early stopping when training the network. As the analysis relies heavily on a strong similarity function, the majority of the data (around two thirds) is used for $T_{network}$. The exact sizes are given in Table 1.

Data set	#students	#texts	#Sim
T_{train}	5418	70432	934720
T_{val}	989	12997	173536
$T_{analyze}$	3688	47666	N/A
Total	10095	131,095	1108256

Table 1: Data set overview. The table lists the number of students and texts, as well as the number of problem instances #Sim for training the Siamese neural network.

Data for network training

For training and evaluating the Siamese neural network, we require problem instances consisting of a pair of texts, and a label indicating whether they are by the same author (positive sample) or by different authors (negative sample). We refer to these instances as Sim-instances, and generate them for the training set T_{train} and the validation set T_{val}.

Positive Sim-instances are generated by using $t_i, t_j \in T_\alpha$ with $i \neq j$, while negative instances are generated by using $t_i \in T_{\beta_1}$ and $t_j \in T_{\beta_2}$, where i, j, β_1, β_2 are selected at random, with $\beta_1 \neq \beta_2$. A balanced 50:50 data set is generated by generating the maximum number of positive instances

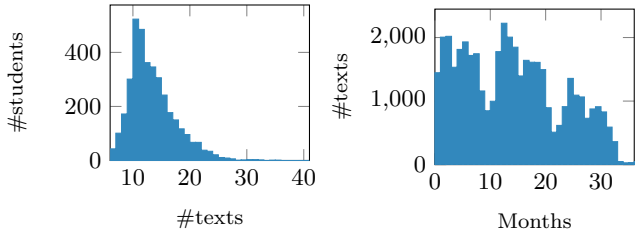


Figure 2: Statistics for $T_{analyze}$. Distribution of students according to number of essays written (left) and total number of essays written at any time during a students stay in high school (right).

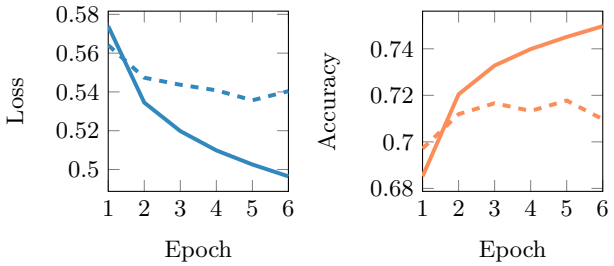


Figure 3: Plot of training (solid) and validation (dashed) loss (left) and accuracy (right), the latter computed by rounding the output. Minimum validation loss was obtained at epoch 5.

for each student, and an equal number of negative instances. The final numbers of SIM-instances for T_{train} and T_{val} are shown in Table 1.

Note, that in generating these samples, we assume all claimed authors in the data to be the real authors; in reality, several students may use ghostwriters or plagiarism, in which case the labels will be wrong. However, we expect that the number of invalid labels is low.

Data for clustering and analysis

The clustering is performed on the remaining data in $T_{analyze}$. Each data point consists of a single student and their texts. As mentioned, an author has around 13 texts in average, distributed over three years; the actual distribution is shown in Figure 2 (left)[3].

Figure 2 (right) shows the number of essays handed in during the three years of high school. The summer vacations are clearly visible in the plot. Note also, that the number of hand-ins drops during the third year. A few students spend more than three years (not shown in the figure), but as only a few students hand-in after 30 months, we consider only the data within 30 months in the experiments[4].

3.2 Neural Network Training

The similarity network described in Section 2.1 was implemented using TensorFlow. We generate SIM-instances for T_{train} and T_{val}, and optimize the network for cross entropy using the Adam optimizer. The final network obtains a training loss of 0.5026 and a validation loss of 0.5357. Rounding the computed similarity to 0 or 1, we can compute an accuracy of 0.7451 for the training set and an accuracy of 0.7178 for the validation set. Figure 3 shows a plot of the loss and accuracy, as the network was trained.

3.3 Clustering

Using the similarity network to compute the similarity function s, we construct profiles as described in Section 2.2. We found that using $m = 2$ texts for determining the initial writing style yielded good results. Thus, profile P_α consists of $|T_\alpha| - 1$ pairs (τ_j, p_j) with:

$$p_j = \frac{s(t_{j+1}, t_1) + s(t_{j+1}, t_2)}{2}.$$

and τ_j being the time in months since hand-in of t_2. With a single profile constructed per student, the total number of profiles is equal to the number of students, as given in Table 1. As mentioned, the lengths of the profiles depend on the number of texts written during the time, they spend in high school. Thus the distribution of the lengths of profiles follows that presented in Figure 2 (left).

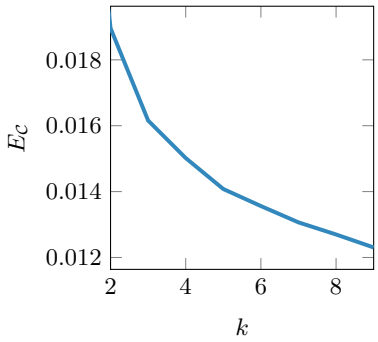


Figure 4: The cluster error E_C obtained for various values of k.

We now apply the elbow method in order to determine the optimal number of clusters k. We compute a clustering for $k = 2, 3, ..., 9$, and plot the resulting cluster error E_C in Figure 4.

Based on Figure 4, we select $k = 5$, as the curve flattens considerable for $k = 6$. The final clustering is performed, obtaining five clusters: $\mathcal{C}_1$, $\mathcal{C}_2$, $\mathcal{C}_3$, $\mathcal{C}_4$, and $\mathcal{C}_5$, with a cluster error of $E_C = 0.01407$. The curves representing the final clusters are shown in Figure 5, while Table 2 lists the number of members in each cluster. Furthermore, we sampled two million random pairs of texts with random (different) authors, and computed the similarity for these samples, obtaining an average of 0.3470. This average is also plotted in Figure 5, while the similarity with respect to time is plotted as a heat map in Figure 7.

Finally, Figure 6 shows a more detailed view for each cluster.

[3]Recall that, students with less than 5 essays is not considered in this study.

[4]The time span considered is smaller than three years (36 months), since we measure the time from first hand-in until the last. Combining this with vacation and finals, most students appear to only be active within the 30 month period.

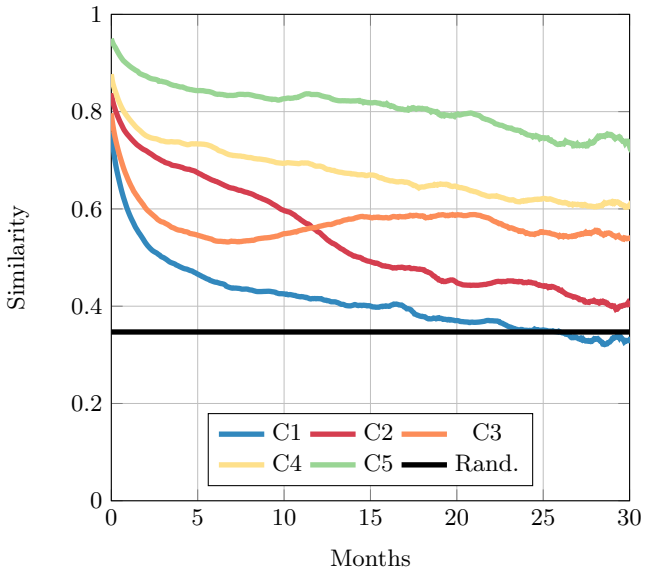


Figure 5: The curves representing the five clusters found. The average similarity between random texts by different students is also plotted.

Cluster	#students
$\mathcal{C}_1$	603
$\mathcal{C}_2$	720
$\mathcal{C}_3$	884
$\mathcal{C}_4$	969
$\mathcal{C}_5$	512

Table 2: The number of students in each cluster.

The similarity curve plots include a plot of the middle 90% of profiles in each cluster. The SMOG score, the noun and verb phrases, and the average text length (in words) are also plotted, as indicators for writing quality changes for the given cluster, see also Section 2.2.

Note, that in visualizing and inspecting the clusters, we consider only data until 30 months, since, as mentioned, only few students are active after 30 months, and the number of data points contributing to that part of the cluster curve becomes small.

4. ANALYSIS AND DISCUSSION

This section presents our analysis and discussion of the five clusters found. Section 4.1 describes and discusses the characteristics of each cluster. Section 4.2 discussed how similarity between random students behaves with time.

4.1 Cluster Analysis

When analyzing the clusters, three properties are important in order to understand a cluster: the initial value of the curve, the shape of the curve, and the total change from start to end. The *initial value of the curve* describes the similarity between the second text of a student and their initial writing style, which is based on the first two texts of the student. Thus a smaller initial value indicates a high initial variance in writing style, which could be an indication of a developing writing style. The *shape of the curve* describes the rate of change in writing style. And finally the *total change* tells us how much the writing style has evolved.

While the similarity curves themselves give no information about *the quality* of the writing, we will use the indicators of writing style, described in Section 2, in the discussion: the SMOG grade, and the noun and verb phrases per sentences. For each of these indicators, the average curves for each cluster are plotted in Figure 6.

Before discussing each cluster in detail, we note some patterns common for all clusters. Across all clusters, it seems the number of words written increases (with the exception of $\mathcal{C}_5$), and the increase seems to be correlated with the corresponding decrease in similarity. Furthermore, on average, students in all clusters appear to be improving with respect to the quality metrics. While positive, some clusters see a smaller increase than others, indicating that these clusters represents suboptimal development profiles. Finally, we note for the SMOG grade, that the maximum increase, occurring in $\mathcal{C}_1$, is only slightly above 1, which might not seem impressive across three years. However, as discussed in Section 2.2, the SMOG grade is a measure designed for readability of English texts, and thus may not be entirely accurate for Danish texts.

Below follow detailed descriptions of each cluster:

$\mathcal{C}_1$ The initial similarity of $\mathcal{C}_1$ is the lowest among the clusters found. Furthermore, the similarity drops rapidly during the first year, and continues the decline, leading to $\mathcal{C}_1$ having the lowest final similarity with the initial writing style, among all the clusters. In fact, the similarity between the first and the last assignments for students in this cluster is so low, that they could just as well have been written by different students, as can be seen when comparing to the average similarity between random students plotted in Figure 5. Thus, $\mathcal{C}_1$ contains students with a significant change in writing style, happening mostly during the first year of high school. Considering the other metrics plotted in Figure 6, we first note the increase in number of words written, as it is particular extreme in the case of $\mathcal{C}_1$, increasing by almost a factor 2 from start till end. This increase also helps explain the decrease in similarity in two ways: a) length is itself a part of writing style recognized by the network, and b) it seems that writing style changes is correlated with when you start writing more. Looking at the SMOG grade, we see an overall large increase, indicating that the students of $\mathcal{C}_1$ does indeed improve, especially compared to the other clusters. Nouns and verbs per sentence are also both increasing, which also indicates that the students in this cluster write longer sentences.

$\mathcal{C}_2$ The initial value of cluster $\mathcal{C}_2$ of about 0.84 (third highest among the clusters) indicates an initial low variance in writing style, but the following drop to about 0.4 is quite significant, indicating that the writing style of students in $\mathcal{C}_2$ change a lot during high school, similar to $\mathcal{C}_1$. However, where $\mathcal{C}_1$ had a sudden drop in

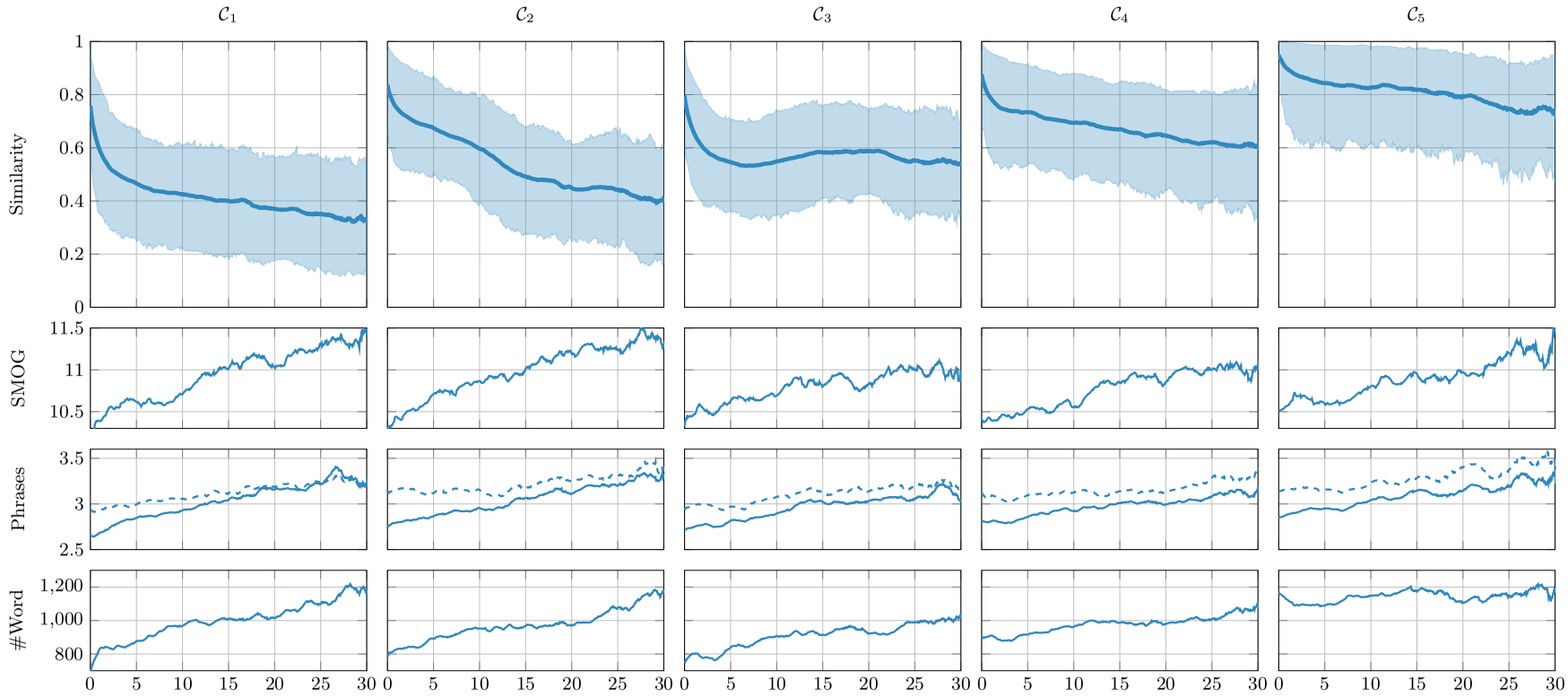


Figure 6: Detailed plots of the five clusters found. The top plot shows the similarity curve, with the middle 90% of profiles in each cluster plotted as well. The second plot from the top shows the development of the SMOG grade, while the third plot shows noun (solid) and verb (dashed) phrases. Finally the bottom plot shows the development of number of words written.

similarity, the change in $\mathcal{C}_2$ is more constant.

Considering the other metrics, we see the number of words written is increasing from about 800 to almost 1200, while the SMOG grade is again showing a large increase from about 10.3 to 11.3. Noun and verb phrases see modest increases. All in all, the metrics indicate a good development of writing style among students in $\mathcal{C}_2$, similar to $\mathcal{C}_1$. However, the more gradual change in similarity of $\mathcal{C}_2$ is preferable to that of $\mathcal{C}_1$, as the development does not stagnate already after the first year.

$\mathcal{C}_3$ After a significant initial drop in similarity in $\mathcal{C}_3$, the similarity actually increases again after the first year, showing the students in this cluster actually reverts to writing style more similar to their original work, before dropping a bit again in the last months. This corresponds well with a smaller improvement in e.g. SMOG grade (around 0.5) compared to the other clusters.

The setback seems to start around the first summer vacation. While not necessarily bad (as students could be reverting back from a worsened writing style), the increase in similarity could indicate reverting to a worse writing style. As such, students in $\mathcal{C}_3$ may be at risk. Many remedies for helping these students could be imagined, from simply encouraging the student to write during their vacation, to going to summer school.

$\mathcal{C}_4$ The similarity of $\mathcal{C}_4$ drops slightly at first, but then decreases slowly at a constant pace, until it reaches a similarity of about 0.6. The total change is smaller than several of the other clusters, as is the improvement in both SMOG grade and noun/verb phrases, indicating that students in the cluster improve less than students in e.g. $\mathcal{C}_1$ and $\mathcal{C}_2$.

This indicates suboptimal development among students in $\mathcal{C}_4$; while we do not see students reverting back, as in $\mathcal{C}_3$, the lower increase in SMOG grade is alarming, indicating students in this cluster may be at risk, and in need of extra attention or encouragement. As for $\mathcal{C}_3$, the total number of words also increases only slightly at a steady pace, from around 900 to 1100.

$\mathcal{C}_5$ Cluster $\mathcal{C}_5$ seems quite distinct from the other clusters. Most notably, students in this cluster have the highest initial similarity, while also decreasing the least amount, ending with a very high similarity of about 0.75. Furthermore, the number of words written is quite high and remains fairly stable, which is quite different from the other clusters, and might be part of the reason the decrease in similarity is as low as it is. Despite the fall in similarity being so low, we still see an increase in SMOG score from about 10.5 to a bit below 11.5, indicating that students are, in fact, improving. A similar pattern occurs for the noun and verb phrases.

The higher-than-average initial SMOG grade and number of words written, indicates that students in $\mathcal{C}_5$ are the initially strong students. While they do develop their writing style, they do not improve as much as students in $\mathcal{C}_1$ and $\mathcal{C}_2$; this could be an indication, that schools do not manage to properly encourage/teach students, who are initially strong.

While not included in the plots, we also investigated several other metrics for the clusters and the set of students in general. Most notably, the average word length increases with time for all clusters. A similar trend was seen in [4], although the study was in a very different setting and time frame.

Summarizing the clusters, the development in SMOG grade was greatest for $\mathcal{C}_1$ and $\mathcal{C}_2$, making those clusters appear the most beneficial for writing style development. While students in $\mathcal{C}_5$ also increased their SMOG grade, they started higher than the students in the other clusters, and did not manage to improve as much as $\mathcal{C}_1$ and $\mathcal{C}_2$. As to $\mathcal{C}_3$ and $\mathcal{C}_4$, they seem to be suboptimal with regards to writing style development, and students in these clusters may need attention.

Looking at Table 2, we see that $\mathcal{C}_3$ and $\mathcal{C}_4$ are the largest individually, indicating that quite a few students are exhibiting suboptimal writing style development. However, the majority of students included in our data are located in $\mathcal{C}_1$, $\mathcal{C}_2$ and $\mathcal{C}_5$, indicating optimal or at least fair development through high school.

4.2 Investigating Similarity Between Random Students

As mentioned, we also investigated how the similarity develops between different students, across the time spent in high school. Based on roughly 2 million sampled text pairs from different students, we computed the average similarity between random students to be 0.3470. As seen in Figure 5, the similarity observed among students in $\mathcal{C}_1$ actually drops below this value. This motivated a further investigation of how the similarity between different authors behave on average, conditioned on how long time they each have spend in high school. Based on the samples, we constructed the heat map shown in Figure 7.

The plot shows students starting out similar in writing style and then becoming less similar as time passes. The most surprising thing to notice is that a student in their first year and a student in their third year are equally or even more similar in writing style on average, compared to two different students in their third year. One explanation could be that the initial space of possible writing styles start out small and grows as students are educated, i.e. writing styles among students coming from primary school are fairly similar, but grow more diverse during high school. One would expect some writing styles to diminish or even disappear, but from this data it looks like more new and diverse writing styles develop, than disappear. And not only that; the amount of possible directions for the writing style to develop is so large, that we see first and third year students as equally or more similar on average, than two students both within their third year.

Education is sometimes accused of destroying individuality and/or creativity; these findings indicate the opposite to such claims, at least in regards to writing style.

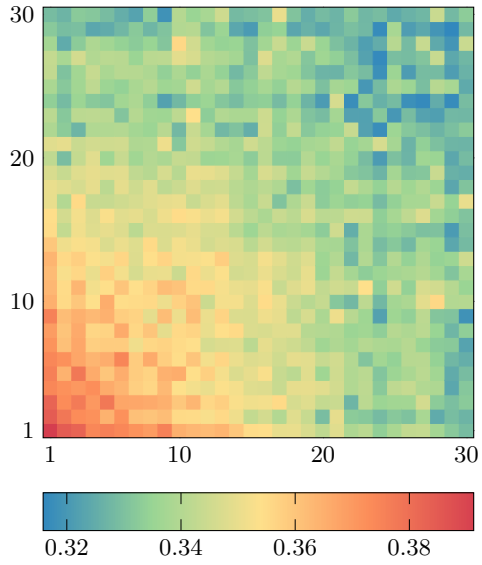


Figure 7: Heat map showing the average similarity between different authors, depending on how long time the two authors have been in high school.

