Doctoral theses at NTNU, 2021:61

Huy Quang Duong

Event Detection in Changing and Evolving Environments

Doctoral thesis

NTNU
Norwegian University of Science and Technology
Thesis for the Degree of
Philosophiae Doctor
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Norwegian University of
Science and Technology

Huy Quang Duong

Event Detection in Changing and Evolving Environments

Thesis for the Degree of Philosophiae Doctor

Trondheim, March 2021

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

NTNU

Norwegian University of
Science and Technology

NTNU
Norwegian University of Science and Technology

Thesis for the Degree of Philosophiae Doctor

Faculty of Information Technology and Electrical Engineering
Department of Computer Science

ISBN 978-82-326-5838-1 (printed ver.)
ISBN 978-82-326-5874-9 (electronic ver.)
ISSN 1503-8181 (printed ver.)
ISSN 2703-8084 (online ver.)

Doctoral theses at NTNU, 2021:61

Printed by NTNU Grafisk senter

Abstract

The availability of modern technology and the recent proliferation of devices and sensors have resulted in a tremendous amount of data being generated, stored and handled in various applications that affect almost all aspects of daily life. The analysis and detection of interesting events from such massive amounts of data, which typically originate from multiple sources and have many different forms and characteristics, are important tasks. A major challenge is that the data generated in this way are inherently dynamic, and their underlying distribution may change and evolve over time. Despite the efficiency of existing state-of-the-art methods, many challenges remain to be solved. One limitation is that most existing methods are designed to work well with certain specific characteristics, or for certain predefined assumptions about the data, for example that they are stable and independent, and have identical underlying distributions. Moreover, although many existing approaches have been shown efficient and effective in practical applications, the proposed solutions often lack formal theoretical foundations. Hence, these results are mostly based on heuristics and empirical observations.

An important aspect, particularly in dynamic environments, is that the data characteristics are normally unknown beforehand, and that such assumptions about data are rarely valid in real life. With this in mind, the main goal of this thesis is to address the aforementioned drawbacks and challenges, by developing novel approaches and techniques for detecting events, while taking into account the different characteristics and the dynamic nature of the underlying distribution of the data. An important contribution of this work is that in addition to demonstrating the efficiency and effectiveness of our methods in practice, via empirical studies and experiments, we also provide principles and theoretical foundations to prove that the efficiency of the proposed methods also holds in formal proofs. Several extensive experiments are carried out to thoroughly evaluate the performance of the proposed methods. In particular, we perform comprehensive evaluations using both synthetic datasets with ground truths and real-world datasets. The experimental results show that our proposed approaches outperform alternative state-of-the-art algorithms on event detection problems, and demonstrate their efficiency, effectiveness, and applicability.

Preface

This thesis is submitted to the Norwegian University of Science and Technology (NTNU) for partial fulfillment of the requirements for the degree of philosophiae doctor.

The doctoral program has been conducted at the Department of Computer Science, NTNU, Trondheim. The main supervisor was Professor Heri Ramampiaro, and Professor Kjetil Nørvåg was co-advisor.

Acknowledgements

First of all, I would like to thank my supervisors, Professor Heri Ramampiaro and Professor Kjetil Nørvåg, for their advice and great support during my PhD program.
I would like to thank colleagues, the IDI administrative and technical staffs for creating a nice working environment and providing all necessary support for working.
Above all, all my gratitude is to my family for their endless love, unconditional encouragement, and support.

"No man ever steps in the same river twice, for it's not the same river and he's not the same man."

- Heraclitus, *535 BC – 475 BC*

"It is not the strongest of the species that survives, nor the most intelligent; it is the one most adaptable to change."

Charles Darwin, *1809 – 1882*

"Everything in the real world is dynamic and changing."

Unknown

Contents

List of Tables

List of Figures

Part I

Overview and Preliminaries

This part introduces the motivation and research context for this PhD work. It also presents the research questions and the methodologies used to conduct this research. An overview of the connection between the contributions of this work and the publications making up this thesis is presented at the end of the first chapter. The second chapter briefly presents the background and preliminaries related to the research topic of this thesis, and discusses related work and alternative state-of-the-art approaches.

Chapter 1

Introduction

In this chapter, we first introduce the motivation and research context for this PhD work (Sections 1.1 and 1.2). We raise several research questions related to the context and motivation, and use these questions to define the problem addressed in this thesis. Next, we present the methodologies used in this research to address these research questions (Section 1.3). Section 1.4 explains the research method used, and Sections 1.5 and 1.6 describe our contributions and the connection between the contributions and publications. Finally, the structure of the thesis is described.

1.1 Motivation

This work seeks to tackle and overcome challenges related to event detection that arise from evolving data (i.e., data that change over time). Data are naturally generated, and are used in real-world applications in every aspect of daily life. They are used in many important domains such as geometry, physics, biology and computer science, to name only a few. In real-world applications, data exist in many different forms, with many different characteristics. For instance, the two main types of data are unstructured and structured data (where semi-structured data can be considered as a form of structured data). Most data in real applications is in unstructured form, such as in text and documents, and these unstructured data are normally preprocessed and transformed into structured data in order to enable easy estimation and maintenance. Structured data offer a standardized format that can provide comprehensive and insightful knowledge of the whole data. There are various types of structured data, for example the transactional data used in retail, attribute-relation data, points of interest (POI), multidimensional array data (tensors), and graphical data, among others.

Detecting and analyzing interesting events (i.e., changes, anomalies, fraud detection) within massive amounts of data in different forms is an important task in the field of data mining. Since data are constantly changing and different types of data may have different characteristics, detecting events in evolving data is a relatively difficult problem.

A significant challenge that arises when working with various types of structured data in dynamic environments is that information on the characteristics of the data is not available. Although efficient methods have been designed to work well with certain specific characteristics, the characteristics of the data are in most cases unknown beforehand, and a robust solution needs to take into account the different characteristics of data in order to avoid bias. This means that an efficient solution must take into account the different characteristics of the data, and needs to automatically adapt to any changes both quickly and accurately.

Another important challenge is that data evolve (e.g., data in the form of a stream), and their underlying distributions may also change independently of each other, under constraints that may also be continuously changing. In addition, the correlations and transitions between features or groups of features extracted from data are complex. Nevertheless, most existing approaches assume that data are: (i) stable; (ii) drawn from the same distribution; and (iii) independent and identically distributed. These assumptions usually do not hold in real, dynamic environments, and the performance of traditional approaches is consequently dramatically reduced when changes occur. Meanwhile, several empirical experiments and studies have shown that there are important dependencies among data points in real-world applications [178, 17]. No sufficient study has investigated detection events in dynamic and fast-evolving data with a focus on the correlations and transitions of the data. This leads to a particular need to design an efficient solution that carefully considers the dependencies among data, in which the corresponding hypotheses for detecting events are capable of capturing insightful and meaningful events for the user.

In the context of high dimensional and streaming data, where data arrive in a stream with high volume and velocity, the design of an efficient solution with limitations on memory and computational infrastructure is also a challenge. More specifically, the task of detecting events, e.g., dense subtensors and subgraphs, is generally hard, and this is typically an NP-complete or NP-hard problem. Thus, instead of using an exact method with high complexity and high resource consumption, approximation approaches are commonly used, and these methods usually have polynomial time complexity that depends on the dimensions and the size of the data. Despite their

efficiency in practical applications, most currently existing approaches lack a formal theoretical foundation that can guarantee the quality of the solution, since the results and efficiency are mostly based on heuristics and empirical observations. An important drawback of methods with formal theoretical foundations is that they can only provide a loose theoretical guarantee for the solutions to the event detection problem. A further limitation of current methods is a lack of generalization, e.g., the number of estimations they can perform at a time. This means that state-of-the-art methods can estimate only one instance of an estimator with minimal guarantees on the quality of the solution. Motivated by this, we aim in this work to address these drawbacks and challenges by proposing novel techniques and methods for detecting events in various types of data in evolving environments. More specifically, we investigate the following:

1. Dynamic memory allocation and data structures that take into account the unknown distribution and characteristics of data, in order to avoid bias between dense and sparse distributions when constructing patterns;

2. Temporal dependencies and hypotheses for detecting changes in evolving data;

3. Simulations of the correlations and transitions of features in order to sketch a stream with concept drift;

4. Generalization and expansion of the problem of multiple event estimation, with a better guarantee of the quality of the solutions; and

5. Provision of a well-founded theoretical solution to prove the efficiency and correctness of our solutions.

1.2 Research Context

This research work forms part of the BigData strategic research project [1] at NTNU, and was supported and funded by the MUSED (MUlti-Source Event Detection) [2] project, one of the strategic projects within the BigData project. MUSED is hosted by the Faculty of Information Technology and Electrical Engineering at NTNU. It focuses on developing the framework and techniques necessary for effective and efficient detection and prediction of events from multiple and possibly heterogeneous streaming data sources.

The proposed approaches need to take into account the fact that data in this context are generally dynamic, and that the underlying distribution

may be constantly changing. The aim of this PhD thesis is to examine and solve challenges related to event detection and prediction in a continuously changing environment involving various types of data. Some of the main challenges faced in relation to event detection problems in big, dynamic, streaming data are as follows:

- The high volume and high velocity of data;
- Limitations on the memory and computation infrastructure;
- Data that constantly change and evolve, with unknown underlying distributions;
- Highly complex and correlated features within high-dimensional data;
- The assumption of independent and identically distributed characteristics, and hypotheses based on this assumption;
- A lack of theoretical foundations for guarantees of the quality, effectiveness, and correctness of these methods, from both a practical and a theoretical perspective.

1.3 Research Questions

The goal of this research study is to design and build a new, efficient and effective framework for mining (event detection) in various types of data such as transactional data, attribute-relation data, points of interest, tensors, and graph data. The main research problem is to fully understand and utilize the *characteristics* (e.g., density, distribution, dependence) of various types of *dynamic* data (transactions, points of interest, graphs, tensors) to *model* user behaviors for *event detection* problems such as patterns, changes, fraud and anomaly detection. Figure 1.1 illustrates the relations between data structure, behavior modeling and events in an event detection task, and studying the relationships between these three aspects is the main task of this thesis. This work also aims to address the problems with the current lack of insight, understanding and theoretical analysis of state-of-the-art methods.

The research objective is to develop efficient algorithms for:

- Data structure analysis and change detection hypotheses;
- User behavior modeling; and

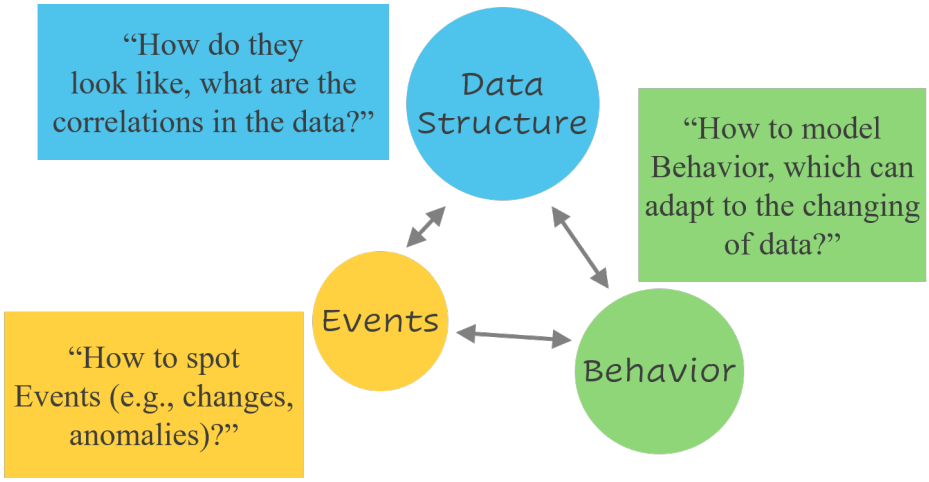


Figure 1.1: Relationships between data structure, behavior modeling and events in the event detection problem.

- Event detection in high-dimensional and complex data.

The primary objective of the work is to propose methods that not only give good performance in terms of runtime and memory consumption, but also offer high levels of accuracy and high scalability. In addition to demonstrating the efficiency of these methods in practice through empirical experiments, a further important objective is to provide principles and formal theoretical foundations for the proposed methods, in order to prove their efficiency. In order to achieve the main goal of this research, a set of research questions and corresponding sub-tasks can be established as follows:

RQ1. Selection of data characteristics and features:

- On which characteristics of the data do the efficiency and accuracy of a data stream mining algorithm largely depend?
- How can we avoid bias when constructing the data characteristics? How can we design efficient methods to work well with a variety of different characteristics of data, and specifically with streaming data in a high-dimensional space?

RQ2. Change hypotheses:

- How can we identify hypotheses for changes in the underlying distribution of data?

- How can we handle the evolving and drifting characteristics of streaming data?
- What is the impact of the dependencies of data features on modeling data features?

RQ3. Simple and robust models:

- What is the most effective data structure that can be employed?
- How can we design models that easily and quickly adapt to the evolution of data?
- What are the mechanisms for the tuning process of these learning models?

RQ4. Efficiency and effectiveness of methods:

- How can we develop accurate, fast and adaptable models?
- How can we achieve a low computational complexity with a rapid response to changes?
- How can we automatically set the coefficients of these models?

RQ5. Correlations between data features:

- Are there any correlations/transitions between data points or the features of data in the data space?
- Are these correlations impacted by or involved in the evolving or drifting of features?

RQ6. Theoretical guarantee foundation:

- What are the limitations of the current methodologies and methods for event detection in terms of theoretical guarantee foundation?
- Are there any other fundamental theories and guarantees that can provide a more theoretical guarantee for the current state-of-the-art algorithms in the context of event detection?

RQ7. Generalization:

- Can we overcome these limitations on guarantees of current solutions?

- Is it possible to generalize the problem?
- Can we provide concrete proofs to guarantee better solutions?