5. CONCLUSIONS AND FUTURE WORK

We trained a Siamese neural network to be able to tell people apart by their writing, and used this network as a similarity function for analyzing the development of writing style in Danish high schools. Writing style development profiles were constructed for 3688 students, and five clusters were found and discussed. Based on quality indicated by noun/verb phrases and SMOG grade, two were found to be optimal, while three were found to be suboptimal, especially two clusters exhibited limited improvement.

The optimal clusters both exhibited a large degree of change in writing style, although at different rates, while the suboptimal clusters showed less development, with one cluster even reverting back to an earlier writing style. The setback in similarity occurred around the summer vacation after the first year. The effect of summer vacation on student learning is highly discussed topic among researchers, teachers, and parents [15]; in the case of the found cluster, the effect appears to be negative.

One tendency, we saw in all clusters, was that writing style changed more when students start writing more words in their essays. It does not seem surprising that your writing style changes as you write more, but it could be an indication of even more: writing style changes, when students are pushed out of their comfort zone, i.e. in the end of their assignments, when they write more than what they usually do. It could be interesting to investigate the scenario, where a student starts writing longer texts: does changes in writing style occur in the entire text, or only near the end, where the student is literally writing more than before?

Furthermore, we saw from Figure 7 how students become less alike, as they go through high school. Specifically, we saw how first year and third year students had higher or equal writing style similarity than two students both in third year, indicating that as Danish students go through high school, their writing styles diverge and become more individual.

5.1 Future Work

It is easy to pose several new questions based on the clusters found and the conclusions made above.

With regards to improving the analysis, using different quality measures tailored to Danish instead of SMOG would be interesting. Another way would be to consider the grades given to the students (as many essays in Danish high school are graded individually), although good writing style is only a requirement for a good grade, but not sufficient.

As mentioned above, one could also consider a more fine grained analysis, by investigating style changes within texts, and maybe even being able to pinpoint exactly where in a text the writing style develops/changes. One could easily imagine drawing inspiration from studies of style breach detection [10, 11, 20].

One could also investigate prediction of writing style development, possibly based on the methods used in this study. This would allow for an early warning system, allowing identification of at-risk students, e.g. students likely to have a setback in writing style due to summer vacation.

The methods used in this study build upon methods used for authorship verification, in which Siamese networks are utilized directly in order to verify authorship [25]. While a sudden deviation in writing style could be an indication of a ghost writer, detecting these reliably using our method will probably not be able to compete with the more direct methods. However, the results obtained here could potentially be used to improve authorship verification techniques, with respect to the fairness perspective: The fact, that the clusters found show such different similarity development, is of interest from a fairness perspective. Fairness is a general issue in machine learning algorithms where the predictions have severe consequences [6, 13]. In the setting of ghost writing detection in high school it is extremely difficult to get non-artificial negative samples and even guaranteeing correctness of labels is rare in large scale data sets. Which makes fairness even more difficult to measure than usual. It could be interesting to check that clusters such as $\mathcal{C}_5$, which would be the cluster most likely to be classified as a false negative, have a representative distribution in regards to gender, race, social status, etc.

Another interesting course of study, would be to further investigate the fact that students seem to become less similar during high school. It could be interesting to pursue this on a larger timescale, perhaps all the way from primary school and on through college. Another take could be to look at how similarity in writing among people behaves with age after they have finished their education. Will the trend continue?

Finally, one could investigate how similarity in writing develops among the genders. Several studies have shown, with some success, that gender can be predicted from writing [17, 21, 22], but no one has settled whether this is due to bi-

ology or environment. One could try to answer this question by looking at how similarity in writing style changes with age, while considering three groups: female-female, male-male, female-male. If the cross gender similarity changes faster than same gender similarity, it would be an indication that the differences in writing style are taught, more than it is something you are born with.

6. ACKNOWLEDGMENTS

The work is supported by the Innovation Fund Denmark through the Danish Center for Big Data Analytics Driven Innovation (DABAI) project. The authors would like to thank MaCom for the cooperation.

7. REFERENCES

[1] Shlomo Argamon, Moshe Koppel, Jonathan Fine, and Anat Rachel Shimoni. Gender, genre, and writing style in formal written texts. *TEXT*, 23:321–346, 2003.

[2] Shlomo Argamon, Moshe Koppel, James W. Pennebaker, and Jonathan Schler. Automatically profiling the author of an anonymous text. *Commun. ACM*, 52(2):119–123, February 2009.

[3] Paul B. Diederich. Measuring growth in english. 01 1974.

[4] Fazli Can and Jon M. Patton. Change of writing style with time. *Computers and the Humanities*, 38(1):61–82, Feb 2004.

[5] Jun Chu and Sargur Srihari. Writer identification using a deep neural network. In *Proceedings of the 2014 Indian Conference on Computer Vision Graphics and Image Processing*, ICVGIP '14, pages 31:1–31:7, New York, NY, USA, 2014. ACM.

[6] Sam Corbett-Davies, Emma Pierson, Avi Feller, Sharad Goel, and Aziz Huq. Algorithmic decision making and the cost of fairness. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, pages 797–806, New York, NY, USA, 2017. ACM.

[7] William Weidong Du, Michael Fang, and Ming Shen. Siamese convolutional neural networks for authorship verification. https://www.semanticscholar.org/paper/Siamese-Convolutional-Neural-Networks-for-Du-Fang/c5d1c54511d7f688963cd29a8556d0cf02595890, 2017. Accessed April 2019.

[8] Marjorie Freedman, Alex Baron, Vasin Punyakanok, and Ralph Weischedel. Language use: What can it tell us? In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: Short Papers - Volume 2*, HLT '11, pages 341–345, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics.

[9] Niels Dalum Hansen, Christina Lioma, Birger Larsen, and Stephen Alstrup. Temporal context for authorship attribution: a study of Danish secondary schools. In *Multidisciplinary information retrieval*, pages 22–40. Springer, 2014.

[10] Daniel Karaś, Martyna Śpiewak, and Piotr Sobecki. OPI-JSA at CLEF 2017: Author Clustering and Style Breach Detection—Notebook for PAN at CLEF 2017. In *CLEF 2017 Evaluation Labs and Workshop – Working Notes Papers, 11-14 September, Dublin, Ireland*. CEUR-WS.org, September 2017.

[11] Jamal Ahmad Khan. Style Breach Detection: An Unsupervised Detection Model—Notebook for PAN at CLEF 2017. In *CLEF 2017 Evaluation Labs and Workshop – Working Notes Papers, 11-14 September, Dublin, Ireland*. CEUR-WS.org, September 2017.

[12] Moshe Koppel, Jonathan Schler, and Kfir Zigdon. Determining an author's native language by mining a text for errors. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, KDD '05, pages 624–628, New York, NY, USA, 2005. ACM.

[13] Joshua R. Loftus, Chris Russell, Matt J. Kusner, and Ricardo Silva. Causal reasoning for algorithmic fairness. *CoRR*, abs/1805.05859, 2018.

[14] Annie Louis and Ani Nenkova. What makes writing great? first experiments on article quality prediction in the science journalism domain. *Transactions of the Association for Computational Linguistics*, 1:341–352, 2013.

[15] Andrew Mceachin and Allison Atteberry. The impact of summer learning loss on measures of school performance. *Education Finance Policy*, 12, 05 2016.

[16] Harry G. McLaughlin. SMOG grading - a new readability formula. *Journal of Reading*, pages 639–646, May 1969.

[17] Claudia Peersman, Walter Daelemans, and Leona Van Vaerenbergh. Predicting age and gender in online social networks. In *Proceedings of the 3rd International Workshop on Search and Mining User-generated Contents*, SMUC '11, pages 37–44, New York, NY, USA, 2011. ACM.

[18] Emily Pitler and Ani Nenkova. Revisiting readability: A unified framework for predicting text quality. In *EMNLP 2008*, 2008.

[19] Chen Qian, Tianchang He, and Rao Zhang. Deep learning based authorship identification. https://www.semanticscholar.org/paper/Deep-Learning-based-Authorship-Identification-Qian-He/ab0ebe094ec0a44fb0013d640b344d8cfd7adc81, 2018. Accessed April 2019.

[20] Kamil Safin and Rita Kuznetsova. Style Breach Detection with Neural Sentence Embeddings—Notebook for PAN at CLEF 2017. In *CLEF 2017 Evaluation Labs and Workshop – Working Notes Papers, 11-14 September, Dublin, Ireland*. CEUR-WS.org, September 2017.

[21] Kosgi Santosh, Romil Bansal, Mihir Shekhar, and Vasudeva Varma. Author profiling: Predicting age and gender from blogs - notebook for pan at clef 2013. In *CLEF*, page 10, 2013.

[22] Anat Rachel Shimoni, Moshe Koppel, and Shlomo Argamon. Automatically Categorizing Written Texts by Author Gender. *Literary and Linguistic Computing*, 17(4):401–412, 11 2002.

[23] V. Spandel. *Creating Writers: Through 6-trait Writing Assessment and Instruction*. Longman, 2001.

[24] Efstathios Stamatatos. A survey of modern authorship attribution methods. *J. Am. Soc. Inf. Sci. Technol.*, 60(3):538–556, March 2009.

[25] Magnus Stavngaard, August Sørensen, Stephan Lorenzen, Niklas Hjuler, and Stephen Alstrup.

Detecting Ghostwriters in High Schools. In *27th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, 2019.

[26] Robert L. Thorndike. Who belongs in the family? *Psychometrika*, 18(4):267–276, Dec 1953.

[27] Yuting Yang, Juan Cao, Mingyan Lu, Jintao Li, and Chia-Wen Lin. How to write high-quality news on social network? predicting news quality by mining writing style. *CoRR*, abs/1902.00750, 2019.

Sequence Modelling For Analysing Student Interaction with Educational Systems

Christian Hansen, Casper Hansen, Niklas Hjuler, Stephen Alstrup, Christina Lioma
Department of Computer Science
University of Copenhagen, Denmark
{chrh,bnq,hjuler,s.alstrup,c.lioma}@di.ku.dk

ABSTRACT

The analysis of log data generated by online educational systems is an important task for improving the systems, and furthering our knowledge of how students learn. This paper uses previously unseen log data from Edulab, the largest provider of digital learning for mathematics in Denmark, to analyse the sessions of its users, where 1.08 million student sessions are extracted from a subset of their data. We propose to model students as a distribution of different underlying student behaviours, where the sequence of actions from each session belongs to an underlying student behaviour. We model student behaviour as Markov chains, such that a student is modelled as a distribution of Markov chains, which are estimated using a modified k-means clustering algorithm. The resulting Markov chains are readily interpretable, and in a qualitative analysis around 125,000 student sessions are identified as exhibiting unproductive student behaviour. Based on our results this student representation is promising, especially for educational systems offering many different learning usages, and offers an alternative to common approaches like modelling student behaviour as a single Markov chain often done in the literature.

Keywords

Markov Chains, Sequence Modelling, Clustering

1. INTRODUCTION AND RELATED WORK

How students interact with educational systems is today an important topic. Knowledge of how students interact with a given system can give insight in how students learn, and directions for the further development of the system based on actual use. The interaction can be studied both by explicit studies [7] directly observing student interaction *in situ*, or by the use of log data collected automatically by the use of the system as done in this paper.

Analysis of log data is often viewed as an unsupervised clustering problem at the student level [4, 8]. Our work takes another direction and focuses on the action sequence level. For clustering sequences, Markov models are popular as they provide a convenient way of modelling the transitions and dependencies of the sequences [9]. For action sequence mining, both hidden and explicit models have been used depending on the tested hypothesis, and on whether the states are explicit or implicit. Beal et al. use hidden Markov models for student prediction, assuming underlying hidden states of engagement, which can be clustered [2]. Köck and Paramythis use explicit states for analysing problem solving activity sequences, as the states in this case are explicit and therefore appear directly in the log [9].

The choice of clustering of the Markov models depends on the application area. Klingler et al. did student modelling by the use of explicit Markov chains, and the clustering was done by different similarity measures defined on the Markov chains themselves [8], e.g. euclidean distance between transitional probabilities, or Jensen-Shannon Divergence between the stationary probabilities of the chains. When individual sequences are clustered, an underlying assumption of the data coming from a mixture of Markov chains has been used [10], where the individual chains represent the cluster centres, and the task is finding both the chains and the mixing coefficients.

The work presented in this paper is using discrete Markov chain models for action sequence analysis, on log data[1] acquired from the company Edulab. Edulab is the largest provider of digital learning for mathematics in Denmark, having 75% of all schools as customers, and receiving more than 1 million student answers a day. Using a mixture of Markov chains, we assume that each chain will represent a prototype student behaviour. So the underlying assumption in this work is that each student can be modelled as behaving according to some underlying behaviour during each session, and a student can then be seen as a distribution over different behaviours. Edulab's product offers many different ways of learning mathematics, ranging from question-heavy workloads to video and text lessons, and other activities depending on whether the student is in class or at home. This allows to model a student as "distributed" over different behaviours, in contrast to a single student behaviour model of how the student usually interacts with the system.

We reason that mixture of Markov chains will allow for a qualitative study of what type of behaviour each chain rep-

[1]The data is proprietary and not publicly available

resents, and thus ultimately it can be used to show how a student uses the educational system.

Mixtures of Markov models can be solved by the EM algorithm, which however is notoriously slow to run for large amounts of data, and only local optimal solutions are found [6]. In this paper we need fast processing in order to analyse the large amounts of data produced by Edulab, so we simplify the assumptions on the underlying Markov chains, which allows for a modified version of k-means clustering.

Initial cluster centres, representing underlying student behaviour, can be chosen by domain experts and then refined through the clustering. However, since the true number of underlying clusters is unknown, it is difficult for an expert to predefine sensible cluster centres for a range of different numbers of clusters. In this work we first perform simulations to consider the effect of starting at the correct locations versus adding noise to the correct location until the starting points are completely random. Based on these results clustering is done on the Edulab dataset, and a qualitative analysis is performed on the resulting Markov chains. This shows how students are distributed among the Markov chains, and how unproductive system usage can be detected using the Markov chains.

In summary the primary research questions this paper addresses are: 1) to what extent can students be modelled as a distribution over underlying usage behaviours which is changing across sessions, and 2) how this modelling leads to insight in future improvements of the system for the producers of educational systems.

2. DATA

The data used in this work is produced by matematikfessor.dk, a Danish mathematics portal made by Edulab that spans the curriculum for students aged 6 to 16. The website offers both video and text lessons in combination with exercises covering the whole curriculum, such that it can be used as a primary tool for learning, and not only supplementary. Log data generated by the grade levels corresponding to students of age 12 to 14 for the 2016 school year is used (from August 2016 to February 2017). An action in this system can either be watching a lesson, which contains either a video or text description, or answering a question. Lessons and questions both have a topic id, specifying the general topic of the question or lesson. The data statistics are summarized in Table 1. The lessons and questions can be assigned as homework or done freely by the students (this study does not differentiate between whether it is homework or not). It should be noted that a lesson takes significantly longer time doing than answering a question hence the lower ratio of lessons, compared to other actions, in Table 1.

The logs do not contain information about when a session is started or finished, so we define a session as a sequence of actions, where the time between two actions is less than 15 minutes. A student has on average 12.5 sessions (standard deviation of 13.3), and the histogram of the number of actions in each action sequence can be seen in Figure 1, where sequence lengths larger than 200 have been removed from the plot for the purpose of visualization. When a student interacts with the system his actions are stored and seen as

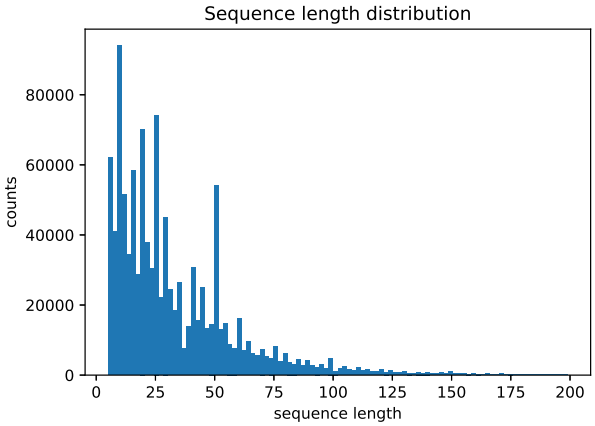


Figure 1: The distribution of action sequence lengths with lengths larger than 200 removed.

Number of sequences	1.08M
Number of actions	37.5M
Number of lessons	1.35M
Number of correctly answered questions	27.44M
Number of wrongly answered questions	8.71M

Table 1: Data statistics. The number of lessons and question answers sum to the number of actions.

an action sequence, an example of one is:

$$Qr_1^{t_1}, Qw_2^{t_2}, L_3^{t_1}, Qw_4^{t_3}, Qr_5^{t_1}, Qr_6^{t_1}, Qr_7^{t_1} \quad (1)$$

Qr is a correctly answered question, Qw is an incorrectly answered question, and L is a lesson. The subscript denotes the action number in a temporal ordering, and the superscript denotes the topic id, which is associated with each lesson and question.

3. METHOD

Our method for action sequence clustering will be explained in this section, and is based on modelling interactions with the system as Markov chains. Our Markov chain model with its transitions is shown in Figure 2. Our model consists of 8 states as will now be explained with their abbreviations in parentheses. These abbreviations are used for visualizing the resulting Markov chains from the clustering. The first two are start (S) and end (E). The rest consists of three general states: Doing a lesson (L), answering a question right (Qr), or answering a question wrong (Qw). Each lesson and question have an associated topic id, which might change from action to action creating the last three states: doing a lesson in another topic than the previous action (L_c), answering a question right in another topic (Qr_c), and answering a question wrong in another topic (Qw_c). If we consider the sequence described in Equation 1, then that

would correspond to visiting the following states

$$S \to Qr \to Qw\_c \to L\_c \to$$
$$Qw\_c \to Qr\_c \to Qr \to Qr \to E \quad (2)$$

The pipeline for clustering has the following procedure.

1. For every session we extract a sequence of actions $A_1, ..., A_n$, and each action sequence corresponds to a path in the used Markov chain model.

2. Since the Markov chains are unknown, priors $P_1, ..., P_k$ (which themselves are Markov chains) are generated at random such that each edge shown in Figure 2 has a transition probability taken uniformly at random from 0 and 1. Each random chain is normalized such that each state's outgoing transitional probabilities sum to one. These priors function is the pendant to the usual initial cluster centers, which most often are random data points. Generating a Markov chain from a randomly chosen point would however not work in our case, since many zero valued transition probabilities would occur.

3. Each action sequence is assigned to the prior which was most likely to generate it, i.e.

$$\arg\max_{1 \le j \le k} \left(\prod_{i=1}^{m} p^j_{b_{i-1}, b_i} \right) \quad (3)$$

   where $p^j_{b_{i-1}, b_i}$ is the transition probability from state b_{i-1} to b_i in prior P_j, m is the number of transitions between states, and k is the number of priors.

4. After each action sequence has been associated with a prior, then each prior is updated by generating the Markov chain most probable given its associated action sequences. This is done by counting the state transitions in each sequence in a new Markov chain model, and normalizing afterwards.

5. Points 3 and 4 are ideally reiterated until convergence, i.e. no action sequence changes its associated prior. However for computational reasons we stop iterating after less than 5% of the sequences have changed their assigned prior.

The clustering technique is very similar to ordinary k-means clustering, with the major difference that the clustering is not dependent on a similarity measure directly on the sequence, but dependent on the Markov chains generated by the clustering. Comparing to ordinary k-means clustering, the produced chains in each iteration are analogous to the ordinary cluster center found by some mean. The mixture model could also be estimated by the EM algorithm [1], which has the benefit that sequences that do not belong to a single clear cluster, i.e. that have multiple highly probable chains, will weight in on all of them. This has the downside that clusters take longer to be separated, and the convergence is therefore slower. Under the assumption of the chains being distinct, each sequence will mostly weight on a single chain, and here the k-means clustering method and EM algorithm will perform very similarly. For the data from Edulab we assume most of the chains to be distinct,

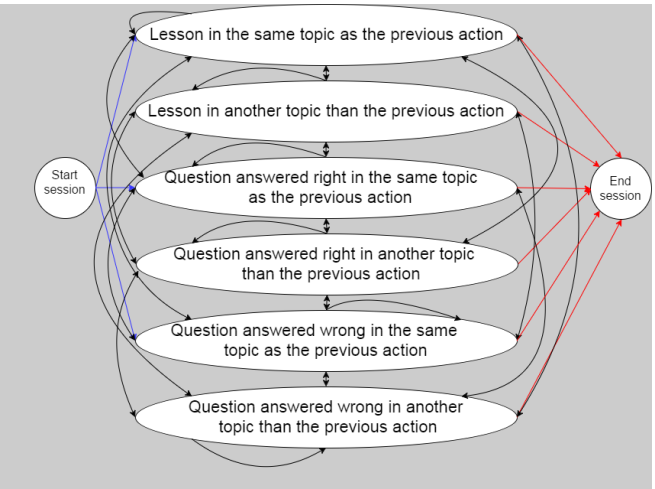


Figure 2: Markov chain representing the possible states and transitions. Note the transitions each way do not have to be equal.

but not necessarily all. In addition a very large number of sequences will have to be clustered in the future when the full dataset is used, and not restricted as done for this paper. We are therefore mostly interested in how well the k-means clustering approach performs as it is more computationally feasible when the data size is increased.

The above procedure leaves two challenges: 1) How do we know the resulting Markov chains are close to the real ones? and 2) How to estimate the number of priors? We address these points next.

The first point is dealt with using synthetic data, where k random Markov chains are made, and each action sequence is generated from one of those chosen uniformly at random. In order to ensure a suitable length of the generated action sequences, the ingoing probabilities to the end state are fixed to allow for an average sequence length of 20. After generating the synthetic data, the most probable Markov chain for each sequence is assigned as its label, and the goal in the clustering is to be able to capture these clusters. Note, that since each sequence is randomly generated using the chosen Markov chain, then its most probable Markov chain might not be the one generating it. To determine the ability to capture the original clusters we consider the average purity of the resulting clusters:

$$Average_{purity} = \frac{1}{n} \sum_{i=1}^{n} \frac{\max_{1 \le j \le k}(|C_j \cap S_i|)}{|S_i|} \quad (4)$$

Where S_i is an estimated cluster, C_j is the true cluster, n is the number of clusters, and k is the number of true clusters. An average purity of 1 represents that the method fully captures the original clusters. The underlying Markov chains are unknown on real data, so increasingly noisy versions of the underlying Markov chains are experimented with as priors, to show how the method is expected to perform under real circumstances.

In the case of real data, the true underlying Markov chains are unknown, so in this case the sum of the log likelihoods is calculated for the sequences to their most probable prior:

$$\text{sum of log likelihood} = \sum_{i=1}^{n} \log\left(\mathcal{L}(s_i|P_i^*)\right) \tag{5}$$

where s_i is an action sequence, P_i^* is the prior most likely to generate action sequence s_i, and $\mathcal{L}(s_i|P_i^*)$ is the likelihood that P_i^* generates s_i.

The second point mentioned earlier, about estimating the number of priors, can be solved using either the average purity in the synthetic case, or from the sum of log likelihoods in the real case. The sum of log likelihoods as a function of k will be monotonically increasing, but the slope will decrease as k exceeds its true underlying value. Since the method starts with randomly chosen priors, it is repeated a number of times, and the solution with the largest log likelihood is chosen for each value of k.

4. SIMULATED EXPERIMENT WITH NOISY PRIORS

There are two approaches for estimating the Markov chains for the Edulab data set. 1) The prior Markov chains can be chosen by domain experts - by specifying common sequences we would expect to find in the data, and then refine them during the clustering. 2) The second approach is as described in the method section, starting with random chains, and running k-means multiple times, and taking the clustering which gives the highest sum of log likelihoods. To measure how the method behaves as the initial priors are increasingly noisy versions of the underlying Markov chains, k-means is run with the priors chosen as:

$$P_i = (1-\alpha)P_i^* + \alpha P_{rand} \tag{6}$$

Where all Ps are Markov chains represented by matrices of transitional probabilities, and α is the noise parameter. P_i is the i^{th} prior, P_i^* is the i^{th} underlying Markov chain used when generating the synthetic data, and P_{rand} is a random Markov chain. The higher α, the more noisy the initial prior is.

In Figure 3, we see how the average purity behaves as a function of noise parameter α. The experiment is run for $k = 6$, and 6 random chains are generated. The transition probabilities to the end state are fixed at 0.05 for all states for all chains to allow for sequences of average length 20. 50000 sequences are sampled uniformly from the 6 chains. The modified k-means is then run with the priors varying depending on α, and the experiments are run 10 times and purity is the average over the 10 runs. First we note that even with using the modified k-means algorithm and not the EM algorithm the resulting average purities are quite high. It is seen that even with $\alpha = 1$ representing completely random priors, the reduction in purity is not too large compared to starting with the same priors as the data is generated from. Even starting with the same priors which generated the data does not guarantee perfect purity, which is expected as there are some sequences that are almost as likely under multiple chains, so small differences in the data determined Markov chains will move them from one chain to another. Based on the above result we will not define the priors by an expert, and instead let them be random. This has the benefit of being more manageable than handcrafting specific priors for each choice of k, which would be very difficult to do in a meaningful way when k is large.

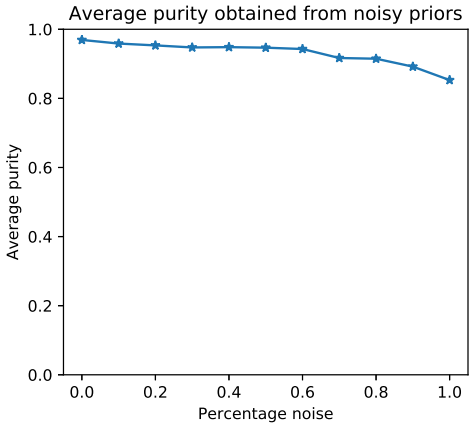


Figure 3: Average purity as a function of increasingly more noisy priors. A completely random prior (1.0 on the x axis) is able to perform well.

5. REAL DATA EXPERIMENT

5.1 Choosing the number of clusters

The problem of determining the number of clusters is common for all unsupervised learning tasks. In this paper we consider the sum of the log likelihoods for the action sequences. A common approach is the use of the "elbow" heuristic, where the choice of k is chosen based on the slope of the sum of log likelihoods function over k.

In order to argue that there is structure in the data, and that the method is able to capture this structure, a randomized experiment is made. The randomized experiment consists of randomly permuting each sequence (but keeping the start and end states), and seeing how the sum of log likelihoods is affected by it. If there is no structure originally in the sequences, then one can not expect it to perform better than the permuted data.

In Figure 4 we see that the sums of log likelihoods are considerably lower in the permuted data set, with only slightly higher sum of log likelihoods when $k = 20$ compared to $k = 2$ for the real data set. The action sequences therefore have structure which the Markov chain captures, and it is therefore not just random chains that the k-means clustering produces. Since the chains capture some inherent structure in the data, it is meaningful to analyse the individual chains with regards to what user behaviour they capture.

There is not an obvious breaking point in the sum of log likelihoods, but the increase before $k = 6$ is large, while the increase for $k > 10$ is notably smaller, so a value of k between 6-10 is sensible. We will in the qualitative assessment of the chains use $k = 6$.

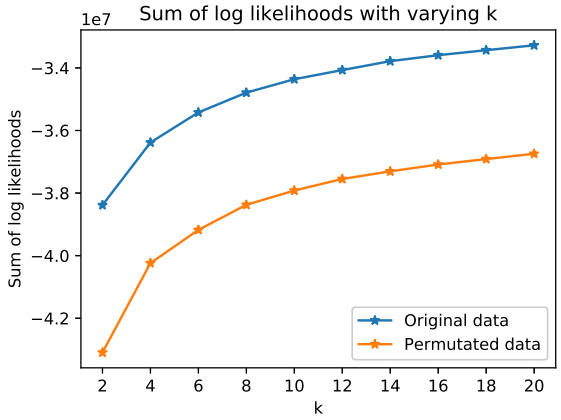


Figure 4: Sum of likelihoods for the best performing clusters for each k. Each experiment is run 5 times for each k. The permutations of each sequence is done for each value of k in each of the 5 times.

5.2 Qualitative assessment of Markov chains

This section will make qualitative assessments of what the different resulting Markov chains represent with regards to what type of user behaviour they capture. Even with six chains there is some similarity between some chains, so in this section we will focus on the three most distinct chains shown in Figure 5. The thickness of the arrows is proportional to the transitional probability for each state, except the ending state. The transitional probabilities are sorted and only drawn until 70% of the probability mass is covered. For the ending state, 70% of the incoming transitional probabilities are drawn.

In general not all chains can be described as either being a positive or negative usage of the system. Chain 2 captures usage where most of the questions being answered are either right or wrong, and there is very little mixing between taking lessons and answering a question. Usage like this could indicate an unproductive session for students, since they are mostly getting all questions right or all questions wrong, and research shows that students feel more intrinsic pleasure when the difficulty level is slightly challenging [5] leading to more engaged sessions [3]. Similarly, watching lessons without engaging with the material via questions leads to students not training the learned material, which is important for the learning process.

Chain 6 can be described as a positive usage of the system, as the most probable transitions lead to a question being correctly answered, except for the two transitions in the lessons. Generally students are focused on one topic at a time.

Chain 4 has high transitional probability when switching between topics, so this could indicate a session with a primary focus on repetition as the topic is varying, and students most often answer questions from another topic than the watched lessons.

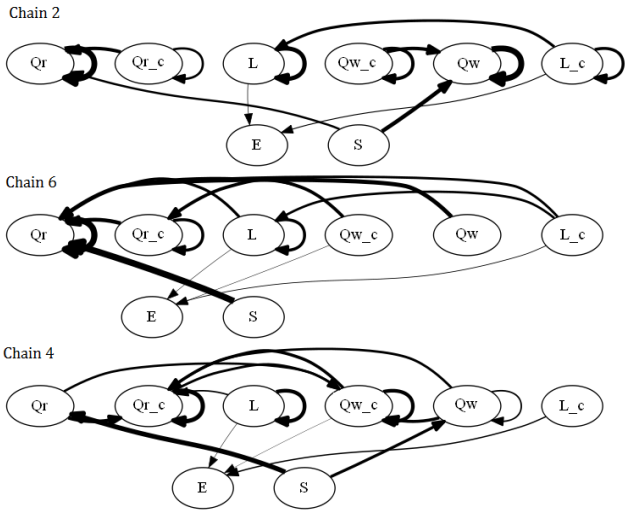


Figure 5: Chains 2, 6, and 4 of the six chains. The thickness of the arrows is proportional to the transitional probability for each state, except the ending state. The transitional probabilities are sorted and only drawn until 70% of the probability mass is covered. For the ending state 70% of the incoming transitional probabilities are drawn. State abbreviations are explained in section 3.

	Num. sequences	Avg. sequence length
chain 1	295,792	34.81
chain 2	126,683	36.88
chain 3	198,736	26.79
chain 4	131,460	28.79
chain 5	194,174	36.12
chain 6	144,121	44.85

Table 2: The number of sequences and average length of sequences for each Markov chain

The distribution of the sessions over the chains can be seen in Table 2.

The length of the sequences is varying, but no single chain in general captures either the very short or very long sequences. Instead a combination of shorter and longer sequences is captured by each chain. The most common chain can be seen in Fig 6. This chain is similar to chain 4 (Fig 5), but with more topic changes and more wrongly answered questions when changing topics, which can be seen in the self loop for Qw_c. Chain 4 is also shorter on average. As seen in Table 2, generally all six chains contain a large amount of sequences on average. This indicates that the system usage does indeed vary, and is not limited to all sequences of the same length defining the same use of the system. If one considers each user's distribution of Markov chains, then on average each user has 3.5 different types of sessions out of 6 with a standard deviation of 1.5. This supports the assumption that a single Markov chain is not optimal for user profiling for educational systems similar to the one generating our data, where there is a lot of user freedom in what

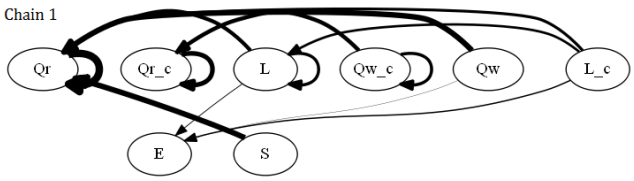


Figure 6: Chain 1, the most common chain. State abbreviations are explained in section 3.

activities they engage in.

6. DISCUSSION AND CONCLUSION

In this work first order Markov chains have been used, but it is generally known that the action sequences do not fulfil the Markov property of transition to a state only being dependent on the previous state. No order of Markov chain will completely capture the underlying transition between states, as the usage is dependent on many external factors which are unknown, but higher order chains would be able to capture more complex dynamics in the usage. Even though the Markov property is violated, Markov chains are still very widely used in educational data mining [4, 8], and provide a good tool for comparisons of action sequences across different lengths, focusing on the flow of actions taken. In future work an interesting extension would be considering time dependent Markov models, such that the transitional probabilities are dependent on how long the states have been unchanging. This would allow for more interpretative models, e.g. we could see when the probability of a session ending gets high.

When inspecting the Markov chains produced by the clustering, chain number 2 indicated suboptimal or unproductive usage of the system, where the students either experience questions that are too easy or too hard, or never train what they learn in the lessons. The chain has 126,683 sessions in its cluster, and it is therefore a significant amount of sessions where the learning outcome most likely could be improved. Based on this it could be recommended to have a few obligatory questions after a lesson to strongly encourage the student to use what they have just learned, and detect negative spirals where the students are always wrong by recommending lessons to help the student move forward.

Modelling the student as a distribution over Markov chains, which can be considered usage patterns, results in a vector representation of the individual students. This representation allows to apply standard techniques directly on the student model, compared to working on more complex student models. An example is the issue of drift in student behaviour over time, corresponding to some learning, or wider cognititive development of the student. This problem has also been considered in a similar context in [8], where distances between single Markov chains on a student level were estimated. However, in our setting standard methods could readily be used to detect this type of drift and potentially alert the teacher.

The work presented shows a qualitative study of the proposed student representation, and experiments using synthetic data show that our methodology is able to capture the underlying generative Markov chains very well, when the number of chains has been estimated. A source for future work will be using the student vectors in a predictive task, such that quantitative measures can be acquired. An interesting path would be using knowledge tracing methods over the different session types, to see if there are any unexpected differences between the knowledge acquired by the student depending on the type of session - i.e. the kind of Markov chain the session originates from.

7. ACKNOWLEDGMENTS

The work is supported by the Innovation Fund Denmark through the DABAI project.

8. REFERENCES

[1] D. Barber. *Bayesian Reasoning and Machine Learning*. Cambridge University Press, New York, NY, USA, 2012.

[2] C. Beal, S. Mitra, and P. R. Cohen. Modeling learning patterns of students with a tutoring system using hidden markov models. In *Proceedings of the 2007 Conference on Artificial Intelligence in Education: Building Technology Rich Learning Contexts That Work*, pages 238–245, Amsterdam, The Netherlands, The Netherlands, 2007. IOS Press.

[3] M. Csikszentmihalyi and I. Csikszentmihalyi. *Optimal Experience: Psychological Studies of Flow in Consciousness*. Cambridge University Press, 1992.

[4] L. Faucon, L. Kidzinski, and P. Dillenbourg. Semi-markov model for simulating mooc students. In T. Barnes, M. Chi, and M. Feng, editors, *EDM*, pages 358–363. International Educational Data Mining Society (IEDMS), 2016.

[5] J. Gottlieb, P.-Y. Oudeyer, M. Lopes, and A. Baranes. Information-seeking, curiosity, and attention: computational and neural mechanisms. *Trends in Cognitive Sciences*, 17(11):585–93, Nov. 2013.

[6] R. Gupta, R. Kumar, and S. Vassilvitskii. On mixtures of markov chains. In *Advances in Neural Information Processing Systems*, pages 3441–3449, 2016.

[7] S. Hutt, C. Mills, S. White, P. J. Donnelly, and S. K. D'Mello. The eyes have it: Gaze-based detection of mind wandering during learning with an intelligent tutoring system. In *Proceedings of the 9th International Conference on Educational Data Mining, EDM 2016, Raleigh, North Carolina, USA, June 29 - July 2, 2016*, pages 86–93, 2016.

[8] S. Klingler, T. Käser, B. Solenthaler, and M. Gross. Temporally Coherent Clustering of Student Data. In *Proceedings of EDM*, pages 102–109, 2016.

[9] M. Köck and A. Paramythis. Activity sequence modelling and dynamic clustering for personalized e-learning. *User Modeling and User-Adapted Interaction*, 21(1):51–97, 2011.

[10] Y. Yang, Q. Yang, W. Lu, S. J. Pan, R. Pan, C. Lu, L. Li, and Z. Qin. Preprocessing time series data for classification with application to crm. In S. Zhang and R. Jarvis, editors, *Australian Conference on Artificial Intelligence*, volume 3809 of *Lecture Notes in Computer Science*, pages 133–142. Springer, 2005.

Tracking Behavioral Patterns among Students in an Online Educational System

Stephan Lorenzen
University of Copenhagen
lorenzen@di.ku.dk

Niklas Hjuler
University of Copenhagen
hjuler@di.ku.dk

Stephen Alstrup
University of Copenhagen
s.alstrup@di.ku.dk

ABSTRACT

Analysis of log data generated by online educational systems is an essential task to better the educational systems and increase our understanding of how students learn. In this study we investigate previously unseen data from Clio Online, the largest provider of digital learning content for primary schools in Denmark. We consider data for 14,810 students with 3 million sessions in the period 2015-2017. We analyze student activity in periods of one week. By using non-negative matrix factorization techniques, we obtain soft clusterings, revealing dependencies among time of day, subject, activity type, activity complexity (measured by Bloom's taxonomy), and performance. Furthermore, our method allows for tracking behavioral changes of individual students over time, as well as general behavioral changes in the educational system. Based on the results, we give suggestions for behavioral changes, in order to optimize the learning experience and improve performance.

Keywords

Student clustering, Non-negative matrix factorization, Educational Systems

1. INTRODUCTION + RELATED WORK

How students behave in educational systems is an important topic in educational data mining. Knowledge of this behavior in an educational system can help us understand how students learn, and help guide the development for optimal learning based on actual use. This behaviour can be understood both through an explicit study [5], or as in this paper through the automatically generated log data of the system.

The analysis of log data is usually done as an unsupervised clustering of students [2, 3, 4, 7]. A popular approach is to extract action sequences and transform them into an aggregated representation using Markov models [4, 7]. The Markov chains can then be clustered by different methods. Klingler et al. did student modeling with the use of explicit Markov chains and the clustering with different distance measures defined on the Markov chains [7]. Hansen et al. assumed the actions sequences to be generated by a mixture of Markov chains and used an heuristic algorithm to find the generating Markov chains [4]. Gelman et al. used non-negative matrix factorization to find clusters for different measures of activity aggregated in weekly periods during a MOOC course. These clusters are then matched from week to week by cosine similarity.

Our work is similar to Gelman et al. [3] in that we also use *Non-negative Matrix Factorization* (NMF) to make a soft clustering at the student level in a given time period, however our clustering is only made once, and we are looking at primary school data over a vastly longer period of time, (2 years compared to 14 weeks).

Our soft clustering by non-negative matrix factorization is based on log data from Clio Online.[1] Clio Online is the largest provider of digital learning for all subjects in the Danish primary school (except mathematics), having 90% of all primary schools in Denmark as customers.

Using NMF, we assume that the set of features chosen can be represented by a set of fewer underlying behaviors. These underlying behaviours would each be represented by a cluster in the non-negative matrix factorization. Each student will then get a number for each cluster in each time period representing how much of that underlying behavior he has shown in the given time period. Non-negativity gives the behaviors an additive structure, which is more natural than showing a negative amount of a given behavior. We reason that the soft clustering will show both the behaviors of individual students, as well as how the behaviors change over time, both individually and on a system-wide level.

In this paper, we will consider two main questions: a) how does student activity in the system affect performance, and b) how does student activity distribute between different levels of Bloom's taxonomy in different subjects. Both questions are important in regards to optimizing learning; the first in relation to performance, the latter in relation to utilization of all taxonomy levels.

[1]This data is proprietary and not publicly available.

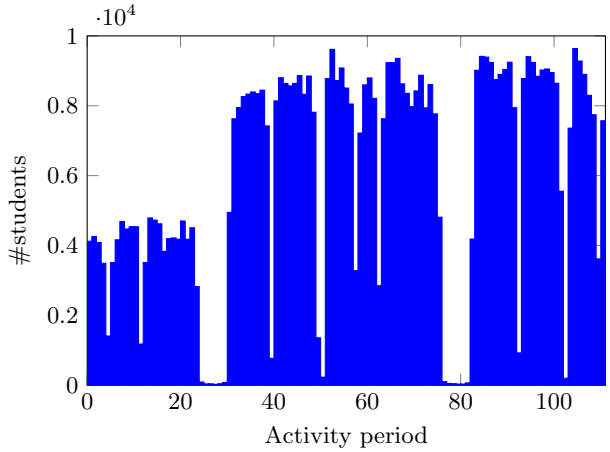


Figure 1: Number of students active in each period. Note that period 0 starts on 2015-01-08, while period 111 ends on 2017-03-01. The drops in activity occur due to vacation in Danish primary school, with the two large drops around periods 25 and 79 being due to the summer vacation.

2. EXPERIMENTAL SETUP

This section describes our experimental setup and methods. We start by describing our data and how it is preprocessed, and then move on to describing our clustering method.