In order to answer these research questions, we explore and address these challenges by proposing novel techniques and methods for detecting events in various types of data in evolving environments. The following areas are studied:

1. A dynamic memory allocation mechanism and a linear joining method for avoiding bias when constructing patterns with unknown distributions and characteristics of dense and sparse data;

2. Hypotheses for detecting changes in evolving data and the impact of data dependencies on the detection of changes and concept drift;

3. Simulations of the correlations and transitions of features in relation to change detection and sketching a stream with concept drift;

4. Generalization of the problem of multiple event estimation with a better quality guarantee for the solutions, and expansion of this problem to both tensor and graph data;

5. A well-founded theoretical solution that can guarantee the efficiency and correctness of these solutions.

1.4 Research Method

In order to successfully carry out this research work, the following iterative ideas were applied via a sequence of steps.

1. Defining and articulating a research question: The objective of this work is specified by determining two targets: (i) providing a novel and better solution to existing research problems; and (ii) solving a novel practical research problem.

2. The topics and challenges that will be pursued in more detail are thoroughly described.

3. For each working problem, we carry out a detailed examination of the background and literature for the problem.

4. The state-of-the-art algorithms are identified, with their advantages and the limitations and drawbacks that could be addressed and improved.

5. For each working problem, new theoretical principles and hypotheses are proposed to demonstrate the correctness of the proposed methods.

6. An empirical study is carried out to propose a novel approach and develop a new solution to solve the problem or improve existing solutions.

7. The algorithms are implemented and extensive experiments performed to allow us to compare the proposed method with the state-of-the-art alternatives in the literature. We analyze and evaluate this comparison and present an in-depth discussion to demonstrate the strong and weak points of each method.

1.5 Contributions

The contributions of this thesis include novel techniques, approaches and improvements to current state-of-the-art methods in order to address issues related to event detection problems. In particular, our contributions are a combination of both practical research and a novel theoretical foundation. Novel and efficient algorithms are designed using the research methodology and the novel proposed theoretical foundation. Extensive experiments are also performed to evaluate the performance of these methods. Furthermore, we discuss and analyze the results to show that our proposed methods can address the issues raised in our research questions. The following paragraphs present a brief overview of our contributions with respect to the corresponding research questions of this study. The main contributions of this thesis can be summarized as follows:

I. Detection in Streaming Data

C1. We propose a summary utility-list structure to reduce the memory consumption and speed up the join operation for constructing the information of patterns in the detection of high profit patterns. This structure is integrated into a novel algorithm for efficiently discovering high utility patterns. The proposed method efficiently stores and retrieves these utility-lists by dynamically allocating memory, and reuses memory during the mining process. The memory for the utility-list is organized as a stream, and is dynamically estimated. Moreover, we introduce a linear time method for constructing the utility-list segments in a utility-list buffer. These contributions relate to research questions RQ1 to RQ3, and are discussed in Chapter 3 of this thesis.

C2. We introduce an algorithm to detect changes in the utility distributions of itemsets in a stream of quantitative customer transactions. The proposed algorithm utilizes a fading function to quickly adapt to changes in a data stream. We propose an approach that uses statistical testing based on Hoeffding's inequality with Bonferroni correction to report significant changes to the user. The proposed method is capable of discovering both increasing and decreasing trends. In addition, we propose a new distance measure that generalizes the cosine similarity by using the distances between pairs of high utility itemsets (HUIs) to detect changes in the structure of HUIs. These contributions relate to research questions RQ1, RQ2 and RQ5, and are discussed in Chapter 4.

C3. We propose a new and efficient method of detecting changes in streaming data by exploring the temporal dependencies of data in the stream. We introduce a new statistical model to compute the probabilities of finding change points in the stream, using the temporal dependency information or factors between different observed data points in a stream. The computed probabilities are used to generate a distribution, which is then used in statistical hypothesis tests to determine the candidate changes. We also use the proposed model to develop a new algorithm for detecting change points in linear time, which can therefore be used in real-time applications. These contributions relate to research questions RQ2 to RQ5, and are discussed in Chapter 5.

II. Feature Correlations with Concept Drift

C4. We propose a novel robust method for sketching streaming histograms with limited computing resources, based on an ensemble randomization method. We develop an algorithm that uses an evolving model with adaptive coefficients to obtain the histogram elements. Our algorithm considers the timestamps of different observations in each coefficient, and solves an optimization problem to evolve the values of the coefficients to optimal values. We use Adaptive Relaxed Alternating Direction Method of Multipliers (ARADMM) [166] as a solver for this optimization problem. These contributions relate to research questions RQ1 and RQ3 to RQ5, and are discussed in Chapter 6.

III. Dense Subregion Detection

C5. We propose novel concrete proofs for the problems of detecting dense subtensors in a tensor data, and dense subgraphs from a graph. We introduce a better theoretical density guarantee for both dense subtensor

and dense subgraph detection for approximation algorithms. The new boundary is better than in state-of-the-art algorithms, and is not only dependent on the dimension of data space, but our novel density guarantee is also constrained to the size of the densest subtensor/subgraph. These contributions answer research questions RQ1, RQ3, RQ6 and RQ7, and are discussed in Sections 7.4 and 7.5.

C6. We propose a new technique that can improve the task of dense subtensor detection. The contributions include both theoretical and practical solutions. We develop concrete theoretical proofs for the estimation of dense subtensors in a tensor problem, provide a guarantee for a higher lower bound density for the estimated subtensors, and develop a new theoretical foundation to guarantee a high density of multiple subtensors. Furthermore, we develop a new algorithm that is not only less complex, but also guarantees the correctness of its effectiveness. This method is applicable to the detection of dense subtensors, and is more efficient than existing methods. These contributions relate to research questions RQ1, RQ3, RQ6 and RQ7, and are discussed in Chapter 7.

1.6 Publications

The contributions described above have been published in several highly regarded journals and international conferences. This thesis is based on the content of the following publications:

P1. Quang-Huy Duong, Philippe Fournier-Viger, Heri Ramampiaro, Kjetil Nørvåg, and Thu-Lan Dam, *Efficient High Utility Itemset Mining using Buffered Utility-lists.* In Applied Intelligence, Springer, 48(7):1859-1877, 2018.
Description: This publication introduces an efficient method for mining high utility patterns [44]. It proposes a novel structure, and summarizes the memory in the form of a stream. The algorithm accesses this summary, and dynamically allocates memory on request to avoid performing costly joins. The content of this publication is discussed in Chapter 3.

P2. Quang-Huy Duong, Heri Ramampiaro, Kjetil Nørvåg, Philippe Fournier-Viger, and Thu-Lan Dam, *High Utility Drift Detection in Quantitative Data Streams.* In Knowledge-Based Systems, Elsevier, 157:34-51, 2018.

Description: This study proposes the use of two types of changes (local and global) in change detection within a quantitative data stream [50]. It introduces an efficient algorithm to detect changes of a single item or changes in the structure of high utility patterns in a stream of data. The content of this publication is discussed in Chapter 4.

P3. Quang-Huy Duong, Heri Ramampiaro, and Kjetil Nørvåg, *Applying Temporal Dependence to Detect Changes in Streaming Data.* In Applied Intelligence, Springer, 48(12):4805-4823, 2018.
Description: In real-world applications, the assumption of identical and independent distributions of data is not reasonable in many contexts. This work studies and investigates the temporal dependence of data in the context of change detection in streaming data [46]. The dependence of the data is based on a projection of the last k data points onto a ℓ_1 ball constraints. The content of this publication is presented in Chapter 5.