2.1 Data Preprocessing

As mentioned, we consider log data generated in the Danish online educational system Clio Online. The system is used in Danish primary schools and contains learning objects across all Danish subjects (except mathematics), for instance texts, videos, sound clips and exercises. Furthermore, the system includes a large number of quizzes, used for evaluating students. Students may use the system for self study, but they may also be assigned homework by their teacher. Our data covers 14,810 students.

The raw data consists of logs detailing page accesses for individual students in the system. For quizzes, the final score (between 0 and 1) and total time spent for the quiz is also available. In our preprocessing, we combine these log entries to *sessions*. Two consecutive entries are considered in the same session, if they have the same subject, and their timestamps differ by less than some threshold. For our study, we choose this threshold to be 600 seconds, based on recommendations from Clio Online, who have a deeper knowledge of the content and flow of the system (e.g. expected time per page). Furthermore, quizzes are considered separate sessions. A total of 3 million sessions is obtained in this way.

With the sessions defined, we consider student activity in *activity periods*, with a length of one week. The data spans a total of 112 activity periods, starting January 2015 and ending in March 2017. For each activity period, we add an entry for a student, if the student is active (accesses the system) within that period. The entry for the given student contains all sessions for that student, which starts within the activity period. We end up with approximately 677,000 student entries across the 112 periods. Figure 1 shows the active number of students in each period. Note the drop in active students around periods 25 and 79; these drops in activity occur due to summer vacation.

The final step of data preprocessing is the feature extraction. For each activity period, a set of activity/performance related features are extracted. The features are chosen so as to answer the questions posed in the previous section. A complete overview of all features considered in our experiments is given in Table 1, including the maximum, mean and variance across all active students in all periods. Not all features are used for each experiment, see section 3.

All features are aggregates over the activity period. Below follows a detailed description:

- f_1 describes the activity during the period of day, where Danish students are normally in school, while f_2 describes the activity during non-school hours.
- f_3, f_4 and f_5 describe time spent doing exercises, reading texts and taking quizzes respectively.
- f_6, f_7 and f_8 describe time spent working with different topics: languages (Danish, English, German), societal (social studies, history, etc.) and science (physics, biology, etc.), respectively.
- f_9 is the average session length during the activity period.
- f_{10} is the average quiz score; this feature may be missing, if a student takes no quizzes during an activity period, but our analysis methods can handle this, see section 2.2.
- f_{11}, f_{12}, f_{13} and f_{14} describe the time spent doing exercises of different complexity, measured by their level in Bloom's taxonomy. We regroup the levels of Bloom's taxonomy into 4 levels:

  f_{11} **Remember/Understand**: Exercises involving reading and describing, e.g. "Read a map".
  f_{12} **Apply**: Exercises involving application of previously learned concepts, e.g. "Practice adjectives".
  f_{13} **Analyze/Evaluate**: Exercises involving discussion, analysis and experimenting, e.g. "Work with the poem", "Analyze the game".
  f_{14} **Create**: Exercises involving creation of a product, e.g. "Create a cartoon", "Write a story".

Having extracted m features for each student in each period, we construct the matrix $\mathbf{X} \in \mathbb{R}^{n \times m}$, where each of the n rows consists of the feature vector for an active student in a given activity period. Thus each student occurs several times in $\mathbf{X}$; once for each period, where they are active.

2.2 Soft Clustering using Non-negative Matrix Factorization

We will utilize non-negative matrix factorization for our soft clustering. The use of NMF as a soft clustering technique has become popular in recent times [10], with applications within several fields, such as clustering of images and documents [8, 13]. NMF has also seen success in the educational data mining community, for clustering tasks, as well as other tasks such as performance prediction [3, 12].

i	f_i	Max	Mean	Variance
1	Hours between 8AM and 4PM	31.85	0.940	0.862
2	Hours before 8AM and after 4PM	71.84	0.174	0.283
3	Hours doing exercises	3.61	0.048	0.019
4	Hours reading texts	7.73	0.344	0.148
5	Hours taking quizzes	23.76	0.231	0.297
6	Hours working with language subjects	58.28	0.531	0.693
7	Hours working with societal subjects	45.96	0.294	0.285
8	Hours working with science subjects	103.69	0.277	0.326
9	Average session length in hours	7.91	0.268	0.027
10	Average quiz score (in $[0, 1]$)	1.00	0.733	0.034
11	Hours working with Bloom level 1	2.83	0.016	0.006
12	Hours working with Bloom level 2	1.64	0.008	0.002
13	Hours working with Bloom level 3	1.51	0.014	0.003
14	Hours working with Bloom level 4	2.04	0.009	0.003

Table 1: Overview of features.

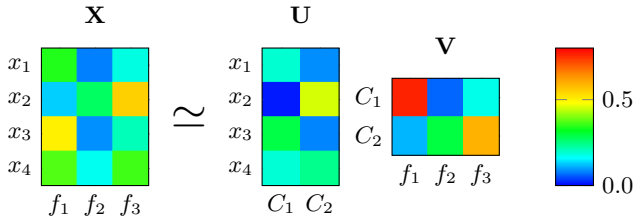


Figure 2: The soft clustering given by NMF.

NMF is a dimensionality reduction method, in which we are given a non-negative matrix $\mathbf{X} \in \mathbb{R}_+^{n \times m}$ and $k \in \mathbb{N}$, and wish to determine $\mathbf{U} \in \mathbb{R}_+^{n \times k}, \mathbf{V} \in \mathbb{R}_+^{k \times m}$, such that $\mathbf{X} \simeq \mathbf{UV}$. More specifically, we search for $\mathbf{U}$ and $\mathbf{V}$, such that the error $||\mathbf{X} - \mathbf{UV}||_F$ is minimized, where $|| \cdot ||_F$ is the Frobenious norm. For our analysis, we need to be able to handle missing values in $\mathbf{X}$. In this case the NMF problem is reformulated as the *weighted non-negative matrix factorization*, in which we are also given a binary weight matrix $\mathbf{W} \in \{0, 1\}^{n \times m}$, where a 0 indicates missing data. Now, we wish to find $\mathbf{U}, \mathbf{V}$ such that $||\mathbf{W} \odot (\mathbf{X} - \mathbf{UV})||_F$ is minimized[2].

$\mathbf{U}$ and $\mathbf{V}$ admits a soft k-clustering as shown in Figure 2; $\mathbf{V}$ describes the importance of each feature for each cluster (for instance, f_1 has high importance in C_1), while $\mathbf{U}$ describes the membership of each data point to the different clusters (for instance, x_3 is mostly in C_1, while x_4 is in both clusters).

Note, that for NMF, we have $\mathbf{X} \simeq \mathbf{UV} = \mathbf{UIV} = \mathbf{UA}^{-1}\mathbf{AV}$, where $\mathbf{I}$ is the $k \times k$ identity matrix and $\mathbf{A}$ is a $k \times k$ invertible matrix. This means that we may rescale $\mathbf{U}$ and $\mathbf{V}$ by this matrix, $\mathbf{A}$, and its inverse. In our analysis, we use this to rescale $\mathbf{V}$, such that all rows of $\mathbf{V}$ (the clusters) sum to one, thus making the clusters comparable, and membership of the different clusters easier interpretable.

There exist several algorithms for obtaining the non-negative matrix factorization of $\mathbf{X}$, for instance basic gradient descent, multiplicative update rules and alternating least squares; [1] gives a good overview in the non-weighted setting. Several of these algorithms have been adapted for the WNMF case, while approaches based on *expectation maximization* have also been proposed, see [6]. For our analysis, we will use the weighted version of the multiplicative update method, proposed by Lee and Seung [9].

The NMF algorithm given in [9], adopted to WNMF [6], is as follows:

1. Initialize $\mathbf{U}$ and $\mathbf{V}$.
2. Repeatedly update $\mathbf{U}$ and $\mathbf{V}$ by the following rules:

$$\mathbf{U} \leftarrow \mathbf{U} \odot \frac{(\mathbf{W} \odot \mathbf{X})\,\mathbf{V}^T}{(\mathbf{W} \odot (\mathbf{UV}))\,\mathbf{V}^T}$$

$$\mathbf{V} \leftarrow \mathbf{V} \odot \frac{\mathbf{U}^T\,(\mathbf{W} \odot \mathbf{X})}{\mathbf{U}^T\,(\mathbf{W} \odot (\mathbf{UV}))}$$

where division is done element-wise.

The literature explores several ways of initializing $\mathbf{U}$ and $\mathbf{V}$; in our case, we will simply use random initialization. The alternating optimization steps are applied until the decrease in error reaches below a set threshold. Finally, Lin has noted that the procedure described above may not converge to a stationary point, hence we modify the update rules as proposed by them [11]. Furthermore, since we in our case know all missing values of $\mathbf{X}$ to be bounded by a constant c, we modify the above procedure such that 0-weight values of $\mathbf{UV}$ that deviate above c are penalized, i.e. whenever a value $(\mathbf{UV})_{ij}$ with $\mathbf{W}_{ij} = 0$ gets larger than c, we set $\mathbf{X}_{ij} = c$ and $\mathbf{W}_{ij} = 1$, before the next update step. If $(\mathbf{UV})_{ij}$ decreases below c again, the weight is reset to 0.

It remains to be seen, how we select the number of clusters, k. For each experiment, we construct clusterings with $k = 1, 2, ...,$ and stop when the decrease in error going from k clusters to $k + 1$ clusters is below some threshold, which depends on the initial error. As a consequence clusters will be uncorrelated on a student level, since otherwise we would pick a lower k.

[2] $\odot$ denotes the Hadamard product (element-wise multiplication).

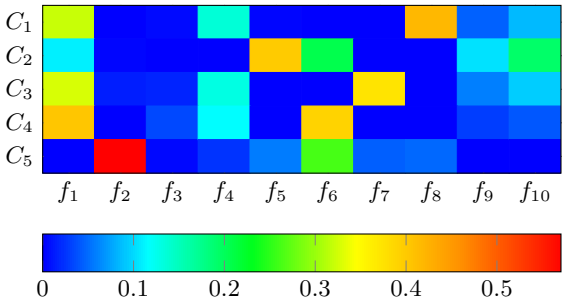


Figure 3: The cluster matrix for the first experiment.

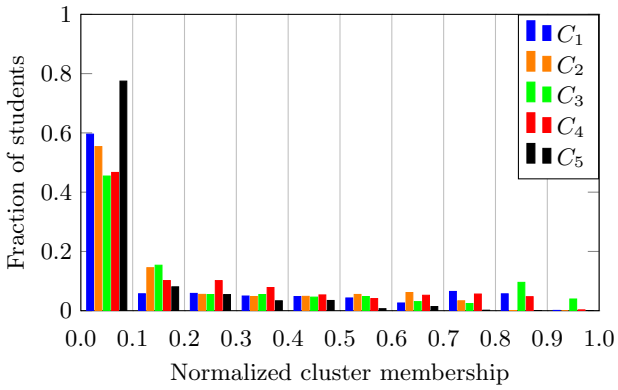


Figure 4: The distribution of cluster membership for the first experiment.

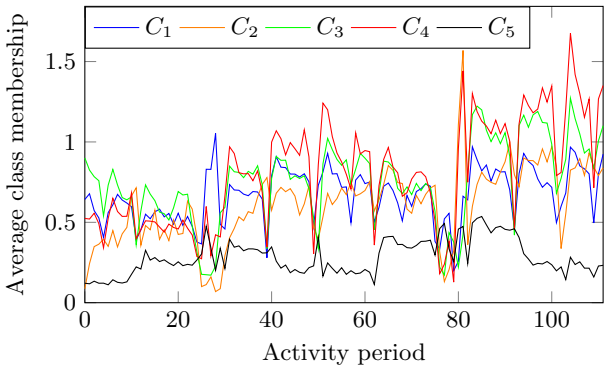


Figure 5: The average cluster membership in each activity period for the first experiment.

3. EXPERIMENTS AND RESULTS

In this section, we present two different experiments using the setup described above. In the first experiment, we investigate the relation between activity, activity type, subject, time of day, average session length and performance. In the second experiment, we investigate the relation between complexities of exercises and subjects.

3.1 Performance and Optimal Behavior

In the first experiment, we investigate the relation between activity, activity type, subject, time of day, average session length and performance, i.e. we consider features $f_1, ..., f_{10}$. The features are extracted and $k = 5$ is selected, as described in section 2. We run the WNMF algorithm, and obtain the cluster matrix V as shown in Figure 3. From the figure, we can make several observations about the clusters:

- C_1 In this cluster, we find students mostly working with the science subjects (f_8). These students seem to work mostly during school hours (f_1). The students also seem to spent a lot of time reading (f_4).
- C_2 Students in this cluster spend a lot of time taking quizzes (f_5). They will spend some time during school hours (f_1) and some time working with language subjects (f_6). Furthermore, students in this cluster seem to both have fairly long average session length and high performance (f_9 and f_{10}).
- C_3 In cluster C_3, we see students working with societal subjects (f_7). They work during school hours (f_1) and spend time reading texts in the system (f_4).
- C_4 This cluster shows a relationship between being active in school (f_1) and spending time in the language subjects (f_6). Students in this cluster also spend time reading texts (f_4) and doing some exercises (f_3).
- C_5 The most important feature for C_5 is f_2, i.e. the students in this cluster spend most time using the system during non-school hours. These students spent time in all subjects, but mostly languages (f_6), and they spent time taking quizzes (f_5).

From the clusters, we can see that the impact on performance from different behaviors depends on the subject. From cluster C_2, we see that students working mostly with language subjects gain most performance from spending time taking quizzes and working during school hours, whereas students working mostly with societal (cluster C_3) and science (cluster C_1) subjects gain most from reading texts, while working mostly during school hours. Note that cluster C_4 indicates that students working with languages may also improve performance by reading texts, but to a lesser degree than students working in other subjects. Finally, C_5 indicates that working mostly from home and primarily taking quizzes, does not improve performance. While C_5 indicates this for all subjects, the high importance of f_4 indicates that this most often occur for students working with languages, confirming the observations from C_2. Finally, it is also worth noticing, that there is a strong relation between performance and average session length (clusters C_1, C_2 and C_3), indicating that students, who perform well, also have longer sessions on average.

From the above discussion, it appears that the behavior in clusters C_4 and C_5 are sub-optimal, when considering performance, while students gain more from being in C_1, C_2 or C_3, i.e. by working during school hours, having longer sessions and taking quizzes (in the case of languages) or reading texts (in the case of societal or science subjects).

Figure 4 describes the distribution of cluster membership across all students and all activity periods , i.e. the columns of the first interval $[0, 0.1)$ gives for each cluster the fraction of students with 0%-10% membership. We see, that we do indeed get a soft clustering, with students often belonging to more than one cluster. Only C_3 seems to be the sin-

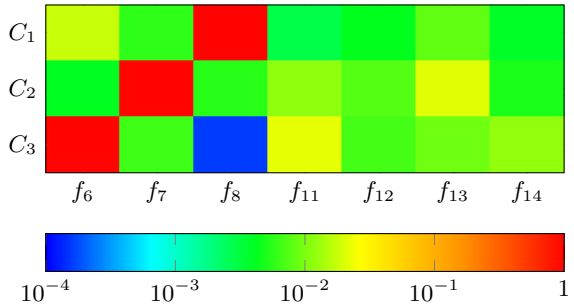


Figure 6: The cluster matrix for the second experiment. Note, that a logarithmic scale is used for this plot.

gle dominant cluster of some students. From the figure, we also see that students are typically never exclusively in C_5, which is positive, as the behavior observed in that cluster was not very productive in terms of performance. Other than that, we generally observe that students seem to distribute fairly well between the top four clusters, indicating most time spent during school hours and a varied use of both quizzes and texts across all subjects.

Next, we analyze how the membership of different clusters change over time. Figure 5 plots the average membership for each period, i.e. the average of rows from $\mathbf{U}$ belonging to the given period. The first observation we make from Figure 5, is that clusters C_1, C_2, C_3 and C_4 appear correlated at the system-wide level. This is due to these clusters being dependent on the general activity in the online system; most of the sudden drops occur at the same time as Danish school vacations, most notably the two larger drops around activity periods 25 and 79 (see Figure 1). C_5 seems to be relatively unaffected by the general activity, but this makes sense, as C_5 contains mostly students, who work outside school hours, and thus a lower membership is expected in that cluster in general, which is also the pattern we see in periods with no vacation.

Looking at the general distribution between the different clusters, C_3 and C_4 seem to be the most dominant, indicating that most students are working with language and societal subjects and reading texts. Cluster C_1 (science subjects) is fairly constant in the non-vacation periods, and C_2 seems to increase starting period 80, indicating that more students spend time taking quizzes. Finally, as mentioned, C_5 is the least active cluster across most periods. One general trend for the top four clusters seem to be an increase in activity during the 112 periods, indicating that students are spending more time in the system on average.

3.2 Subject and Exercise Complexity

In the second experiment we look at the relation between subjects and exercises grouped by Bloom's taxonomy level, i.e. we consider features $f_6, f_7, f_8, f_{11}, f_{12}, f_{13}, f_{14}$

We expect three clusters, one for each of the subject classes, which will tell us how much each Bloom level is used within each subject class. Figure 6 shows the cluster matrix found. From Figure 6, we make the following observations:

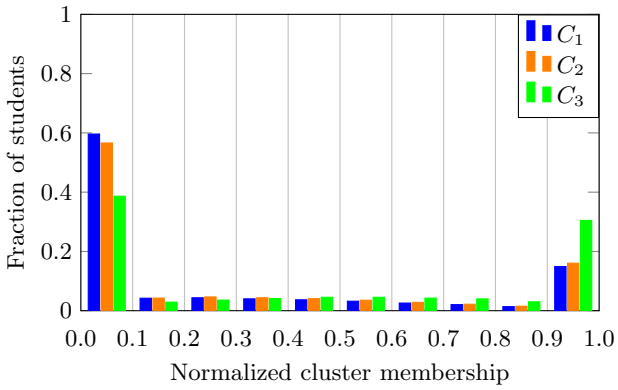


Figure 7: The distribution of cluster membership for the second experiment.

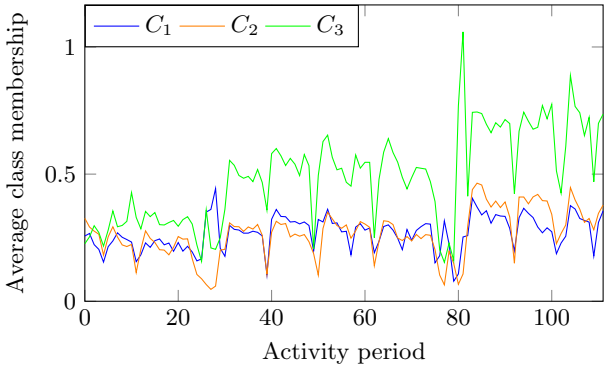


Figure 8: The average cluster membership in each activity period for the second experiment.

- C_1 In the science subjects, only very little of the 3 higher levels are used, and almost none of reading and understanding.
- C_2 For societal subjects, students have only little activity in the first 2 levels, a lot in analyzing and evaluating, and very little activity in creation.
- C_3 In languages, students have a tendency to read and understand a lot, and then distribute almost evenly on the 3 higher levels.

This implies that if we want to attract students to use an online educational system for languages, focus should be on exercises with Bloom's taxonomy level read and understand. For societal subjects the focus should be on exercises with analyzing and evaluating. For science we see no preference.

From Figure 7, we see that the clustering has many high values which is most likely explained by having a teacher who uses the system exclusively in only one of the subjects, which we can see happens most often for languages.

As we can see in Figure 8 all three clusters share similar curvature, which is partly explained by holidays. Especially the science and societal clusters behave seem highly correlated on a general level. We also see that in all three subjects, the average time spent during a week has gone from 15 minutes,

to 45 minutes for languages and 25 minutes for both societal subjects and sciences. A clear indication that teachers and students in Denmark are using online educational systems more, especially for languages.

4. CONCLUSIONS AND FUTURE WORK

Several points can be taken from our analysis. We have identified three optimal and two sub-optimal behaviors in relation to subject and performance. One notably conclusion is that students using the Clio Online system during non-school hours (at home) do not seem to gain any significant boost to performance. We also saw how taking quizzes seems to increase the performance of students in languages, more so than in other subjects, where reading texts are of more importance. This fits the intuition that skills such as grammar need to be trained, in order to be learned. We inform how exercises are used depending both on their subject and their level in Bloom's taxonomy. And lastly we see that the average amount of time spent in the system is increasing both generally and for the individual students in all subjects, but especially for students working with languages. Furthermore, both experiments show how behaviors can have high correlation on a system-wide level, despite being uncorrelated on the individual student level. While the change of behavior for individual students was not directly analyzed in this paper (due to privacy concerns), our method allows for tracking such individual changes, hopefully helping teachers encourage optimal student behavior, e.g. by recommending training quizzes for students working with languages, or making sure that students are allowed more time to use the system in school.