P4. Quang-Huy Duong, Heri Ramampiaro, and Kjetil Nørvåg, *Sketching Streaming Histogram Elements using Multiple Weighted Factors.* In Proceedings of the 28th ACM International Conference on Information and Knowledge Management, CIKM, pages 19–28, 2019.
Description: This publication proposes an evolving model to study the correlation of data features in an evolving streaming data [47]. It simulates the transitions of features by optimizing a proposed objective cost. The proposed method sketches the stream in a compact presentation, quickly adapts to concept drift, and preserves the similarity of data with high accuracy. The content of this publication is described in Chapter 6.

P5. Quang-Huy Duong, Heri Ramampiaro, and Kjetil Nørvåg, *Density Guarantee on Finding Multiple Subgraphs and SubTensors.* In ACM Transactions on Knowledge Discovery from Data, pages 1-32, 2021.
Description: This paper proposes novel concrete proofs for the detection of dense subtensor and dense subgraph problems. In this publication, we introduce a better theoretical density guarantee for both dense subtensor and dense subgraph detection, for greedy approximation-based algorithms. The content of this publication is discussed in Sections 7.4 and 7.5.

P6. Quang-Huy Duong, Heri Ramampiaro, and Kjetil Nørvåg, *Multiple Dense Subtensor Estimation with High Density Guarantee.* In Proceedings of the 36th IEEE International Conference on Data Engi-

neering, ICDE, pages 637–648, 2020.
Description: This work provides a proof of a higher lower bound density for the dense estimated subtensors [48]. It also introduces a novel theoretical foundation to prove that there are multiple dense subtensors with a density that is guaranteed to be higher than a lower bound. In this publication, we propose a novel technique for estimating several dense subtensors with a guarantee of the density. The content of this publication is presented in Chapter 7.

Additional Publications: In addition to the above publications, the author also contributed to the following publications:

P7. Thu-Lan Dam, Heri Ramampiaro, Kjetil Nørvåg, and Quang-Huy Duong, *Towards Efficiently Mining Closed High Utility Itemsets from Incremental Databases.*
In Knowledge-Based Systems, 165:13–29, 2019.

P8. Philippe Fournier-Viger, Peng Yang, Jerry Chun-Wei Lin, Quang-Huy Duong, Thu-Lan Dam, Jaroslav Frnda, Lukas Sevcik, and Miroslav Voznak, *Discovering Periodic Itemsets using Novel Periodicity Measures.*
In Advances in Electrical and Electronic Engineering, 17(1):33–44, 2019.

P9. Thu-Lan Dam, Sean Chester, Kjetil Nørvåg, and Quang-Huy Duong, *Efficient Top-k Recently-Frequent Term Querying over Spatio-Temporal Textual Streams.*
In Information Systems, 97:101687, 2021.

The contents of the above publications are only partly relevant to this thesis [39, 58, 35], and were therefore not included in this work. Table 1.1 shows the details of the relationships between each of these publications and theirs contributions in terms of addressing the research questions.

1.7 Thesis Structure

This thesis is organized into five main parts. Part I introduces the motivation, research context, research questions, and related work on state-of-the-art methods. In Parts II-IV, we present our proposed approaches for addressing the challenges of simulating the correlations and transitions of data features with unknown underlying distributions, and discuss the results

Table 1.1: Links between contributions and publications.

	P1	P2	P3	P4	P5	P6
RQ1	•	•		•	•	•
RQ2		•	•			
RQ3	•		•	•	•	•
RQ4			•	•		
RQ5		•	•	•		
RQ6					•	•
RQ7					•	•

of the proposed solutions in terms of event detection in changing environments. Part V summarizes the work in thesis, and discusses possible future research directions. In more detail, the thesis is organized as follows:

Part I Overview and Preliminaries. This part includes this chapter, and contains the background and preliminaries related to the research topic of the thesis.

Chapter 1 introduces the motivation and research context of this PhD work. It also includes the research questions and the methodologies used to conduct this research. An overview of the connections between the contributions of this work and the publications making up this thesis is presented at the end of this chapter.

Chapter 2 briefly presents the background and preliminaries to the research topic in this thesis. Related work and other state-of-the-art approaches are also discussed in this chapter.

Part II Pattern Discovery and Change Detection in Evolving Data. This part focuses on investigating the dependencies of data points and statistical hypotheses to detect changes.

Chapter 3 introduces an approach to solving the problem of detecting patterns with an unknown distribution.

Chapter 4 describes an efficient algorithm for detecting both local and global changes in the structure of high utility patterns.

Chapter 5 presents a method of utilizing the temporal dependencies of data to detect changes in streaming data.

Part III Feature Correlations with Concept Drift. In this part, we focus on investigating the correlations and transitions of data in an evolving and changing context, using multiple factors.

Chapter 6 explains the problem of sketching streaming data. It presents an evolving model that uses multiple weighted factors to simulate the correlations between data features when estimating streaming data with concept drift.

Part IV Dense Subregion Detection. This part emphasizes complex data structures such as tensors and graphs. We introduce both a theoretical foundation and practical work to generalize the dense detection problem and guarantee solutions.

Chapter 7 introduces a better theoretical density guarantee for both dense subtensor and dense subgraph detection with greedy approximation-based algorithms. It provides proofs for the higher lower bound density of the estimated subtensors and subgraphs, and introduces a novel theoretical foundation to generalize the problem of multiple dense subtensor detection with density guarantee.

Part V Conclusions. This part summarizes the contributions of the thesis and outlines possible topics for future work.

Chapter 8 presents a summary and conclusion for the work in this thesis and its contributions. It also discusses and identifies potential research directions for future research.

Chapter 2

Background and Related Work

In this chapter, we briefly review the background to the area of data mining in general [76], and then specifically focus on related works and state-of-the art methods in a particular sub-field of data mining research, the issue of of event detection problems in a dynamic environment with changing and evolving data.

2.1 Data Mining

Data mining is a step in knowledge discovery from data (KDD), a process for extracting useful knowledge from data. Given a set of data, mining aims to find meaningful, interesting information and knowledge from these data, and provide the discovery to the user. The availability of modern technology and the proliferation of mobile devices and sensors have resulted in a tremendous amount of data being generated and stored by applications, relating to all aspects of everyday life. As a result of this explosive growth in data volumes, analysis of massive data is emerging as an important task. An understanding of the vast amounts of data and relevant insightful knowledge can support important decision making. In turn, this helps in improving existing approaches, allowing them to quickly adapt to the natural dynamics of data, and in designing more effective and efficient methods for solving more complex mining problems.

Nowadays, data mining is one of the major fields that has evolved from the area of computer science and information technology, and in-depth data analysis has therefore naturally become an important aspect of the field of data mining in the current information era, in which huge amounts of data are maintained in storage devices. For example, to illustrate the need for in-depth, insightful data analysis, we can use the example of the behavior of

tourists in relation to POI check-ins in several cities at the same time. There are several underlying factors that can affect user check-ins. In addition to the types and places of the POIs, changes in the weather conditions, seasons and times of day can influence changes in user behavior. For instance, a sightseeing location may normally get more visitors during summer than in winter, but this could also be affected by (possibly sudden) changes in the weather conditions. It therefore becomes obvious that analyzing check-in behaviors without considering both the changes in the dynamic properties and time-dependent changes would be insufficient to reflect the overall picture.

In general, advanced data mining needs to consider data from several different perspectives and views, on different characteristics, in order to be capable of providing the user with advanced functionalities such as data classification, clustering, event (outlier/anomaly/fraud) detection, and detection of changes over time in the characteristics of the data.

A Step in the Process of Knowledge Discovery

The KDD process is an infinite cycle of iterative steps, and is illustrated in Figure 2.1. This process includes three basic components: pre-processing, mining, and post-processing.

1. **Pre-processing component**: The functionality of this block is to prepare the most suitable data for the mining processing block. This component contains four sequential steps, as follows:

   (a) *Data cleaning*: The purpose of this step is to clean the original data, i.e., by removing noise and outliers.

   (b) *Integration*: The data input to the process may be from multiple heterogeneous database sources with different formats or attributes. This step is therefore necessary to combine the multiple source data into a consistent, consolidated format.

   (c) *Selection*: This step involves selection of the relevant data and features for processing. Actions involved in this step include feature selection, reducing the dimensions of the data (intrinsic), projection onto a simpler space, or sampling to obtain a representative subset of the data.

   (d) *Transformation*: In this step, data are aggregated by performing a standard or other specific transformation, e.g., summarization, normalization, randomization, and standardization methods.

2. **Mining component**. After the pre-processing stage, the data are in a good suitable form for mining. At this stage, we need to design efficient, effective methods and utilize advanced knowledge to extract interesting patterns from data. This step is typically carried out with support from domain experts with significant, insightful knowledge of the field. The result of this step is then used in the evaluation steps in the next block.

3. **Post-processing component**. This involves the following steps:

   (a) *Evaluation*: This step evaluates the patterns found in the previous block using predefined measures. Extensive and thorough evaluations are conducted using numerous factors and measures to confirm the true meaning and the correctness of the results.

   (b) *Visualization*: The aim of this step is to simulate the interesting patterns. The results are visualized and presented to users in an understandable, readable form.

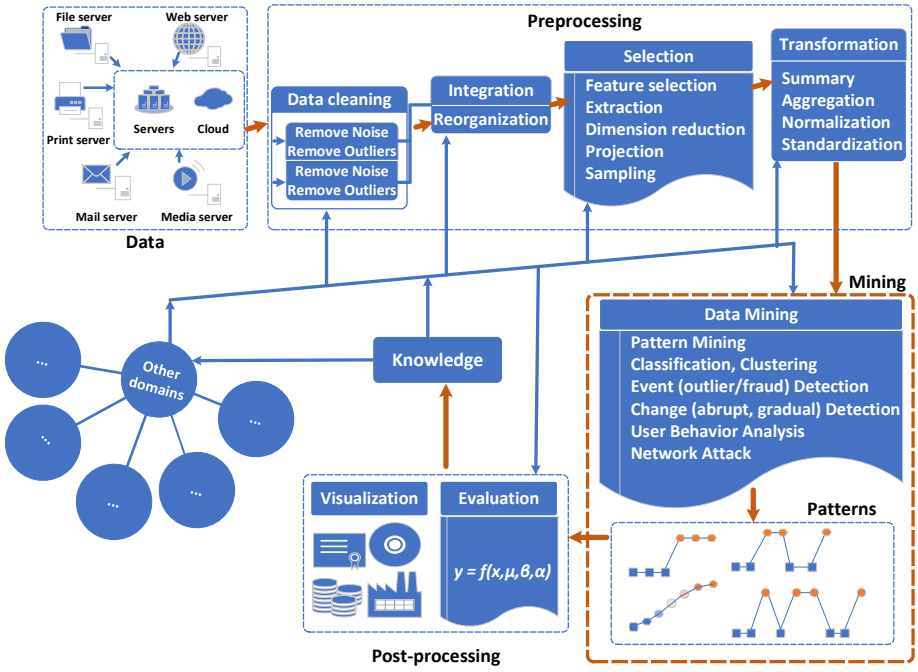


Figure 2.1: Overview of the process of knowledge discovery from data.

The meaningful knowledge extracted via the KDD process is in turn sent back to the whole process as a new knowledge base to improve the performance of the discovery process. A new cycle is then continuously processed, and a new sequence of steps is used to expand the knowledge base.

Data Mining Techniques

Data can now be stored in many different types of databases and storage devices. The massive amounts of data generated are too large for the memory capacity and infrastructure of any isolated system. The generated data are drawn from various heterogeneous sources, and analyzing these data therefore plays a significant role in real-world applications. The effective and efficient analysis of data in such different forms requires the integration and combination of many different domains, such as information retrieval (IR), statistics, data mining, computing and so on. An important reason for the integration of other domains is the need to solve the complex problems that arise regarding the changing characteristics of data from different data sources, which is a challenging task. One advantage of data mining techniques is their capacity to be applied to any type of data. The most common form for the data that are used in many useful daily applications is a transactional database, which stores information about user transactions. Numerous data mining techniques have been proposed to solve problems related to transactional databases, such as the mining of frequent/closed itemsets, HUIs, association rules, and sequential patterns, to name only a few. In addition to transactional data, there are many other flexible, complex types of data structure, such as sequence, spatial, text, multimedia, tensor, graph and networked data. Traditional or advanced mining techniques can also be used to solve problems using these types of data.

However, despite the flexibility of these data techniques when applied to many types of data, challenges arise with respect to data preprocessing, the complexity of integrating highly correlated and transitional features of data from multiple sources, and the natural complexity of mining methods for high-dimensional and changing data. The mining of such widely differing, dynamic data is an advanced and interdisciplinary topic. The methods involved in a data mining problem are drawn from many domains, allowing them to address the complex issues that emerge. A flow diagram for a data mining task starts with a request from a user; based on this request, domain experts decide how to address the problem, and identify the techniques involved in solving the problem. We can use the aforementioned example of an analysis of user activity in social networks to illustrate this. In this application, given a stream of data consisting of user check-in activities