In our setting, the number of clusters is fixed. It may be interesting to use an adaptive clustering strategy instead, as done in [7], as one might expect clusters to change over time. In the future, it might also be interesting to include other features, that were not available to us at this time, for instance whether a text (or quiz) have been assigned by a teacher, or whether the student reads it by themselves. For this study, we also only had access to a limited amount of data; better and more reliable results might be obtained by including more data.

5. ACKNOWLEDGMENTS

The work is supported by the Innovation Fund Denmark through the Danish Center for Big Data Analytics Driven Innovation (DABAI) project. The authors would like to thank Clio Online, and the reviewers for their thorough and insightful feedback.

6. REFERENCES

[1] Michael W. Berry, Murray Browne, Amy N. Langville, V. Paul Pauca, and Robert J. Plemmons. Algorithms and Applications for Approximate Nonnegative Matrix Factorization. *Computational Statistics & Data Analysis*, 52(1):155 – 173, 2007.

[2] Louis Faucon, Lukasz Kidzinski, and Pierre Dillenbourg. Semi-Markov model for simulating MOOC students. In *Proceedings of the 9th International Conference on Educational Data Mining (EDM)*, pages 358–363. International Educational Data Mining Society (IEDMS), 2016.

[3] Ben U. Gelman, Matt Revelle, Carlotta Domeniconi, Kalyan Veeramachaneni, and Aditya Johri. Acting the Same Differently: A Cross-Course Comparison of User Behavior in MOOCs. In *Proceedings of the 9th International Conference on Educational Data Mining (EDM)*, pages 376–381. International Educational Data Mining Society (IEDMS), 2016.

[4] Christian Hansen, Casper Hansen, Niklas Hjuler, Stephen Alstrup, and Christina Lioma. Sequence modelling for analysing student interaction with educational systems. In *Proceedings of the 10th International Conference on Educational Data Mining (EDM)*, pages 232–237. International Educational Data Mining Society (IEDMS), 2017.

[5] Stephen Hutt, Caitlin Mills, Shelby White, Patrick J. Donnelly, and Sidney K. D'Mello. The Eyes Have It: Gaze-based Detection of Mind Wandering during Learning with an Intelligent Tutoring System. In *Proceedings of the 9th International Conference on Educational Data Mining (EDM)*, pages 86–93. International Educational Data Mining Society (IEDMS), 2016.

[6] Yong-Deok Kim and Seungjin Choi. Weighted Nonnegative Matrix Factorization. In *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1541–1544, 2009.

[7] Severin Klingler, Tanja Käser, Barbara Solenthaler, and Markus Gross. Temporally Coherent Clustering of Student Data. In *Proceedings of the 9th International Conference on Educational Data Mining (EDM)*, pages 102–109. International Educational Data Mining Society (IEDMS), 2016.

[8] Cosmin Lazar and Andrei Doncescu. Non Negative Matrix Factorization Clustering Capabilities; Application on Multivariate Image Segmentation. In *Proceedings of the 3rd International Conference on Complex, Intelligent and Software Intensive Systems (CISIS)*, pages 924–929, 2009.

[9] Daniel D. Lee and H. Sebastian Seung. Algorithms for Non-negative Matrix Factorization. In *Advances in Neural Information Processing Systems 13, Papers from Neural Information Processing Systems (NIPS) 2000, Denver, CO, USA*, pages 556–562, 2000.

[10] Tao Li and Chris Ding. Non-negative matrix factorization for clustering: A survey. In *Data Clustering: Algorithms and Applications*, pages 149–176. Chapman & Hall/CRC, January 2013.

[11] Chih-Jen Lin. On the Convergence of Multiplicative Update Algorithms for Non-negative Matrix Factorization. *Trans. Neur. Netw.*, 18(6):1589–1596, 2007.

[12] Stephan Lorenzen, Ninh Pham, and Stephen Alstrup. On Predicting Student Performance Using Low-rank Matrix Factorization Techniques. In *Proceedings of the 16th European Conference on e-Learning (ECEL)*, pages 326–334. Academic Conferences and Publishing International, 2017.

[13] Farial Shahnaz, Michael W. Berry, Victor P. Pauca, and Robert J. Plemmons. Document clustering using nonnegative matrix factorization. *Information Processing & Management*, 42(2):373 – 386, 2006.

Fully Dynamic Consistent Facility Location

Anonymous Author(s)
Affiliation
Address
email

Abstract

We consider classic clustering problems in fully dynamic data streams, where data elements can be both inserted and deleted. In this context, several parameters are of importance: (1) the quality of the solution after each insertion or deletion, (2) the time it takes to update the solution, and (3) how different consecutive solutions are. The question of obtaining efficient algorithms in this context for facility location, k-median and k-means has been raised in a recent paper by Hubert-Chan et al. [WWW'18] and also appears as a natural follow-up on the online model with recourse studied by Lattanzi and Vassilvitskii [ICML'17] (i.e.: in insertion-only streams).

In this paper, we focus on general metric spaces and mainly on the facility location problem. We give an arguably simple algorithm that maintains a constant factor approximation, with $O(n \log n)$ update time, and total recourse $O(n)$. This improves over the naive algorithm which consists in recomputing a solution at each time step and that can take up to $O(n^2)$ update time, and $O(n^2)$ total recourse. Our bounds are nearly optimal: in general metric space, inserting a point takes $O(n)$ times to describe the distances to other points, and we give a simple lower bound of $O(n)$ for the recourse. Moreover, we generalize this result for the k-medians and k-means problems: our algorithms maintains a constant factor approximation in time $\widetilde{O}(n + k^2)$.

We complement our analysis with experiments showing that the cost of the solution maintained by our algorithm at any time t is very close to the cost of a solution obtained by quickly recomputing a solution from scratch at time t while having a much better running time.

1 Introduction

Clustering is a core procedure in unsupervised machine learning and data analysis. Due to the large number of applications, clustering problems have been extensively studied for several decades. The existing literature includes both very precise algorithms[1, 16, 26], and very fast ones [28]. Due to the importance of the task, clustering problems have also been studied in several computing settings, such as the streaming model [9] and the sliding-window model [6], the distributed model [3], in the dynamic model [21], and others.

Applications nowadays operate on dynamically evolving data, e.g., pictures are constantly added and deleted from picture repositories, purchases are continuously added into online shopping systems, reviews are added or being edited in retail systems, etc. Due to the scale and the dynamic nature of the data at hand, conventional algorithms designed to operate on static inputs become unable to handle the task for two main reasons. First, the running time of even the most efficient algorithms is too expensive to execute after every single change in the input data. Second, re-running a static algorithm after every update might generate solutions that differ substantially between consecutive updates, which might be undesirable for the application at hand. The number of changes in the

Submitted to 33rd Conference on Neural Information Processing Systems (NeurIPS 2019). Do not distribute.

maintained solution between consecutive updates is called the *recourse* of the algorithm. Our study is motivated by these limitations of static algorithms, or dynamic algorithms that are effective on only one of the two objectives.

Most fundamental problems in computer science have been studied in the dynamic setting. In a very high-level, a dynamic algorithm computes a solution on the initial input data and as the input undergoes insertions or/and deletions of elements, the algorithm updates the solution to reflect the current state of the data. A dynamic algorithm may allow only insertions or only deletion, or may support an intermixed sequence of insertions and deletions, in which case the algorithm is called fully-dynamic. The running time of a dynamic algorithm can either guarantee a worst-case update time after each update, or a bound on the average update time over a sequence of updates which is called amortized update bound. A dynamic algorithm with worst-case update bounds is the most desirable, and often hard to obtain, but in several applications algorithms with amortized update bounds are sufficient.

In this paper, we study fully-dynamic algorithms for classic clustering problems. In particular, we consider the facility location, the k-means, and the k-median problems in the dynamic setting. In the static case, these problems are defined as follows. Let X be a set of n points, and $d : X \times X \to \mathbb{R}$ a distance function. We assume that d is symmetric and that (X, d) forms a metric space. For the (k, p)-clustering problem, the objective function that we seek to optimize is $\mathrm{C}_p(X, S)$, where $S \subseteq X, |S| = k$. $p = 1$ gives the k-median objective, and $p = 2$ the k-means one. For the facility location problem the objective function is $\mathrm{C}(X, S)$.

$$\mathrm{C}(X, S) := \sum_{x \in X} \min_{c \in S} d(x, c) + f \cdot |S| \qquad \mathrm{C}_p(X, S) := \sum_{x \in X} \min_{c \in S} d^p(x, c),$$

All of these problems are NP-Hard, so our best hope is to design algorithms with provable approximation guarantees. In the dynamic setting, the goal is to maintain efficiently a good solution to the clustering problem at hand as the set of points undergoes element insertions and deletions. The main criterion for designing a *good* dynamic algorithm for these problems is the quality of the clustering, with respect to the optimum solution, at any given time. However, in many applications, it is equally important to maintain a *consistent* clustering, namely a clustering with bounded recourse. Lattanzi and Vassilvitskii [25] have recently considered consistent clustering problems in the online setting, where the points appear in sequence and the objective is to maintain a constant factor approximate solution while minimizing the total number of times the maintained solution changes over the whole sequence of points. Another criterion that has been much less explored but which is of high importance when dealing with massive data is the time it takes to update the solution after each update so that the solution remains a constant factor approximate solution.

1.1 Our Contribution

We present the first work that studies fully-dynamic algorithms while considering all of the three aforementioned objectives at the same time, that is, the approximation guarantee, consistency and update time. From an input perspective, we consider general metric spaces, an element of the input is thus a point in that metric space which is defined by the distances to the other points of the metric. The contribution of our paper can be summarized as follows:

• We give a fully-dynamic algorithm for the facility location problem with a constant factor approximation, a constant number of changes to the clustering at each time step, and $O(n \log n)$ update time. We moreover show that a constant number of changes per update is necessary for achieving a constant factor approximation.

• We extend the algorithm for facility location to the k-median and k-means problems. Here, our algorithm maintains a constant factor approximate solution with $\widetilde{O}(n + k^2)$[1] update time (Theorem 3.1). This is the first non-trivial result for these problems, as the only known solution for these problems was to recompute from scratch after each update: this requires time $\Omega(nk)$ for k-median and $\Omega(n^2)$ for facility location. Hence, our time bounds are significantly better than the naive approach for a large range of k.

[1] $\widetilde{O}(\cdot)$ hides polylog factors.

Empirical Analysis. We complement our study with an experimental analysis of our algorithm, on three real-world data sets, and show that it outperforms the standard approach that recomputes a solution from scratch after each update, using a fast static algorithm. Interestingly, we show that this barely impacts the approximation guarantee. At the same time, our algorithm outperforms by at least three orders of magnitude the simple-minded solutions, both in terms of running time and total number of changes made in the maintained solution throughout the update sequence.

1.2 Related Work

Online and Consistent Clustering Online algorithms for facility location were first proposed by Meyerson [29] in his seminal paper. Fotakis [14] later showed that the algorithm has a competitive ratio of $O(\log n / \log \log n)$, which is also optimal. Additionally, the algorithm has a constant competitive ratio if the points arrive in random order [29, 24]. There also exist $O(\log n)$ competitive deterministic algorithms, see [2, 13]. This was recently extended to the online model that incorporates deletions [10].

Online algorithms for clustering that are only allowed to place centers cannot be competitive. This led to the consideration of the incremental model, which allows two clusters to be merged at any given time. Work in this area includes [8, 12]. The number of reassignments (commonly referred to as *recourse*) over the execution of an incremental algorithm may be arbitrary. However, recently, Lattanzi and Vassilvitskii [25] considered the online clustering problem with bounded total recourse. They showed a lower bound of $\Omega(k \log n)$ changes over an arbitrary sequence of updates, and presented an algorithm that can maintain a constant factor approximation while limiting the total recourse to $O(k^2 \cdot \log^4 n)$. Their work differs to ours in that element can only be added, and that they do not consider optimizing the running time. In the fully dynamic case their bound on the recourse does not hold, and we moreover show that constant recourse per update is unavoidable.

Fully-Dynamic and Streaming Algorithms. Streaming algorithms for clustering can be used to obtain fast dynamic algorithms by recomputing a clustering after each update. Since streaming algorithms are highly memory compressed and typically process updates in time linear in the memory requirement, the approach automatically yields good update times. Low-memory adaptations of Meyerson's algorithm [29] turned out to be simple and particularly popular, see [7, 24, 31]. Another technique for designing clustering algorithms in the streaming models is by maintaining coresets, see the following recent survey for an overview [30]. For fully dynamic data streams, the only known algorithms for maintaining coresets for k-means and k-median in Euclidean spaces using small space and update times are due to Braverman et al. [5] and Frahling and Sohler [15]. There also exists some work on estimating the cost of Euclidean facility location in dynamic data streams, see [11, 22, 23].

For more general metrics, the problem of maintaining clusterings dynamically has been considered by Henzinger et al. [19] and Goranci et al. [17] who consider the facility location in bounded doubling dimension. The arguably most similar previous to ours is due to Hubert-Chan et al. [21]. They consider the k-center problem in general metrics in the fully dynamic model. Here, they were able to maintain a constant factor approximation with update time $O(k \log n)$ [2]. Whether an algorithm in the fully dynamic model with low recourse and update times exists, was left as an open problem.

1.3 Preliminaries

We assume that we are given some finite metric space (X, d), where X is the set of points and $d : X \times X \to \mathbb{R}_{\geq 0}$ a distance function. Every entry $d(a, b)$ is stored in a (symmetric) $n \times n$ matrix D. Our algorithms work in the distance oracle model, which assumes that we can access any entry of D in constant time.

Our input consists of tuples $(X, \mathbb{R}^n_{\geq 0}, \{-1, 1\})$. The first coordinate is the identifier of some point $p \in X$, the second coordinate is the column/row vector in D associated with p, and the last coordinate signifies insertion (1) or deletion (-1). We assume that the stream is consistent, which means that no point can be deleted without having been previously inserted. The adversary generating the point sequence is called adaptive if he can modify the sequence depending on the algorithm's

[2] Under the common assumption that the ratio longest distance / shortest distance of the metric is polynomially bounded.

choices. Throughout the paper, we let X^t be the set of points present at time t, n be the total number of steps, and $n^* := \sup_{t \in 1,\dots,n} |X^t|$ be the maximum number of points present at the same time. All our results could be phrased in term of $|X^t|$, but for simplicity we present them in terms of n^*.

Roadmap. Our paper is organized as follows. In Section 2, we describe our algorithm for fully dynamic facility location. Section 3 extends these results to k-median and k-means clustering. We conclude with an experimental evaluation of our algorithms in Section 4 on real-world benchmarks. All omitted proofs can be found in the supplementary material.

2 Dynamic Facility Location

The goal of this section is to prove the following theorem.

Theorem 2.1. *There exist a randomized algorithm that, given a metric space undergoing insertion and deletions of points, maintains a set of center S^t such that :*

- *each update is processed in time $O(n^* \log(n^*))$ with probability $1 - 1/n^*$*
- *at any given time t, $C(X^t, S^t) = O(1)C(X^t, OPT^t)$ with probability $1 - 1/n^*$*
- *$\sum_{t=0}^{n} |S^t \Delta S^{t+1}| = O(n)$, i.e. the amortized recourse is $O(1)$ per step.*

The proof is divided into several lemmas: we first study how the optimal cost behaves upon dynamic updates, and we exhibit then an algorithm that maintains a solution whose cost evolves in a similar way as the optimum.

Though perhaps counter intuitive, removing a point from the input in a finite metric may increase the cost of a clustering, if one cannot locate a center there anymore. We show in the supplementary material that this increase is bounded by a factor 2: this leads to the following lemma, which bounds the evolution of the optimal cost.

Lemma 2.2. *Let OPT^0 be the optimal cost of an initial metric space X. After an arbitrary sequence of n_i insertions and n_d deletions of points in X, the optimal solution OPT^1 satisfies $OPT^0/2 - n_d \cdot f \leq OPT^1 \leq 2(OPT^0 + n_i \cdot f)$*

Maintaining a solution during few steps. We now turn on designing an algorithm competitive with the optimal solution, showing first how to deal with a small number of steps. In order to process deletions, we define the notion of *substitute* centers: given a function s mapping every center from the initial solution to a center in the current one, we say that $s(c)$ substitutes c. Initially, $s(c) = c$. When a center c is deleted, the algorithm opens a replacement center c_r, and updates the function s: $s(s^{-1}(c)) = c_r$.

The algorithm is as follows: when a point x is inserted, it is open as a facility, and for convenience we define $s(x) = x$. When a point x is deleted, we have two cases: either x was not an open facility, in which case the algorithm does nothing, or x was a facility. In the latter case, let $c = s^{-1}(x)$: the algorithm opens the closest point c' of c in X^0 that is still part of the metric, and set $s(c) = c'$. This choice of c' ensures that, for all points x in the current metric space, $d(c', c) \leq d(x, c)$.

Lemma 2.3. *Starting from any metric space (X^0, d) and an α-approximation with cost Θ, the algorithm described above maintains a $(8\alpha + 4)$-approximation during $\frac{\Theta}{4\alpha f}$ steps, with recourse $O(1)$ and time $O(n^*)$ at each step.*

Proof. This algorithm opens at most one new facility at every step: the recourse is thus at most 1. The time to process an insertion is constant, and the time to process a deletion is at most $O(n^*)$ (the time required to compute the closest point to x).

We now analyse the cost of the solution produced after t steps. Since the recourse is at most 1 per step, the cost of open facilities increases by at most $t \cdot f$. Since every inserted points is opened as a center, it does not contribute to the connection cost: this cost changes therefore only by deletions of points from X^0. Similarly to Lemma 2.2, one can show that the connection cost of a point $x \in X^0$ at most doubles. More formally, let $c \in X^0$ be the center that serves x in the initial solution. Let $c' = s(c)$ be the center that substitutes c in the current metric. By the choice of c' and triangle

inequality, it holds that $d(x,c') \leq d(x,c) + d(c,c') \leq 2d(x,c)$. Hence, the total serving cost is at most twice as expensive as in the initial solution. The cost at time t is therefore at most $2\Theta + t \cdot f$.

Let $t \leq \frac{\Theta}{4\alpha f}$, and OPT^0 the optimal cost in the initial state. By Lemma 2.2, the optimal cost at time t is at least $\text{OPT}^0/2 - t \cdot f \geq \text{OPT}^0/4$, since by definition of α it holds $t \leq \frac{\text{OPT}^0}{4f}$. Moreover, the cost of our algorithm at time t is at most $\Theta(2 + \frac{1}{\alpha}) \leq \text{OPT}^0(2\alpha + 1)$. Combining the two inequalities gives that our algorithm is a $(8\alpha+4)$-approximation for all $t \leq \frac{\Theta}{4\alpha f}$, which concludes the proof. □

We remark that the parameters can be optimized: for instance, with a suitable data structure, the time to find a substitute center can be logarithmic; however, this is dominated by the complexity of finding the initial α-approximation.

Maintaining a solution for any number of steps. We combine Lemma 2.3 with a classic static $O(1)$-approximation algorithm, namely Meyerson's algorithm, to prove Theorem 2.1.

Proof of Theorem 2.1. We summarize here the useful properties of Meyerson algorithm, and refer to the supplementary material for more details. The algorithm processes the input points in a random order, opening each point x with probability $d(x,F)/f$ (where F is the set of previously opened facilities). If the algorithm opens k facility, its running time is $O(kn^*)$, and the cost is at least $\Theta \geq kf$. Hence, the running time is $O(\Theta/f \cdot n^*)$. Moreover, one can assume that the cost is always at most $n^* f$ (by opening a facility at every point).

We say that a run of Meyerson's algorithm is *good* if it yields a $O(1)$-approximate solution. By the analysis in [29], a run is good in expectation (where the randomness comes from the random ordering of points): hence, by running $\log(2n^*)$ independent copies of the algorithm, at least one run is good with probability $1 - (1/n^*)^2$. We let α be the approximation constant of this algorithm.

Our main algorithm is therefore the following: start with a solution given by Meyerson's algorithm of cost Θ, use Lemma 2.3 to maintain a solution during $\frac{\Theta}{4\alpha f}$ steps, and then recompute from scratch. We call the intervals in between recomputations *periods*, and note that they are random objects: the length of a period is determined by the cost of its initial solution, which is a random variable.

We first analyze the running time of this algorithm. Within one period, Lemma 2.3 ensures that the running time is $O(n^*)$ per step. Moreover, the running time of the initial recomputation is $O(\Theta/f \cdot n^* \log n^*)$, and the length of the period is $\Omega(\Theta/f)$. Therefore the amortized running time is $\widetilde{O}(n^*)$ per update. Since the initial recourse is $O(\Theta/f)$, the same argument proves that the recourse is amortized $O(1)$ per update.