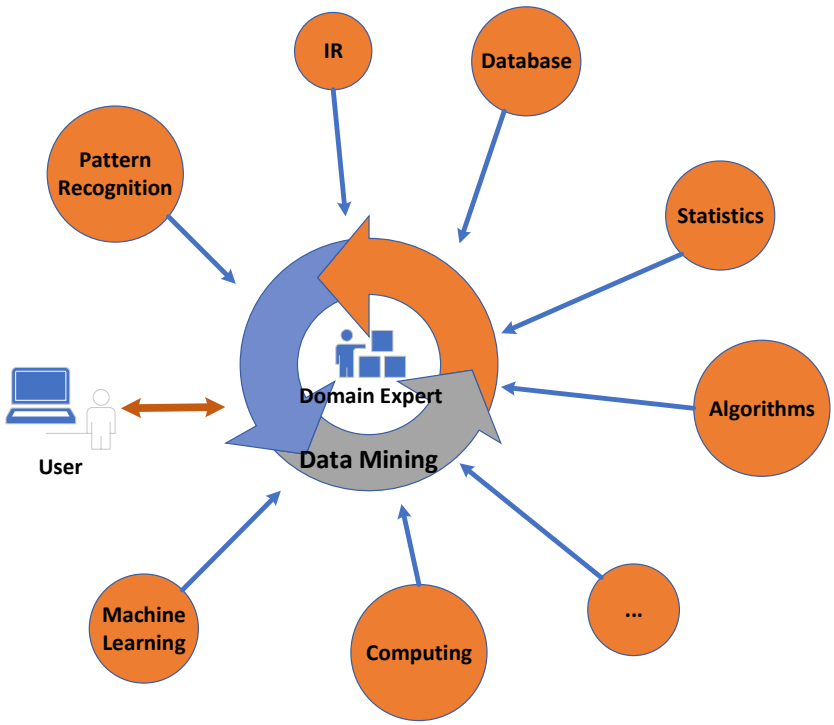


Figure 2.2: Examples of techniques used in data mining.

at interesting places, how can we identify at the current moment, what is the type of a location, does the place change its label, its features or not? To answer this question, we need to monitor the changes (i.e., labels and features of POIs) by analyzing the activity and behaviors of users. This kind of task is helpful in many applications such as recommendation systems, which recommend places of the same type to a user based on check-in activities, and in many other applications, such as classification and clustering.

Database techniques are helpful in handling the huge amounts of data that arrive in a stream with high volume and high velocity. From another perspective, the storing and processing of high-volume and high-velocity streaming data are often infeasible, due to the limitations on memory and computational infrastructure, and approximation (sketching or sampling) approaches are therefore commonly used to estimate the whole set of data in a compact representation. The quality of the sampling process is guaranteed by algorithmic techniques derived from the domains of probability, statistics,

and algorithms. On the other hand, in order to adapt to the natural changes in the data, a robust model needs to automatically turn its coefficients to optimal values by minimizing a particular measure, for instance the error rate. In view of this, data are divided into two subsets: a training set and a testing set. Techniques that involve training the model on a subset of data by solving an optimization problem, and then utilizing the model on the rest of the data (testing), are used in the domains of artificial intelligence, machine learning, and optimization. Computing approaches are often used to improve the solution by parallel processing over distributed resources to gain better performance.

In a nutshell, several techniques from other domains are involved in solving problems in data mining. In particular, a set of mining techniques is a combination of approaches from many domains, as shown in Figure 2.2. These domains may include:

1. *Databases*: Data storage techniques, e.g., data queries, handling of large, distributed data.
2. *Information Retrieval*: Extraction of structured information and knowledge from an unstructured data source in the form of text and multimedia data (documents).
3. *Statistics*: Provides hypotheses for methodologies.
4. *Algorithms*: Algorithmic approaches and measurements of their effectiveness, scalability, and complexity.
5. *Computing*: High-performance infrastructures, including parallel processing and GPUs.
6. *Other domains*: Training and testing the models by optimizing an objective cost, and then making intelligent decisions in order to recognize complex patterns using a learning process based on the data, e.g.
   (a) *Machine Learning*
   (b) *Pattern Recognition*
   (c) *Optimization*

Other Issues and Challenges in Data Mining

In addition to the above-mentioned challenges arising from the dynamic changes in data and the diversity of database types, a mining technique

also needs to consider issues relating to uncertainty, noise, and the incompleteness of data. Data mining methods can effectively extract information from huge amounts of data by carefully applying integrated methods from other disciplines and domains. The performance of mining methods needs also to be considered in terms of both their efficiency and their scalability. A further issue in data mining research is the interaction between the mining system and the users (domain experts), who play an important role in the data mining process. Current challenges in data mining include the problems of how to utilize a knowledge base in a specific data mining context, how to design efficient methods for the discovery of new knowledge that work well and are easy to implement, how to visualize these new discoveries in an easily understandable form, and how to interact with both domain and non-domain experts.

The work in this thesis is motivated by (*i*) *the diversity in the types of database in useful real-world data mining applications*; (*ii*) *the challenges arising from the continuously changing and fluctuating characteristics of data in terms of both internal and external aspects*; (*iii*) *the rapidly growing and changing nature of this topic.* In this study, research is carried out to address the challenges relating to the problem of event detection in data mining from various types of evolving data.

2.2 Pattern Mining

In the following subsections, we introduce some background, related works in this area, and preliminaries to the problem of event detection (e.g., changes) in dynamic, changing data.

High Utility Pattern Mining

The discovery of HUIs in customer transaction databases is a key task when studying the behavior of customers. This consists of finding groups of items that are bought together that yield a high profit. Consider a set of items $I = \{i_1, i_2, \ldots, i_m\}$ representing products sold in a retail store. For each item $i_j \in I$, the external utility of i_j is a positive number representing its unit profit (or more generally, its relative importance to the user). A transaction database D is a set of transactions denoted as $D = \{T_1, T_2, \ldots, T_n\}$, where for each transaction $T_d \in D$, the relationship $T_d \in I$ holds. For each transaction $T_d \in D$, d is a unique integer that is said to be the TID (transaction identifier) of T_d. The internal utility of an item i_j in a transaction T_d is denoted as $q(i, T_d)$. This is a positive number representing the purchase

quantity of the item i_j in T_d. A set of items $X = \{i_1, i_2, \ldots, i_l\} \subseteq I$ containing l items is said to be an itemset of length l, or alternatively, an l-itemset. In the remainder of this thesis, the notation xy will be used to indicate an itemset obtained by concatenating two items x and y. The notation XY will be used to refer to the union of two itemsets X and Y, i.e., $X \cup Y$.

Example 1. *Consider the transaction database depicted in Table 2.1, which comprises five transactions denoted as T_1, T_2, T_3, T_4, and T_5. This database will be used as a running example in this thesis. It contains seven items denoted by the letters a to g, that is, $I = \{$a, b, c, d, e, f, g$\}$. Table 2.2 shows the external utility of each item (e.g., the unit profit). The external utilities of items a, b, c, d, e, f, and g are 5, 2, 1, 2, 3, 1, and 1, respectively. Itemset bc is a 2-itemset appearing in transactions T_3, T_4, and T_5. In transaction T_4, items b, c, d, and e have internal utilities (purchase quantities) of 4, 3, 3, and 1, respectively.*

Table 2.1: A transaction database

TID	Transaction	Transaction Utility
T_1	(a,1), (c,1), (d,1)	8
T_2	(a,2), (c,6), (e,2), (g,5)	27
T_3	(a,1), (b,2), (c,1), (d,6),(e,1),(f,5)	30
T_4	(b,4), (c,3), (d,3), (e,1)	20
T_5	(b,2), (c,2), (e,1), (g,2)	11