We aim at using again Lemma 2.3 to prove that, at a given time t, the solution is a constant factor approximation. For this, let P be the period P in which t lies. If the period is good, then Lemma 2.3 concludes. Unfortunately, the fact that t is in P is not independent of P being good (for instance, if P is very long it is unlikely to be good). However, note that the starting time of P cannot be before $t - n^*$: indeed, a period lasts at most for $\frac{\Theta}{4\alpha f} \leq \frac{n^* f}{4\alpha f} \leq n^*$ steps. Hence, if we condition on all periods starting between $t - n^*$ and t, Lemma 2.3 applies and the solution at time t is a constant factor approximation. Since any period is good with probability $1 - (1/n^*)^2$, all periods between $t - n^*$ and t are good with probability $1 - 1/n^*$ by union bound. This concludes the proof. □

The algorithm sketched in the previous proof can be transformed so that the complexity becomes $\widetilde{O}(n^*)$ in the worst case, by spreading the recomputation over several steps (see supplementary material). Moreover, randomization is not needed in order to maintain the solution (only to compute a starting approximation): hence the algorithm works against an adaptive adversary.

We conclude this section by showing that our algorithm is (up to a logarithmic factors) optimal both for update time and recourse.

Proposition 2.4. *Any algorithm maintaining a $O(1)$-approximation for Facility Location must have an amortized update time $\Omega(n^*)$ and total recourse $\Omega(n)$, where n the total number of steps.*

3 Dynamic k-Median and k-Means in Linear Time

In this section we adapt the algorithm from Section 2 to handle the stricter problem of k-Median and k-Means. For simplicity, we call (k,p)-clustering the problem of finding k centers that minimize C_p ($p=1$ for k-Median and $p=2$ for k-Means).

Roughly speaking, our algorithm works as follows. We use an adaptation of the algorithm from Section 2 to maintain a coreset $\mathcal{R}$ of $\widetilde{O}(k)$ points that contain a constant factor approximate solution for the (k,p)-clustering problem. Then, we apply a constant factor approximation algorithm for the metric (k,p)-clustering problem on the maintained coreset (e.g., we can use a quadratic-time local-search algorithm, see [18]). This yields the following theorem.

Theorem 3.1. *There exists a randomized algorithm that, given a metric space undergoing insertions and deletions of points, maintains a set of centers S^t with $\widetilde{O}(n^*+k^2)$ update time such that, for any time t, $C_p(X^t,S^t)=O(1)\cdot C_p(X^t,OPT^t)$.*[3]

The remainder of this section is devoted in proving Theorem 3.1. The main hurdle in applying the framework from Section 2 is that the optimum solution can change drastically with the addition or deletion of a point, and it is therefore not easy to adapt the previous amortization argument. To overcome this barrier, we make use of the following lemma, from [8] and [25]:

Lemma 3.2. *Let L be some integer. With probability $1/2$, running Meyerson's algorithm for Facility Location with $f=\frac{L}{k(1+\log n^*)}$ gives a set S of $4k\cdot(1+\log n^*)\cdot(2^{2p+1}\cdot\frac{C_p(X,OPT)}{L}+1)$ centers such that $C_p(X,S)\le L+4\cdot C_p(X,OPT)$.*

For completeness, we provide the pseudocode of Meyerson's algorithm, adapted for our purpose, in Procedure $MeyersonCapped$. The lemma implies that, if we know a value L that approximates OPT within a factor 2, Procedure $MeyersonCapped$ computes a set of points $\mathcal{R}$ and an assignment of points ϕ such that $\sum d(p,\phi(p))\le 6\cdot\mathrm{C}_p(X,\mathrm{OPT})$ with probability $1/2$. This probability can be boosted to $1-(1/n^*)^2$ by taking the union of $q=O(\log n^*)$ independent copies of the algorithm. Therefore for all $i=1,...,q$, our algorithm will use this lemma assuming $\mathrm{C}_p(X,\mathrm{OPT})\in[2^i,2^{i+1})$, and taking $L=2^i$. This provides, for all i, a set $\mathcal{R}_i$ of $O(k\log^2 n^*)$ centers.

It remains to maintain those sets dynamically. Similarly to Section 2, we use the solution computed by Procedure $MeyersonCapped$ for the subsequent k steps, so that we can amortize the update-time bound. However, for (k,p)-clustering it is not possible to bound the cost of OPT after a few steps. We overcome this obstacle by updating the sets $\mathcal{R}_i$ more carefully. More precisely, let $\mathcal{R}_i^t$ be the (updated) set $\mathcal{R}_i$ after t updates of the algorithm. The algorithm ensures the following invariant:

Invariant 3.3. *The set $\mathcal{R}_i^t$ has size $O(k\log^2 n^*)$ and, with high probability, there exists i such that $C_p(X^t,\mathcal{R}_i^t)=O(1)\cdot C_p(X^t,OPT^t)$.*

For this, initialize $\mathcal{R}_i$ to be the union of the outputs of $q=O(\log n^*)$ independent executions of $MeyersonCapped(L_i,X)$, for $L_i=2^i$ and $i=1,...,q$. The algorithm updates these sets during k steps before recomputing them from scratch. In the case of a point insertion, it suffices to add the new point to all $\mathcal{R}_i$: over k steps, this changes the cardinality by at most k while the cost remains the same, and therefore the two conditions are met. The case of a point deletion requires more work. The idea is, as in Section 2, to replace the deleted center by its closest point. However, this is not enough to ensure Invariant 3.3: this is taken care of by Procedure $DeletePoint$, which finds the next point in X^t that $MeyersonCapped$ would open, if there was no constraint on the size of $\mathcal{R}$.

We are now ready to describe our fully-dynamic algorithm for maintaining a constant-approximate solution to the (k,p)-clustering problem. The algorithm uses Procedures $MeyersonCapped$ and $DeletePoint$ as subroutines to build and maintain the sets $\mathcal{R}_i$, and after each update calls the static constant-approximate algorithm to compute an approximate solution S_i^t on each weighted instance $\mathcal{R}_i$ (where the weight of each point $p\in\mathcal{R}_i$ corresponds to the number of points of X^t assigned to p by the function ϕ_i, computed in Procedures $MeyersonCapped$ and $DeletePoint$. The algorithm chooses to maintain the solution S_i^t that minimizes $\mathrm{C}_p(\mathcal{R}_i^t,S_i^t)$, that is, $S^t=S_i^t$ for $i=\arg\min_i\{\mathrm{C}_p(\mathcal{R}_i^t,S_i^t)\}$. The algorithm is formally stated in the supplementary material,

[3] We assume (as in [25]) that the minimum distance in the metric is 1 and the maximum Δ is bounded by a polynomial in n^*. Alternatively, our bounds can be stated with $\log\Delta$ instead of $\log n^*$.

Input: An integer L, a set of points X
Output: A set of centers $\mathcal{R}$, an assignment ϕ of point to centers, and t_l the id of the last center opened

Let $\mathcal{R} = \emptyset$, $x_1, ..., x_{|X|}$ be a random order on the points of X
for all $i \in \{1, ..., |X|\}$ **do**
  if $|\mathcal{R}| < 4k \cdot (1 + \log n^*) \cdot (2^{2p+2} + 1)$ **then**
    add x_i to $\mathcal{R}$ with probability $\frac{d(x_i,\mathcal{R})^p k(1+\log n)}{L}$
    if $|\mathcal{R}| = 4k \cdot (1 + \log n^*) \cdot (2^{2p+2} + 1)$ **then**
      $t_l = i$
    end if
  end if
  $\phi(x_i) = \arg\min_{c \in \mathcal{R}} \{d(x_i, c)^p\}$
end for

(a) MeyersonCapped(L, X)

Input: Integers L and t_l, a set of points X and a set of centers $\mathcal{R}$
Output: Updated $\mathcal{R}$ and t_l, an assignment ϕ of points to centers

for all $i \in \{t_l, ..., |X|\}$ **do**
  if No center was opened yet **then**
    add x_i to $\mathcal{R}$ with probability $\frac{d(x_i,\mathcal{R})^p k(1+\log n^*)}{L}$
    if x_i is added to $\mathcal{R}$ **then**
      $t_l = i$, $\phi(x_i) = x_i$
    end if
  else {Update ϕ}
    $\phi(x_i) = \arg\min_{z \in \{\phi(x_i), x_{t_l}\}} \{d(x_i, z)^p\}$
  end if
end for

(b) DeletePoint($L, t_l, X, \mathcal{R}$). L is the value with which we approximate $C_p(X, \text{OPT})$ and t_l is the last time MeyersonCapped opened a center.

together with the proof of Invariant 3.3. The next lemma, together with Invariant 3.3, shows that S^t can be used as a solution for the entire set X^t.

Lemma 3.4. *Let $OPT(\mathcal{R}_i^t)$ be the optimal solution in the weighted set $\mathcal{R}_i^t$. Then it holds that $C_p(X^t, OPT(\mathcal{R}_i^t)) \leq 2^{3p-1}(C_p(X, \mathcal{R}_i^t) + C_p(X^t, OPT^t))$*

This proves the second part of Theorem 3.1: for i such that $C_p(X^t, \mathcal{R}_i^t) = O(1) \cdot C_p(X^t, \text{OPT}^t)$, the solution computed on the set $\mathcal{R}_i^t$ is a good approximation of the optimal solution, and therefore the algorithm maintains a constant factor approximation. The bound on the running time being similar to the one of Section 2, we provide it in the supplementary material.

4 Empirical Analysis

In this section, we evaluate our algorithm for facility location experimentally. Recall that we aim to strike a balance between (1) overall running time, (2) the cost of the solution, (3) the total recourse. Our implementation follows the framework outline in Theorem 2.1. As part of the recomputation step between two periods, we run 5 independent executions of Meyerson's algorithm, and selecting the execution with lowest cost. The updates within a period are handled by assigning to closest center if distance is less than f or otherwise open a new center at the point, and we simply remove a client if it gets deleted. We compare our algorithm against two variants of Meyerson's algorithm. The first one, termed *MeyersonRec*, re-runs Meyerson at every single update. The second, termed *MeyersonSingle*, consists of a single execution of Meyerson for all updates, where deletions are handled by just removing the distance cost of the deleted point. Following Hubert-Chan et al. [21], we incorporate deletions by considering a sliding window over the data set. A point is inserted/deleted when it enters/exists the window, respectively.

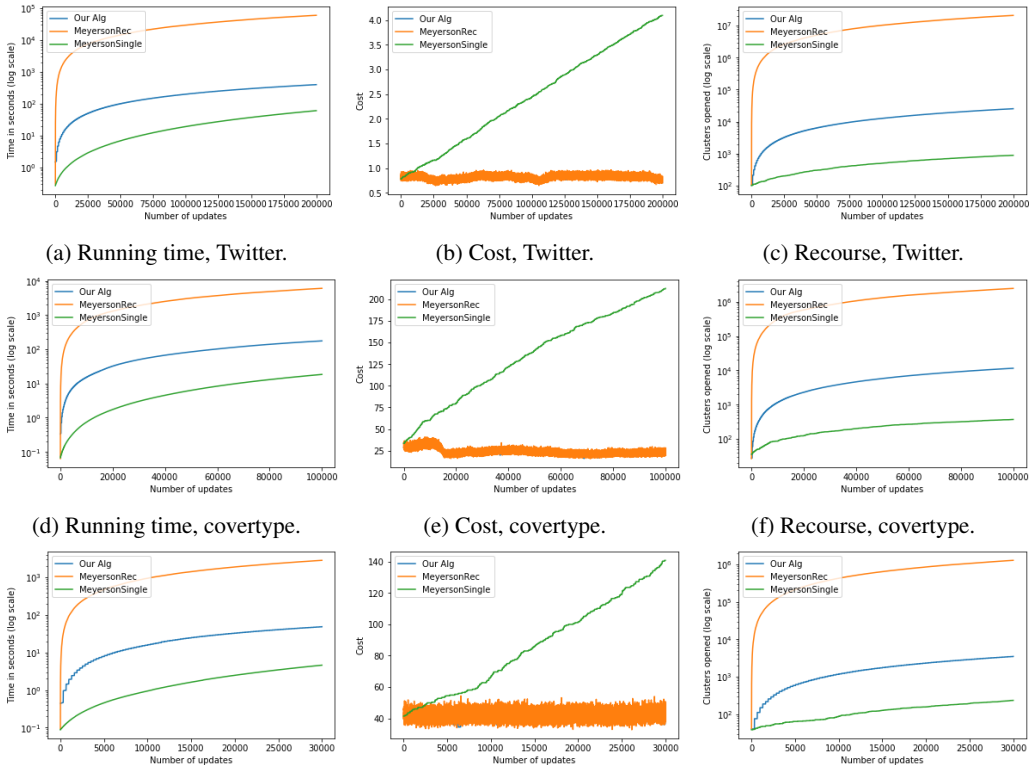
Data Set and Setup. We consider the following data sets, equipped with the Euclidean distance.

• The Twitter data set [20], considered by [21], consists of 4.5 million geotagged tweets in 3 dimensions (longitude, latitude, timestamp). We restricted our experiments to the first 200K tweets.

• The COVERTYPE data set [4], considered by [25], from the UCI repository with 581K points and 54 dimensions. We restricted our experiments to the first 100K points and 10 dimensions (the ones we believed to be appropriate for an Euclidan metric).

• The USCensus1990 data set [27] from the UCI repository has 69 dimensions and 2.5 million points. We restricted our experiments to the first 30K points.

We restricted the number of points considered due to time constraints. Since larger data sets typically have more complicated ground truths, we used a larger windows for them containing more samples. To avoid overfitting, we also adjusted the facility cost depending on the window size, i.e. for larger windows a lower opening cost per facility. For COVERTYPE and USCensus1990, we used a window size of 5000 points and a facility cost of 0.5; for Twitter, the window size was 10000 and the facility cost 0.004. All our codes are written in Python. The experiments were executed on a Windows 10 machine with processor: Intel(R) Core(TM) i7-8700K CPU @ 3.70GHz, 3701 Mhz, 6 cores, 12 Logical processors, and 16 GB RAM.



(a) Running time, Twitter. (b) Cost, Twitter. (c) Recourse, Twitter.

(d) Running time, covertype. (e) Cost, covertype. (f) Recourse, covertype.

(g) Running time, USCensus1990. (h) Cost, USCensus1990. (i) Recourse, USCensus1990.

Figure 2: A comparison of the algorithms we consider in terms of running time (left column), cost of the solution (middle column), and recourse (right column).

Results. In all three data sets we generally observed the same behavior in terms of running time, cost, and the number of clusters opened, see Figure 2. Our algorithm is 100 times faster than MeyersonRec. Compared to MeyersonSingle, our algorithm is slower initially. When the number of processed points becomes very large, the running time of MeyersonSingle deteriorates comparatively, as it never removes a facility once it has been opened: the time to compute the distance to the set of facilities is therefore increasing (see Figure 1 in supplementary material). The cost of MeyersonSingle generally has a linear dependency on the number of updates, though the slope is very gentle. This is also what our algorithm takes advantage off, broadly speaking by approximating the curve with a step function (adapted to handle insertions and deletions). The cost of our algorithm and MeyersonRec is basically indistinguishable, and in certain cases our algorithm fares even slightly better. The recourse of our algorithm is expectedly better than MeyersonRec by a wide margin, and signficantly worse than MeyersonSingle.

Finally, we ran our algorithm with multiple choices of facility cost f, and we observed that the recourse is almost independent of the both cost and running time of the algorithm, and only depends on the number of updates. This is consistent with tracking evolving data in time, where the underlying ground truth clustering also evolves in time.

References

[1] S. Ahmadian, A. Norouzi-Fard, O. Svensson, and J. Ward. Better guarantees for k-means and euclidean k-median by primal-dual algorithms. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 61–72, Oct 2017.

[2] A. Anagnostopoulos, R. Bent, E. Upfal, and P. V. Hentenryck. A simple and deterministic competitive algorithm for online facility location. *Inf. Comput.*, 194(2):175–202, 2004.

[3] O. Bachem, M. Lucic, and A. Krause. Distributed and provably good seedings for k-means in constant rounds. In D. Precup and Y. W. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 292–300, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

[4] J. A. Blackard, D. J. Dean, and C. W. Anderson. Covertype data set, https://archive.ics.uci.edu/ml/datasets/covertype.

[5] V. Braverman, G. Frahling, H. Lang, C. Sohler, and L. F. Yang. Clustering high dimensional dynamic data streams. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pages 576–585, 2017.

[6] V. Braverman, H. Lang, K. Levin, and M. Monemizadeh. Clustering problems on sliding windows. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms*, pages 1374–1390. Society for Industrial and Applied Mathematics, 2016.

[7] V. Braverman, A. Meyerson, R. Ostrovsky, A. Roytman, M. Shindler, and B. Tagiku. Streaming k-means on well-clusterable data. In *SODA*, pages 26–40, 2011.

[8] M. Charikar, C. Chekuri, T. Feder, and R. Motwani. Incremental clustering and dynamic information retrieval. *SIAM J. Comput.*, 33(6):1417–1440, 2004.

[9] M. Charikar, L. O'Callaghan, and R. Panigrahy. Better streaming algorithms for clustering problems. In *Proceedings of the Thirty-fifth Annual ACM Symposium on Theory of Computing*, STOC '03, pages 30–39, New York, NY, USA, 2003. ACM.

[10] M. Cygan, A. Czumaj, M. Mucha, and P. Sankowski. Online facility location with deletions. In *26th Annual European Symposium on Algorithms, ESA 2018, August 20-22, 2018, Helsinki, Finland*, pages 21:1–21:15, 2018.

[11] A. Czumaj, C. Lammersen, M. Monemizadeh, and C. Sohler. (1+)-approximation for facility location in data streams. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 1710–1728, 2013.

[12] D. Fotakis. Incremental algorithms for facility location and k-median. *Theor. Comput. Sci.*, 361(2-3):275–313, 2006.

[13] D. Fotakis. A primal-dual algorithm for online non-uniform facility location. *J. Discrete Algorithms*, 5(1):141–148, 2007.

[14] D. Fotakis. On the competitive ratio for online facility location. *Algorithmica*, 50(1):1–57, 2008.

[15] G. Frahling and C. Sohler. Coresets in dynamic geometric data streams. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC)*, pages 209–217, 2005.

[16] T. F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293 – 306, 1985.

[17] G. Goranci, M. Henzinger, and D. Leniowski. A tree structure for dynamic facility location. In *26th Annual European Symposium on Algorithms, ESA 2018, August 20-22, 2018, Helsinki, Finland*, pages 39:1–39:13, 2018.

[18] A. Gupta and K. Tangwongsan. Simpler analyses of local search algorithms for facility location. *CoRR*, abs/0809.2554, 2008.

[19] M. Henzinger, D. Leniowski, and C. Mathieu. Dynamic clustering to minimize the sum of radii. In *25th Annual European Symposium on Algorithms, ESA 2017, September 4-6, 2017, Vienna, Austria*, pages 48:1–48:10, 2017.

[20] T. Hubert Chan, A. Guerqin, and M. Sozio. Twitter data set, https://github.com/fe6Bc5R4JvLkFkSeExHM/k-center.

[21] T. Hubert Chan, A. Guerqin, and M. Sozio. Fully dynamic k-center clustering. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web, WWW 2018, Lyon, France, April 23-27, 2018*, pages 579–587, 2018.

[22] P. Indyk. Algorithms for dynamic geometric problems over data streams. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 373–380, 2004.

[23] C. Lammersen and C. Sohler. Facility location in dynamic geometric data streams. In *Algorithms - ESA 2008, 16th Annual European Symposium, Karlsruhe, Germany, September 15-17, 2008. Proceedings*, pages 660–671, 2008.

[24] H. Lang. Online facility location against a t-bounded adversary. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1002–1014, 2018.

[25] S. Lattanzi and S. Vassilvitskii. Consistent k-clustering. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pages 1975–1984, 2017.

[26] S. Li. A 1.488 approximation algorithm for the uncapacitated facility location problem. *Information and Computation*, 222:45 – 58, 2013. 38th International Colloquium on Automata, Languages and Programming (ICALP 2011).

[27] C. Meek, B. Thiesson, and D. Heckerman. Us census data (1990), http://archive.ics.uci.edu/ml/datasets/US+Census+Data+(1990).

[28] R. R. Mettu and C. G. Plaxton. Optimal time bounds for approximate clustering. *Machine Learning*, 56(1):35–60, Jul 2004.

[29] A. Meyerson. Online facility location. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 426–431, 2001.

[30] A. Munteanu and C. Schwiegelshohn. Coresets-methods and history: A theoreticians design pattern for approximation and streaming algorithms. *KI*, 32(1):37–53, 2018.

[31] M. Shindler, A. Wong, and A. Meyerson. Fast and accurate k-means for large datasets. In *NIPS*, pages 2375–2383, 2011.

One-Way Trail Orientations

Anders Aamand[*1], Niklas Hjuler[†1], Jacob Holm[*1], and Eva Rotenberg[‡2]

[1]University of Copenhagen
[2]Technical University of Denmark

Abstract

Given a graph, does there exist an orientation of the edges such that the resulting directed graph is strongly connected? Robbins' theorem [Robbins, Am. Math. Monthly, 1939] asserts that such an orientation exists if and only if the graph is 2-edge connected. A natural extension of this problem is the following: Suppose that the edges of the graph are partitioned into trails. Can the trails be oriented consistently such that the resulting directed graph is strongly connected?

We show that 2-edge connectivity is again a sufficient condition and we provide a linear time algorithm for finding such an orientation.

The generalised Robbins' theorem [Boesch, Am. Math. Monthly, 1980] for mixed multigraphs asserts that the undirected edges of a mixed multigraph can be oriented to make the resulting directed graph strongly connected exactly when the mixed graph is strongly connected and the underlying graph is bridgeless.

We consider the natural extension where the undirected edges of a mixed multigraph are partitioned into trails. It turns out that in this case the condition of the generalised Robbin's Theorem is not sufficient. However, we show that as long as each cut either contains at least 2 undirected edges or directed edges in both directions, there exists an orientation of the trails such that the resulting directed graph is strongly connected. Moreover, if the condition is satisfied, we may start by orienting an arbitrary trail in an arbitrary direction. Using this result one obtains a very simple polynomial time algorithm for finding a strong trail orientation if it exists, both in the undirected and the mixed setting.

[*]This research is supported by Mikkel Thorup's Advanced Grant DFF-0602-02499B from the Danish Council for Independent Research under the Sapere Aude research career programme.

[†]This work is supported by the Innovation Fund Denmark through the DABAI project.

[‡]This research was conducted during the fourth author's time as a PhD student at University of Copenhagen.

1 Introduction and motivation

Suppose that the mayor of a small town decides to make all the streets one-way such that it is possible to get from any place to any other place without violating the orientations of the streets[1]. If all the streets are initially two-way then Robbins' theorem [10] asserts that this can be done exactly when the corresponding graph is 2-edge connected. If, on the other hand some of the streets were already one-way in the beginning then the generalised Robbins' theorem by Boesch [1] states that it can be done exactly when the corresponding "mixed" graph is strongly connected and the underlying graph is bridgeless.

However, the proofs of both of these results assume that every street of the city corresponds to exactly one edge in the graph. This assumption hardly holds in any city in the world and therefore a more natural assumption is that every street corresponds to a trail (informally, a potentially self-crossing path) in the graph and that the edges of each trail must be oriented consistently[2].

In this paper we consider such graphs having their edges partitioned into trails. We prove that the trails can be oriented to make the resulting directed graph strongly connected exactly if the initial graph is 2-edge connected (note that this is precisely the condition of Robbins' theorem).

Not only do we show that the strong trail orientation problem in undirected 2-edge connected graphs always has a solution, we also provide a linear time algorithm for finding such an orientation. In doing so, we use an interesting combination of techniques that allow us to reduce to a graph with a number of 3-edge connected components that is linear in the number of edges. Using that the average size of these components is constant and that we can piece together solutions for the individual components we obtain an efficient algorithm.

Finally, we will consider the generalised Robbins' theorem in this new setting by allowing some edges to be oriented initially and supposing that the remaining edges are partitioned into trails. We will show that if each cut (V_1, V_2) in the graph has either at least 2 undirected edges going between V_1 and V_2 or at least 1 directed edge in each direction then it is possible to orient the trails making the resulting graph strongly connected. In fact, we show that if this condition is satisfied we may start by orienting an *arbitrary* trail in an *arbitrary* direction. Although this condition is not necessary it does give a simple algorithm for finding a trail orientation if it exists. Indeed, initially the graph may contain undirected edges that are *forced* in one direction by some cut. For finding a trail orientation if it exists we can thus orient forced trails in the forced direction. If there are no forced trails we orient any trail arbitrarily.

Note that in the mixed setting the feasibility depends on the trail decomposition which is not the case for the other results. That the condition from the generalised Robbins' theorem is not sufficient can be seen from Figure 1.

Earlier methods Several methods have already been applied for solving orientation problems in graphs where the goal is to make the resulting graph strongly connected.

One approach used by Robbins [10] is to use that a 2-edge connected graph has an *ear-decomposition*. An ear decomposition of a graph is a partition of the set of edges into a cycle C and paths $P_1, \ldots, P_t$ such that P_i has its two endpoints but none of its internal vertices on $C \cup \left(\bigcup_{j=1}^{i-1} P_j\right)$. Given the existence of an ear decomposition of a 2-edge connected graph it is easy to prove Robbins'

[1]The motivation for doing so is that the streets of the town are very narrow and thus it is a great hassle when two cars unexpectedly meet.

[2]This version of the problem was given to us through personal communication with Professor Robert E. Tarjan.

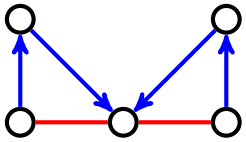


Figure 1: The graph is strongly connected and the underlying graph is 2-edge connected, but irrespective of the choice of orientation of the red trail, the graph will no longer be strongly connected.

theorem. Indeed, any choice of consistent orientations of the paths and the cycle gives a strongly connected graph.

A second approach introduced by Tarjan [4] gives another simple proof of Robbins' theorem. One can create a DFS tree in the graph rooted at a vertex v and orient all edges in the DFS tree away from v. The remaining edges are all back edges (see [4]) and are oriented towards v. It is easily verified that this gives a strong orientation if the graph is 2-edge connected. A similar approach was used by Chung et al. [2] in the context of the generalized Robbins' theorem for mixed multigraphs.

The above methods not only prove Robbins' theorem, they also provide linear time algorithms for finding strong orientations of undirected or mixed multigraphs.

However, none of the above methods have proven fruitful in our case. In case of the ear decomposition we would need one that is somehow compatible with the partitioning into trails, and this seems hard to guarantee. Similar problems appear when trying a DFS-approach. Neither does the proof by Boesch [1] of Robbins' theorem for mixed multigraphs generalise to prove our result. Most importantly, the corresponding theorem would no longer be true for trail orientations as is shown by the example in Figure 1.

Since the classical linear time algorithms rely on ear-decompositions and DFS searches, and since these approaches do not immediately work for trail partitions, our linear time algorithm will be a completely new approach to solving orientation problems.

Structure of the paper The structure of this paper is as follows. In section 3 we prove our generalisation of Robbins' theorem for undirected graphs partitioned into trails. In section 4 we study the case of mixed graphs. Finally in section 5 we provide our linear time algorithm for trail orientation in an undirected graph.

2 Preliminaries

Let us briefly review the concepts from graph theory that we will need.

A graph having some subset of its edges oriented is said to be a *mixed* graph. We will write $\{u, v\}$ for an undirected edge between u and v and (u, v) for an edge directed from u to v.

A *walk* in a graph is an alternating sequence of vertices and edges $v_0, e_1, v_1, e_2, \ldots, v_k$, such that for $1 \leq i \leq k$ the edge e_i has v_{i-1} and v_i as its two endpoints. In a directed or mixed graph we further require that either e_i be undirected or directed from v_{i-1} to v_i

A *trail* is a walk without repeated edges. A *path* is a trail without repeated vertices (except possibly $v_0 = v_k$). Finally, a *cycle* is a path for which $v_0 = v_k$.

Next, a mixed multigraph $G = (V, E)$ is called *strongly connected* if for each pair of vertices $u, v \in V$ there exists a walk from u to v. In case the graph is undirected this is equivalent to

saying that it consists of exactly one connected component. If $A \subseteq V$ we will say that A is *strongly connected in* G if for each pair of vertices $u, v \in A$ there is a walk in G from u to v.

A *cut* or *edge-cut* (V_1, V_2) in a graph is a partition of its vertices into two non-empty subsets V_1, V_2. We recall the definition of k-edge connectivity. A graph $G = (V, E)$ is said to be *k-edge connected* if and only if $G' = (V, E - X)$ is connected for all $X \subseteq E$ where $|X| < k$. A trivially equivalent condition is that each cut (V_1, V_2) in the graph has at least k edges going between V_1 and V_2.

Finally, if $G = (V, E)$ is a mixed multigraph and $A \subseteq V$ we define G/A to be the graph obtained by contracting A to a single vertex (maintaining duplicate edges and self-loops) and $G[A]$ to be the subgraph of G induced by A. The following simple observation will be used repeatedly in this paper.

Observation 2.1. *If $G = (V, E)$ is k-edge connected and $A \subseteq V$ then G/A is k-edge connected. Also, if G is a strongly connected mixed multigraph then G/A is too.*

3 Robbins Theorem Revisited

We are now ready to state and prove our generalisation of Robbins' theorem.

Theorem 3.1. *Let $G = (V, E)$ be an undirected multigraph with E partitioned into trails. An orientation of each trail such that the resulting directed graph is strongly connected exists if and only if G is 2-edge connected.*

We note that this theorem could also be proven using a general result from Király and Szigeti [6] which relies on theorems by Nash-Williams [9]. However, our proof is significantly simpler (in fact we believe that restating the theorem by Király and Szigeti and explaining the reduction would be more cumbersome) and more suitable for constructing algorithms.

Proof. If G is not 2-edge connected, such an orientation obviously doesn't exist, so we only need to prove the converse. Suppose therefore that G is 2-edge connected.

Our proof is by induction on the number of edges in G. If there are no edges, the graph consists of a single vertex, and the statement is obviously true. Assume now the statement holds for all graphs with strictly fewer edges than G. Pick an arbitrary edge e that sits at the end of its corresponding trail.

If $G - e$ is 2-edge connected, then by the induction hypothesis there is a strong orientation of $G - e$ that respects the trails of G. Such an orientation clearly extends to the required orientation of G.

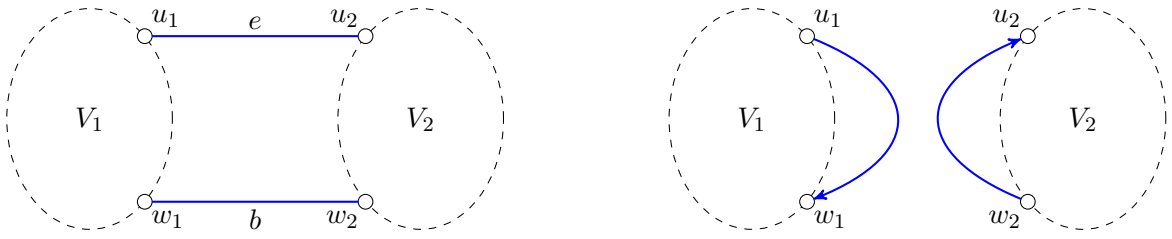


Figure 2: A 2-edge cut and the two graphs $G_1 = G[V_1] \cup \{\{u_1, w_1\}\}$ and $G_2 = G[V_2] \cup \{\{u_2, w_2\}\}$. The orientations of the two new edges are obtained from the strong trail orientations of G_1 and G_2.

If $G - e$ is not 2-edge connected, there exists a bridge b in $G - e$ (see Figure 2). Let V_1, V_2 be the two connected components of $G - \{e, b\}$, and let $e = \{u_1, u_2\}$ and $b = \{w_1, w_2\}$ such that for $i \in \{1, 2\}$, $u_i, w_i \in V_i$ (note that we don't necessarily have that u_i and w_i are distinct for $i \in \{1, 2\}$).

Now for $i \in \{1, 2\}$ construct the graph $G_i = G[V_i] \cup \{\{u_i, w_i\}\}$ (note that $\{u_i, w_i\}$ might be a self-loop but this causes no problems for the argument), and define the trails in G_i to be the trails of G that are completely contained in G_i, together with a single trail combined from the (possibly empty) partial trail of e contained in G_i and ending at u_i, followed by the edge $\{u_i, w_i\}$, followed by the (possibly empty) partial trail of b contained in G_i starting at w_i. Both G_1 and G_2 are 2-edge connected since they can each be obtained as contractions of G with some self-loops deleted. Furthermore, they each have strictly fewer edges than G, so inductively each has a strong orientation that respects the given trails. Further, we can assume that the orientations are such that the new edges are oriented (u_1, w_1) and (w_2, u_2) by flipping the orientation of all edges in either graph if necessary. We claim that this orientation, together with e oriented as (u_1, u_2) and b oriented as (w_2, w_1), is the required orientation of G. To see this first note that (by our choice of flips) this orientation respects the trails. Secondly, suppose $v_1 \in V_1$ and $v_2 \in V_2$ are arbitrary. Since G_1 is strongly connected $G[V_1]$ contains a directed path from v_1 to u_1. Similarly, $G[V_2]$ contains a directed path from u_2 to v_2. Thus G contains a directed path from v_1 to v_2. A similar argument gives a directed path from v_2 to v_1 and since v_1 and v_2 were arbitrary this proves that G is strongly connected and our induction is complete. □

The construction in the proof can be interpreted as a naive algorithm for finding the required orientation when it exists.

Corollary 3.2. *The one-way trail orientation problem on a graph with n vertices and m edges can be solved in $\mathcal{O}(n + m \cdot f(m, n))$ time, where $f(m, n)$ is the time per operation for fully dynamic bridge finding (a.k.a. 2-edge connectivity).*

At the time of this writing [3], this is $\mathcal{O}(n + m(\log n \log\log n)^2)$. In Section 5 we will provide a less naive algorithm which runs in linear time.

4 Extension to Mixed graphs

Now we will extend our result to the case of mixed graphs. We are going to prove the following.

Theorem 4.1. *Let $G = (V, E)$ be a strongly connected mixed multigraph. Then $G - e$ is strongly connected for all undirected $e \in E$ if and only if for any partition $\mathcal{P}$ of the undirected edges of G into trails, and any $T \in \mathcal{P}$, any orientation of T can be extended to a strong trail orientation of $(G, \mathcal{P})$.*

Suppose $G = (V, E)$ is as in the theorem. We will say that $e \in E$ is *forced* if it is undirected and satisfies that $G - e$ is not strongly connected. This terminology is natural as it is equivalent to saying that there exists a cut (V_1, V_2) in G such that e is the only undirected edge in this cut and such that all the directed edges go from V_1 to V_2. If we want an orientation of the trails making the graph strongly connected we are clearly forced to orient e from V_2 to V_1.

Theorem 4.1 is a proper extension of Theorem 3.1 since if G is undirected and 2-edge connected then no $e \in E$ is forced. Furthermore, the theorem suggests a very simple polynomial time algorithm (see Algorithm 1) for finding a strong orientation of the trails if it exists. Indeed, if the mixed graph contains forced edges we direct the corresponding trails in the forced direction. If there are no

forced edges then either the graph is no longer strongly connected in which case we know that a strong trail orientation doesn't exist. Otherwise, we may by Theorem 4.1 orient any trail in an arbitrary direction.

Algorithm 1: Algorithm for mixed graphs.

Input: A mixed multigraph G and a partition $\mathcal{P}$ of the undirected edges of G into trails.
Output: True if $(G, \mathcal{P})$ has a strong trail orientation, otherwise false. If G has a bridge or is not strongly connected, G is left unmodified. Otherwise G is modified, either to have such a strong trail orientation, or to a forced graph that is not strongly connected.

```
if G has a bridge or is not strongly connected then
    return false
end
while |P| > 0 do
    if for some undirected edge e, G − e is not strongly connected then
        Let T ∈ P be the trail containing e.
        if some orientation of T leaves G strongly connected then
            Apply such an orientation of T to G
        else
            return false
        end
    else
        Let T ∈ P be arbitrary.
        Update G by orienting T in an arbitrary direction.
    end
    Remove T from P.
end
return true
```

For proving Theorem 4.1 we will need the following lemma.

Lemma 4.2. *Let G be a directed graph, and let (A, B) be a cut with exactly one edge crossing from A to B. Then G is strongly connected if and only if G/A and G/B are.*

Proof. Strong connectivity is preserved by contractions, so if G is strongly connected then G/A and G/B both are. For the other direction, let (a_1, b_1) be the edge going from A to B. As G/A is strongly connected and (a_1, b_1) is the only edge going from A to B we can for any edge (b_2, a_2) going from B to A find a path from b_1 to b_2 that stays in B. Since G/B is strongly connected, it follows that A is strongly connected in G. By a symmetric argument, B is also strongly connected in G and since the cut has edges in both directions (as e.g. G/A is strongly connected), G must be strongly connected. □

Now we provide the proof of Theorem 4.1.

Proof of Theorem 4.1. If there exists an undirected edge e such that $G - e$ is not strongly connected, then the trail T containing e can at most be directed one way since e is forced, so there is an

orientation of T that does not extend to a strong trail orientation of $(G, \mathcal{P})$. To prove the converse suppose $G - e$ is strongly connected for all undirected $e \in E$.

The proof is by induction on $|\mathcal{P}|$. If $|\mathcal{P}| = 0$ the result is trivial. So suppose $|\mathcal{P}| \geq 1$ and that the theorem holds for all $(G', \mathcal{P}')$ with $|\mathcal{P}'| < |\mathcal{P}|$.

Consider the chosen trail $T \in \mathcal{P}$. If both orientations of T leave a graph where the condition in the theorem is still satisfied we are home by induction. Otherwise, there must exist a cut (A, B) of the following form: (1) T crosses the cut exactly once, (2) exactly one undirected edge from a different undirected trail $T' \in \mathcal{P}$ crosses the cut and (3) every directed edge crossing the cut goes from A to B.

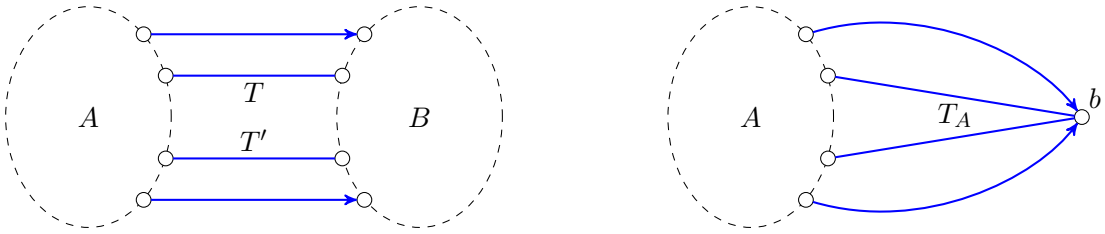


Figure 3: A cut with two undirected edges and all directed edges going from A to B followed by a contraction of B.

Now suppose there is such a cut (A, B) (see Figure 3). Consider the graph G/B and let b be the node corresponding to B in G/B. Let $\mathcal{P}_A$ consist of all trails in $\mathcal{P}$ that are completely contained in A, together with a single trail T_A combined from the (possibly empty) fragments of T and T', joined at b. Since any cut in G/B corresponds to a cut in G, G/B is strongly connected and remains so after deletion of any single undirected edge. By construction $|\mathcal{P}_A| \leq |\mathcal{P}| - 1$, so by induction any orientation of T_A in G/B extends to a strong orientation of $(G/B, \mathcal{P}_A)$. Let G/A, a, $\mathcal{P}_\mathcal{B}$ and T_B be defined symmetrically, then by the same argument any orientation of T_B in G/A extends to a strong orientation of $(G/A, \mathcal{P}_B)$. Now for any orientation of T, we can choose orientations of T_A and T_B that are compatible. The result then follows by Lemma 4.2. □

Theorem 4.1 gives a sufficient condition for the existence of a strong orientation and we deal with the other cases by first orienting all forced edges. However, the generalised Robbins' theorem provides a simple equivalent condition, which we lack. Finding such an equivalent condition in our setting is an essential open problem for strong trail orientations. As seen by the example of Figure 1 such a condition will necessarily have to depend on the structure of the trail partition.

5 Linear time algorithm

In this section we provide our linear time algorithm for solving the trail orientation problem in undirected graphs. For this, we make two crucial observations. First, we show that there is an easy linear time reduction from general graphs or multigraphs to cubic multigraphs. Second, we show that in a cubic multigraph with n vertices, we can in linear time find and delete a set of edges that are at the end of their trails, such that the resulting graph has $\Omega(n)$ 3-edge connected components. We further show that we can compute the required orientation recursively from an orientation of each 3-edge connected component together with the cactus graph of 3-edge connected components.

Since the average size of these components is constant, we can compute the orientations of most of them in constant time individually and thus in linear time taken together. The rest contains at most a constant fraction of the vertices, and so a simple geometric sum argument tells us that the total time is also linear.

We start out by making the following reduction.