Table 2.2: External utility values of items in the transaction database

Item	a	b	c	d	e	f	g
External utility	5	2	1	2	3	1	1

Given a minimum utility threshold value *minutil* and a database D, the problem of HUI mining involves enumerating all HUIs appearing in D, which has a utility no less than *minutil*. The utility of an itemset X in D is defined as:

$$u(X) = \sum_{X \subseteq T_d \wedge T_d \in D} u(X, T_d), \tag{2.1}$$

where $u(X, T_d)$ is the utility of X in T_d. This is a positive number defined as:

$$u(X, T_d) = \sum_{i \in X} u(i, T_d) = q(i, T_d) \times p(i) \tag{2.2}$$

In high utility itemset mining (HUIM), the utility measure is used to assess how important (e.g., how profitable) a pattern is rather than solely frequent [6, 75, 54, 36]. Finding profitable patterns is more useful for businesses than finding frequent patterns. Many algorithms have been designed to discover HUIs [57, 56, 45, 55, 37, 38]. To reduce the search space and mine HUIs efficiently, these algorithms have included various methods to overestimate the utility of itemsets. The solution to this issue, adopted by most HUIM algorithms, has been to rely on upper-bounds on the utility of itemsets that are anti-monotonic to prune the search space without missing any high utility itemsets. Several high utility itemset mining algorithms discover high utility itemsets using the TWU measure and a two-phase approach.

Transaction-Weighted Utilization

Consider an itemset X and a database D. The transaction-weighted utilization (TWU) [112] of X in D is denoted as $TWU(X)$, and is defined as:

$$TWU(X) = \sum_{T_d \in D \wedge X \subseteq T_d} TU(T_d), \tag{2.3}$$

where $TU(T_d) = u(T_d, T_d)$.

The Transaction-Weighted Utilization is the first introduced upper-bounds measure of the utility of itemsets, which was introduced in the Two-Phase algorithm [112], such that it is anti-monotonic to prune the search space without missing any high utility itemsets. Because the TWU is anti-monotonic, and can therefore be used to reduce the search space, while ensuring that no HUIs are missed. The important properties of the TWU measure are described below.

Property 1 (Overestimation [112])**.** *The utility of an itemset X is less than or equal to its TWU, that is, $TWU(X) \geq u(X)$.*

For example, consider the transactions T_1, T_2, and T_3 in the database of the running example. Their TWU values are 8, 27, and 30, respectively. The TWU of the item a is calculated as $TWU(a) = TU(T_1) + TU(T_2) + TU(T_3)$ $= 8 + 27 + 30 = 65$.

Property 2 (Search space reduction using the TWU [112])**.** *For an itemset X, if $TWU(X) < minutil$, it follows that X and its supersets are low utility itemsets.*

This property means that if an itemset has a TWU lower than the *minutil* threshold, all its supersets can be ignored. Although this property has been used by several HUIM algorithms to reduce the search space, one problem that remains is that the TWU is a loose upper bound on the utility of the itemsets. For this reason, many itemsets still need to be considered by algorithms that rely on the TWU measure to extract the set of HUIs. This can result in long execution times and high memory usage.

Two-Phase Approach

The two-phase approach is a method that discovers HUIs using a process with two stages. In the first phase, the algorithms overestimate the utility of the itemsets to obtain a set of candidate HUIs using the TWU measure and other strategies. Then, in the second phase, they scan the database again to calculate the utility of these candidates and filter out those that are not HUIs. Although these algorithms are complete, since they can find the whole set of HUIs, the two-phase approach may require considering and maintaining in memory a very large number of candidate itemsets. In the second phase, the cost of scanning the database for each itemset to calculate its utility is also very high. As a result, these algorithms can be slow and consume huge amounts of memory. This includes algorithms such as Two-Phase [112], UP-Growth+ [157], PB [98] and BAHUI [152].

One-Phase Approach

In order to avoid the drawbacks of the two-phase approach, algorithms have been proposed to mine HUIs using a single phase [110, 57]. These algorithms can directly calculate the utility of an itemset in memory without having to repeatedly scan the database or retain candidates in memory. The concept of a single-phase algorithm was introduced in the HUI-Miner algorithm [110], which used a novel structure called a utility-list. This structure stores all the information needed to calculate the utility of each itemset and reduce the search space, without repeatedly scanning the database. It also utilizes tighter upper-bounds and more efficient strategies to reduce the search space than two-phase algorithms.

The Utility-List Structure

The utility-list structure was proposed in HUI-Miner [110] to discover HUIs in a single phase, and hence avoid the drawbacks of two-phase algorithms,

which require the storage of a large number of candidates in memory and repeated scanning of the database to calculate the utilities of itemsets.

Let $\succ$ be any total order on items from I, and let X be an itemset appearing in a database D. The *utility-list* of X is denoted as $ul(X)$. It contains a tuple $(tid, iutil, rutil)$ for each transaction T_{tid} in which X appears ($X \subseteq T_{tid}$). The *iutil* element of a tuple for a transaction T_{tid} stores the utility of X in the transaction T_{tid}, i.e., $u(X, T_{tid})$. The *rutil* element of a tuple stores the value $\sum_{i \in T_{tid} \wedge i \succ x \forall x \in X} u(i, T_{tid})$, and is called the remaining utility of X [110].

HUI-Miner uses utility-lists to store information about the utilities of itemsets in transactions. This information allows the algorithm to quickly derive the utility of any itemset and to calculate the upper-bounds on the utilities of its supersets to reduce the search space. To discover HUIs, HUI-Miner scans the database to create a utility-list for each item. It then performs a depth-first search to explore the search space of all itemsets containing more than one item. During this search, the utility-list is constructed for each itemset by joining the utility-lists of some of its subsets, that is, without scanning the database. Two important properties of utility-lists can be used to determine the utility of an itemset and to reduce the search space, and these are defined below [110].

Property 3 (Calculating the utility using the sum of iutil values [110])**.** *The utility of an itemset X (denoted as $u(X)$) can be calculated by summing the iutil values in the utility-list $ul(X)$. If this sum is less than the minutil threshold, X is a low utility itemset. Otherwise, it is an HUI [110].*

Property 4 (Pruning using the iutil and rutil values of a utility-list [110])**.** *Let X and Y be two itemsets. It is said that Y is an* extension *of X if Y can be obtained by adding an item c to X, where $c \succ i$, $\forall i \in X$. The sum of the iutil and rutil values in the utility-list $ul(X)$ is an upper-bound on the utility of Y and any other transitive extension of X. As a consequence, if this sum is less than the minutil threshold, it follows that any itemset that is a transitive extension of X must be a low utility itemset, and should therefore be pruned.*

Creating the utility-lists for itemsets using this join operation is costly, and requires a significant amount of memory since an algorithm has to maintain many utility-lists in memory during the search for HUIs. In terms of the execution time, the complexity of building a utility-list is also high [57], as it generally requires the joining of three utility-lists of smaller itemsets. Recently, improved versions of the HUI-Miner algorithm called HUP-Miner [93] and FHM [57] have been proposed, which introduce additional strategies and

optimizations for search space pruning. FHM applies a strategy to eliminate low utility itemsets using information about item co-occurrences. For each itemset eliminated using this strategy, a join operation does not need to be applied, thus reducing the execution time. It has been shown that this pruning strategy can greatly reduce the number of join operations.