Lemma 5.1. *The one-way trail problem on a 2-edge connected graph or multigraph with n vertices and m edges, reduces in $\mathcal{O}(m+n)$ time to the same problem on a 2-edge connected cubic multigraph with $2m$ vertices and $3m$ edges.*

Proof. Cyclically order the edges adjacent to each vertex such that two edges that are adjacent on the same trail are consecutive in the order. Replace each single vertex v with a cycle of length $\deg(v)$, with each vertex of the new cycle inheriting a corresponding neighbour of v such that the order of the vertices on the cycle corresponds to the cyclic ordering (see Figure 4). Note that for a vertex of degree 2, this creates a pair of parallel edges, so the result may be a multigraph. By the choice of cyclic ordering, we can make the cycle-edge between the two vertices on the same trail belong to that trail. The rest of the cycle edges form new length 1 trails. Clearly the new graph is also 2-edge connected so by Theorem 3.1 it has a strong trail orientation, and any strong trail orientation on this graph translates to a strong trail orientation of the original graph. The new graph has exactly $2m$ vertices and $3m$ edges, and is constructed in $\mathcal{O}(m+n)$ time.

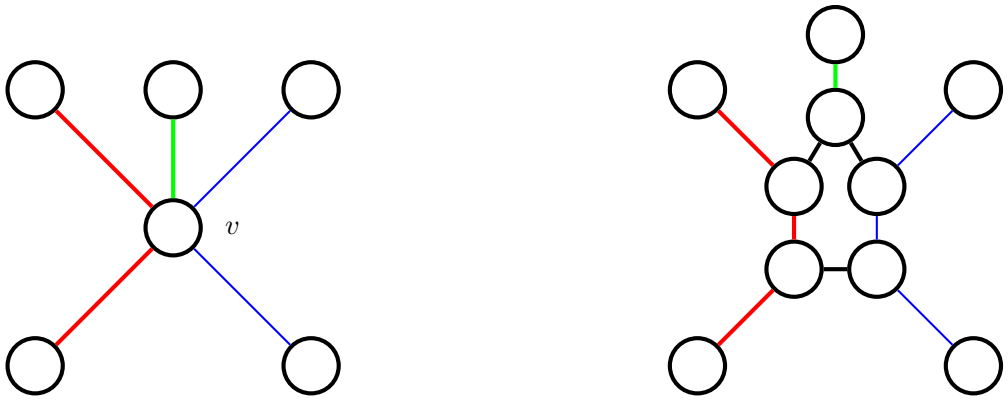


Figure 4: A node of degree 5 turns into a cycle of length 5.

□

Recall now that a multigraph C is called a *cactus* if it is connected and each edge is contained in at most one cycle. If G is any connected graph we let $C_1, \ldots, C_k$ be its 3-edge connected components. It is well known that if we contract each of these we obtain a cactus graph. For a proof of this result see section 2.3.5 of [8]. As the cuts in a contracted graph are also cuts in the original graph we have that if G is 2-edge connected then the cactus graph is 2-edge connected. The edges of the cactus are exactly the edges of G which are part of a 2-edge cut. We will call these edges *2-edge critical*.

It is easy to check that if a cactus has m edges and n vertices then $m \leq 2(n-1)$. We will be using this result in the proof of the following lemma.

Lemma 5.2. *Let $G = (V, E)$ be a cubic 2-edge connected multigraph, let $X \subseteq E$, and let $F \subseteq E$ with $F \supseteq E \setminus X$ minimal (w.r.t inclusion) such that $H = (V, F)$ is 2-edge connected. Then H has at least $\frac{2}{5}|X|$ distinct 3-edge connected components.*

Proof. Let $X_{\text{del}} = X \setminus F$ be the set of edges deleted from G to obtain H, and let $X_{\text{keep}} = X \setminus X_{\text{del}}$ be the remaining edges in X.

By minimality of H there are at least $|X_{\text{keep}}|$ 2-edge-critical edges in H i.e. edges of the corresponding cactus, and thus, if $|X_{\text{keep}}| \geq \frac{4}{5}|X|$, there are at least $\frac{1}{2}|X_{\text{keep}}| + 1 \geq \frac{2}{5}|X| + 1$ distinct 3-edge connected components.

If $|X_{\text{keep}}| \leq \frac{4}{5}|X|$ then $|X_{\text{del}}| \geq \frac{1}{5}|X|$, and since G is cubic and the removal of each edge creates two vertices of degree 2 we must have that H has at least $2|X_{\text{del}}| \geq \frac{2}{5}|X|$ distinct 3-edge connected components. □

Lemma 5.3. *Let $G = (V, E)$ be a connected cubic multigraph with E partitioned into trails. Then G has a spanning tree that contains all edges that are not at the end of their trail.*

Proof. Let F be the set of edges that are not at the end of their trail. Since G is cubic, the graph (V, F) is a collection of vertex-disjoint paths, and in particular it is acyclic. Since G is connected, F can be extended to a spanning tree. □

Note that we can find this spanning tree in linear time e.g. by contracting all edges internal to a trail, finding a spanning tree of the resulting graph, and adding the internal trail edges to the edges of this spanning tree.

Lemma 5.4. *Let $G = (V, E)$ be a cubic 2-edge connected multigraph with E partitioned into trails. Let T be a spanning tree of G containing all edges that are not at the end of their trail. Let H be a minimal subgraph of G (w.r.t inclusion) that contains T and is 2-edge connected. Then for any $k \geq 5$, less than $\frac{4}{5}\frac{k}{k-1}|V|$ of the vertices in H are in a 3-edge connected component with at least k vertices.*

Proof. Let X be the set of edges that are not in T. Since G is cubic, $|X| = \frac{1}{2}|V| + 1$. By Lemma 5.2 H has at least $\frac{2}{5}|X| > \frac{1}{5}|V|$ 3-edge connected components. Each such component contains at least one vertex, so the total number of vertices in components of size at least k is less than $\frac{k}{k-1}\left(|V| - \frac{1}{5}|V|\right) = \frac{4}{5}\frac{k}{k-1}|V|$. □

Definition 5.5. Let C be a 3-edge connected component of some 2-edge connected graph H, whose edges are partitioned into trails. Define $\Gamma_H(C)$ to be the 3-edge connected graph obtained from C by inserting a new edge $\{e_1, f_1\}$ for each min-cut $\{e, f\}$ where $e = \{e_1, e_2\}$ and $f = \{f_1, f_2\}$ and $e_1, f_1 \in C$. Define the corresponding partition of the edges of $\Gamma_H(C)$ into trails by taking every trail that is completely contained in C, together with new trails combined from the fragments of the trails that were broken by the min-cuts together with the new edges that replaced them. See Figure 5.

At this point the idea of the algorithm can be explained. We remove as many of the edges that sit at the end of their trails as possible, while maintaining that the graph is 2-edge connected. Lemma 5.4 guarantees that we obtain a graph H with $\Omega(|V|)$ many 3-edge connected components of size $O(1)$. We solve the problem for each $\Gamma_H(C)$ for every 3-edge connected component. Finally, we combine the solutions for the different components like in the proof of Theorem 3.1.

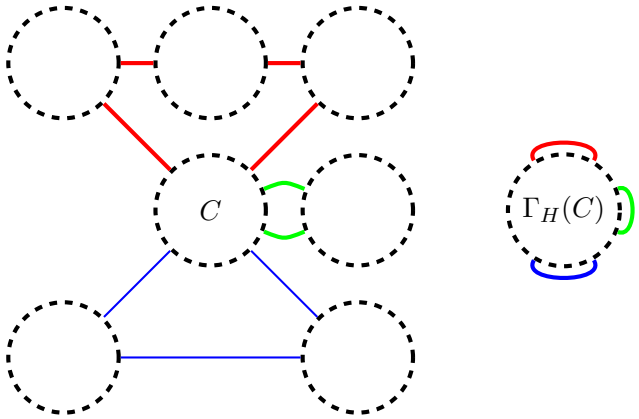


Figure 5: The 3-edge connected components of a 2-edge connected graph. Notice that every edge leaving a 3-edge connected component C becomes part of a cycle if all 3-edge components are contracted. The right hand side shows $\Gamma_H(C)$ where C is the component in the middle.

Theorem 5.6. *The one-way trail orientation problem can be solved in $\mathcal{O}(m+n)$ time on any 2-edge connected undirected graph or multigraph with n vertices and m edges.*

Proof. By Lemma 5.1, we can assume the graph is cubic. For the algorithm we will use two subroutines. First of all, when we have found a minimum spanning tree T containing the edges that are not on the end of their trail we can use the algorithm of Kelsen et al. [5] to, in linear time, find a minimal (w.r.t. inclusion) subgraph H of G that contains T and is 2-edge connected. Secondly, we will use the algorithm by Mehlhorn et al. [7] to, in linear time, build the cactus graph of 3-edge connected components. The algorithm runs as follows:

1. Construct a spanning tree T of G that contains all edges that are not at the end of their trail.
2. Construct a minimal subgraph H of G that contains T and is 2-edge connected[3].
3. Find the cactus of 3-edge connected components of[4] H.
4. For each 3-edge connected component C_i, construct $\Gamma_H(C_i)$.
5. Recursively compute an orientation for each[5] $\Gamma_H(C_i)$.
6. Combine the orientations from each component to a strong trail orientation of H. A such is also a strong trail orientation of G.

First we will show correctness and then we will determine the running time.

Recall that we can flip the orientation in each $\Gamma_H(C_i)$ and still obtain a strongly connected graph respecting the trails in $\Gamma_H(C_i)$. The way we construct the orientation of the edges of G is by flipping the orientation of each $\Gamma_H(C_i)$ in such a way that each cycle in the cactus graph becomes a

[3]See Kelsen [5].
[4]See Mehlhorn [7].
[5]Note that $\Gamma_H(C_i)$ is cubic unless it consists of exactly one node. In this case however we don't need to do anything.

directed cycle[6]. This can be done exactly because no edge of the cactus is contained in two cycles. By construction this orientation respects the trails so we need to argue that it gives a strongly connected graph.

For showing that the resulting graph is strongly connected, consider the graph in which every 3-edge connected component is contracted to a single point. This is exactly the cactus of 3-edge connected component of G which is strongly connected as the cycles of the cactus graph have become directed cycles. Now assume inductively that we have uncontracted some of the 3-edge connected components obtaining a graph G_1 which is strongly connected. We then uncontract another component C (see Figure 6) and obtain a new graph G_2 which we will show is strongly connected. If $u, v \in C$, then since $\Gamma_H(C)$ is strongly connected there is a path from u to v in $\Gamma_H(C)$. If this path only contains edges which are edges of C it will also exist in G_2. If the path uses one of the added (now oriented) edges (e_1, f_1), it is because there are edges (e_1, e_2) and (f_2, f_1) forming a cut and thus being part of a cycle in the cactus. In this case we use edge (e_1, e_2) to leave component C and then go from e_2 back to component C which is possible since G_1 was strongly connected. When we get back to the component C we must arrive at f_1 since otherwise there would be two cycles in the cactus containing the edge (e_1, e_2). Hence we succeeded in disposing of the edge (e_1, f_1) with a directed path in G_2. This argument can be used for any of the edges of $\Gamma_H(C)$ that are not in C and thus C is strongly connected in G_2. Since G_1 was strongly connected this suffices to show that G_2 is strongly connected. By induction this implies that after uncontracting all components the resulting graph is strongly connected.

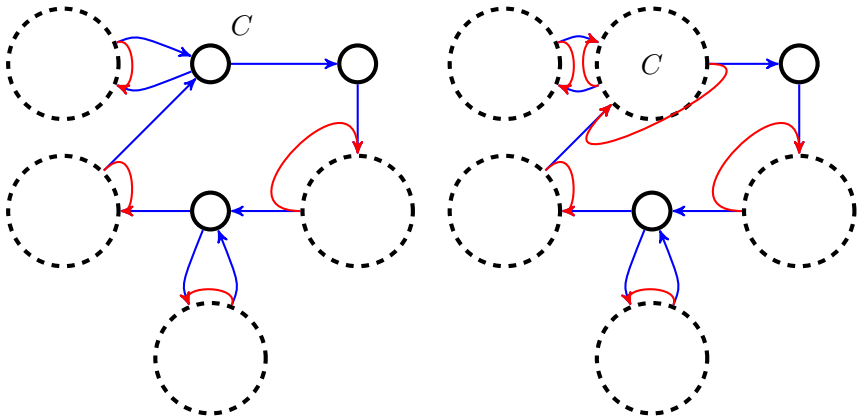


Figure 6: Before and after uncontracting component C. The blue edges are the edges of the cactus i.e. the 2-edge critical edges of H. The red edges are the ones obtained from the 2-edge cuts of H as described in the construction of the $\Gamma_H(C_i)$.

Now for the running time. By Lemma 5.4 each level of recursion reduces the number of vertices in "large" components by a constant fraction, for instance for $k = 10$ we reduce the number of vertices in components of size at least 10 by a factor of $\frac{8}{9}$. Let $f(n)$ be the worst case running time with n nodes for a cubic graph, and pick c large enough such that cn is larger than the time it takes to go through steps 1-4 and 6 as well as computing the orientations in the "small" components. This includes the linear time needed to construct the new set of trails (in 4), and the linear time to

[6]In practice this is done by making a DFS (or any other search tree one likes) of the cactus and repeatedly orienting each component in a way consistent with the previous ones.

reassemble the directed trails (in 6). Let $a_1, \ldots, a_k$ be the number of vertices in the "large" 3-edge connected components. Then $\sum_i a_i \leq \frac{8n}{9}$ and

$$f(n) \leq cn + \sum_i f(a_i).$$

Inductively, we may assume that $f(a_i) \leq 9cn$ and thus obtain

$$f(n) \leq cn + \sum_i f(a_i) \leq cn + \sum_i 9ca_i = cn + 8cn = 9cn$$

proving that $f(n) \leq 9cn$ for all n. □

Algorithm 2: Linear time algorithm for cubic graphs.

Input: A 2-edge connected undirected cubic multigraph G and a partition $\mathcal{P}$ of the edges of G into trails.

Output: G is modified to a strong trail orientation of $(G, \mathcal{P})$.

```
Construct a spanning tree T of G that contains all edges that are not at the end of their trail.
Construct a minimal subgraph H of G that contains T and is 2-edge connected.
Find the cactus C of 3-edge connected components of H.
for each 3-edge connected component C_i in C in DFS preorder do
    Construct G_i = Γ_H(C_i).
    Recursively compute an orientation for G_i.
    if the orientation of G_i is not compatible with its DFS parent then
        Flip orientation of G_i
    end
end
for each edge e deleted from G to create H do
    if no edge on the trail of e has been oriented yet then
        Pick an arbitrary orientation for e.
    else
        Set the orientation of e to follow the trail.
    end
end
```

6 Open problems

We here mention two problems concerning trail orientations which remain open.

First of all, our linear time algorithm for finding trail orientations only works for undirected graphs and it doesn't seem to generalise to the trail orientation problem for mixed graphs. It would be interesting to know whether there also exists a linear time algorithm working for mixed graphs. If so it would complete the picture of how fast an algorithm we can obtain for any variant of the trail orientation problem.

Secondly, our sufficient condition for when it is possible to solve the trail orientation problem for mixed multigraphs is clearly not necessary. It would be interesting to know whether there is a

simple necessary and sufficient condition like there is in the undirected case. Since in the mixed case the answer to the problem actually depends on the given trail decomposition and not just on the structure of the mixed graph it is harder to provide such a condition. One can however give the following condition. It is possible to orient the trails making the resulting graph strongly connected if and only if when we repeatedly direct the forced trails end up with a graph satisfying our condition in Theorem 4.1. This condition is not simple and is not easy to check directly. Is there a more natural condition?

References

[1] Frank Boesch and Ralph Tindell. Robbins's theorem for mixed multigraphs. *The American Mathematical Monthly*, 87(9):716–719, 1980.

[2] Fan R. K. Chung, Michael R. Garey, and Robert E. Tarjan. Strongly connected orientations of mixed multigraphs. *Networks*, 15(4):477–484, 1985.

[3] Jacob Holm, Eva Rotenberg, and Mikkel Thorup. Dynamic bridge-finding in $\tilde{O}(\log 2\ n)$ amortized time. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '18, pages 35–52, 2018.

[4] John Hopcroft and Robert Tarjan. Algorithm 447: Efficient algorithms for graph manipulation. *Commun. ACM*, 16(6):372–378, June 1973.

[5] Pierre Kelsen and Vijaya Ramachandran. On finding minimal 2-connected subgraphs. In *Proceedings of the Second Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms, 28-30 January 1991, San Francisco, California.*, pages 178–187, 1991.

[6] Zoltán Király and Zoltán Szigeti. Simultaneous well-balanced orientations of graphs. *J. Comb. Theory, Ser. B*, 96(5):684–692, 2006.

[7] Kurt Mehlhorn, Adrian Neumann, and Jens M. Schmidt. Certifying 3-edge-connectivity. *Algorithmica*, 77(2):309–335, 2017.

[8] Hiroshi Nagamochi and Toshihide Ibaraki. *Algorithmic Aspects of Graph Connectivity*. Cambridge University Press, New York, NY, USA, 1 edition, 2008.

[9] J. A. Nash-Willams. On orientations, connectivity and odd-vertex-pairings in finite graphs. *Canad. J. Math.*, 12(5):555–567, 1960.

[10] H. E. Robbins. A theorem on graphs, with an application to a problem of traffic control. *The American Mathematical Monthly*, 46(5):281–283, 1939.

0.9 APPENDIX

Danish:

- Amtsavisen.dk
- Aoib.dk
- AOH.dk
- Avisen.dk
- Berlingske.dk
- BT.dk
- Dagbladet-Holstebro-Struer.dk
- Dagbladet Køge
- DagbladetRingSkjern.dk
- Dagbladet Ringsted
- Dagbladet Roskilde
- Dit-Ringsted.dk
- Dit-Soroe.dk
- dknyt.dk
- DR.dk
- Ekstrabladet.dk
- Enyt.dk
- Flensborg Avis
- Folkebladetlemvig.dk
- Folketidende.dk
- Fredericia Dagblad
- Fyens.dk
- Headtopics.com
- Herning Folkeblad
- HSFO.dk
- Information.dk
- Jyllands-Posten.dk

- Kristeligt-Dagblad.dk
- kortenyheder.dk
- LocalEyes.dk
- Lokalavisen.dk
- Lolland-Falsters Folketidende
- MJA.dk
- News.dk
- Nordvestnyt.dk
- Nordvestnyt Holbæk/Odsherred
- Nordvestnyt Kalundborg
- Politiken.dk
- Radio24Syv
- Radioglobus.dk
- Radiovictoria.dk
- Rbot.dk
- Ritzau
- Skive Folkeblad
- SN.dk;
- Stiften.dk
- Tidende.dk
- TV2.dk
- VejleAmtsFolkeblad.dk
- Viborg-folkeblad.dk
- Weekendavisen

Foreign

- Aiois.com
- Acunn.com
- Aksam.com.tr
- Airbeit

- Atmarkit.co.jp
- Atyun.com
- Automotive Technology for Today
- Bolssmania.com
- BXmart.com
- Business Standard
- Chinavoa.com
- Cloud-finder.ch
- Cognilytica
- Cvclavoz.com
- Verdict.co.uk
- Indianexpress.com
- Fossbytes.com
- Datainnovation.org
- De Ingenieur
- Deuxieme.nl
- Devdiscourse.com
- Digitaljournal.com
- Distinguished Cybersecurity
- Drdouggreen.com
- Ecnmag.com
- Edexlive.com
- El Diario Vasco
- EL DISPENSADOR
- Elreporte.com.uy
- Engsincero.com.br
- Emol.com
- Engineering Evil
- Engineering360

- Eurasiaview.com
- EurekAlert
- Europa Press
- Expresscomputer.in
- Financialexpress.com
- Flipboard.com
- Fossbytes.com
- Galileu
- Houseofbots.com
- Hyper.ai
- Ictk.ch
- Indian Strategic Studies
- Indiatoday.in
- Infomance.com
- Innovatorsmag.com
- Innovation Toronto
- Iot.ng
- Jimuenglish.com
- La Nacion
- La Vanguardia
- Lecano.com
- Medkit.info
- Megawhiz.com
- Milliyet.com.tr
- Mimikama.at
- Muckrack.com
- Mundomaistech.com.br
- Nazology.net
- Newslocker

- Neurosciencenews.com
- Newslettercollector.com
- New.qq.com
- News.nicovideo.jp
- Nice.hu
- Njus.me
- Odatv.com
- Parallelstate.com
- Pentoz.com
- Phys.org
- Pickthenews.com
- Popyard.com
- Pressetext.com
- Reddit.com
- Refugo.hol.es
- Sanal Kripto
- Science Daily
- Sözcü
- Science Wiki
- Sciencecodex.com
- Scienmag
- Scientific American
- Scientific India
- Serendeputy
- Sinirsizbilim.com
- Sputnik
- T.cj.sina.com.cn
- Tech Check News
- Techristic.com

- Technologynetworks.com
- Theweek.in
- Techcapon
- Technochoudhary.com
- The Better Parent
- Thelatest.com
- Timesnownews.com
- Tladatatech.com
- Todo Tech 2.0
- Topocaltalk.com
- Urfanatik.com
- Wallstreet-online.de
- Worldpronews.com
- Zeus News
- 20minutos.es
- 15minutenews.com