2.2.1 Dynamic Databases in Data Mining

The goal of HUIM is to discover patterns that generate high profits in static customer transaction databases. The utility-list structure can be used to mine HUIs in a single phase, i.e., without maintaining candidates in memory. The utility of each itemset can be directly calculated using its utility-list without scanning the database again. The simplicity of the utility-list structure has led to the development of numerous utility-list-based algorithms [110, 45, 37], and these generally outperform alternative techniques. Although several studies of HUIM have been conducted [112, 55], most of the existing approaches are suitable for pattern discovery in a static database. With increasing amounts of data being generated as streams, including customer transactions, HUIM must also support pattern discovery in dynamic databases.

2.2.2 Mining in Streaming Data

Consider a data stream S that is an open-ended sequence of values $\{v_1, v_2, \ldots, v_i, \ldots\}$. We assume that our observation V_i of v_i in the data stream is drawn from a Gaussian model with an unknown distribution, $V_i \sim \mathcal{N}(\mu, \sigma^2 I)$, where μ is the (unknown) mean and is used as the measure of interest in our method, and σ^2 is the variance in the error. At each observation time i, the observed mean is μ_i. Hence, for a data stream S, we have a sequence of observed means $\mu_1, \mu_2, \ldots, \mu_i, \ldots$ corresponding to observation times $1, 2, \ldots, i, \ldots$.

Customer transactions in retail stores can be treated as a stream of data, since customers continuously purchase products in stores. This also means that the data are not static, and are often impossible to store in memory due to the large volumes involved.

Streaming Transactional Data

Similarly to general data streams, streaming transactional data are generally infinite and changes continuously. Consider an infinite sequence of increasing positive integers $1 \leq x1 < x2 < x3 \ldots$. A streaming quantitative transactional database D is an infinite sequence of transactions

$D = \{T_{x1}, T_{x2}, T_{x3}, \dots\}$, where for each transaction $T_d \in D$, the relationship $T_d \in I$ holds. Moreover, for each transaction $T_d \in D$, d is a unique integer that is said to be the TID of T_d, and represents its observation time. Thus, for two transactions T_a and T_b, if $a < b$, this indicates that transaction T_a occurred before T_b. We assume that the stream D does not contain two transactions with the same observation time[1]. For two observation times a and b, a data window W_{ab} is a finite sub-sequence of D containing all transactions from observation time a to observation time b. Formally, $W_{ab} = \{T_{y1}, T_{y2}, \dots, T_{yn}\}$, for all integers $y1, y2, \dots, yn$ such that $a \leq y1 < y2 < \dots < yn \leq b$, and $T_{y1}, T_{y2}, \dots, T_{yn}$ appear in D.

For a window W in a stream D and an itemset X, the utility of X in W is defined as $u(X, W) = \sum_{X \subseteq T_d \wedge T_d \in W} u(X, T_d)$. Let the minimum utility threshold *minutil* be a positive number specified by the user such that $0 < minutil$. Then, an itemset X is said to be an HUI in W if its utility is no less than *minutil*, $u(X, W) \geq minutil$. Otherwise, X is said to be a *low utility itemset* in W.

Combined with the high speed of data generation, this makes mining a stream of transactional data to discover patterns more challenging than mining a static database. The development of efficient methods and algorithms to analyze transaction streams is therefore an important research problem [29, 114]. Several algorithms have been proposed to discover high utility patterns in data streams [173, 102, 40], and these studies generally extend traditional HUIM methods to increase their efficiency in a streaming context. On the whole, most studies on this topic, including [103, 173], have focused on adapting traditional data mining techniques to streams and on improving their efficiency to deal with streaming data.

It should be noted, however, that the underlying distribution of data objects in a stream generally changes over time [15], thus making such approaches unsuitable. At the same time, the detection of changes (*concept drift*) is crucial, as it allows us to discover the latest trends in a stream. A concept drift mainly refers to a significant decrease or increase in the distribution of data objects in a data stream with respect to a given measure [66]. Important challenges in terms of analyzing data streams arise because trends may emerge or remain steady over time, and streams often contain noise. In other words, in order to allow decision makers to react quickly to changes, it is necessary to design efficient algorithms that can detect and monitor these changes in real time.

[1] If two transactions are simultaneous, a total order on these transactions can be obtained by incrementing the observation time of one of those transactions by a small value.

2.3 Detection of Concept Drift

Central problems when analyzing data streams include the extraction of interesting events (such as patterns, changes, fraud, and network attacks), and understanding these activity trends. In the context of event detection in streaming data, due to the large volumes of data and the fact that the data are dynamic and continuously arrive at a high speed, online learning approaches are necessary to be able to meet the challenges arising from the different characteristics and the evolution over time of the underlying distribution.

2.3.1 The Problem of Change Detection in Streaming Data

Change detection methods in streaming data consider the difference between adjacent means μ_i and μ_{i+1}, where adjacent means are not necessarily equal. A change detection method is designed to detect all points of change i between two observed adjacent means μ_i and μ_{i+1}, for any i and $\mu_i \neq \mu_{i+1}$. We assume that $t_1, t_2, \ldots t_j, \ldots$ are the true points of change in a distribution of means. The change detector verifies a change point t_j by testing the following statistical hypotheses:

$$H_0 : \mu_{t_{j-1}} \simeq \cdots \simeq \mu_{t_j-1} \simeq \mu_{t_j} \simeq \mu_{t_j+1} \simeq \cdots \simeq \mu_{t_{j+1}-1}$$

against

$$H_1 : \mu_{t_{j-1}} \simeq \cdots \simeq \mu_{t_j-1} \neq \mu_{t_j} \simeq \mu_{t_j+1} \simeq \cdots \simeq \mu_{t_{j+1}-1}$$

Given a significance confidence ρ, hypothesis H_1 is accepted if an objective measure of interest for the mean, i.e., $f(.)$, satisfies:

$$f(\mu_{t_j}) \notin [v_{\frac{\rho}{2}}, v_{1-\frac{\rho}{2}}], \tag{2.4}$$

where $f(.)$ is an objective function, $v_{\frac{\rho}{2}}$ and $v_{1-\frac{\rho}{2}}$ are values such that:

$$Pr(f(\mu_{t_j}) < v_{\frac{\rho}{2}}) = \frac{\rho}{2} \tag{2.5}$$

$$Pr(f(\mu_{t_j}) > v_{1-\frac{\rho}{2}}) = 1 - \frac{\rho}{2} \tag{2.6}$$

A change is said to occur at observation time t_j if the population of the sample at time t_j is significantly different from that at the preceding observation time, and the null hypothesis H_0 is rejected. In this case, t_j is said to be a drift point. This means that the task of detecting change points involves finding all observation times at which there are significant differences in the data distribution for a given measure.

Types of Drift

As mentioned above, the main challenge associated with the mining of streaming data is that data in a stream are inherently dynamic, and the underlying distribution can change and evolve over time, leading to what is referred to as *concept drift* [141]. For example, data on customer purchasing behavior over time may be influenced by the strength of the economy. In this case, the concept is the strength of the economy, which may drift. Concept drift can be either real or virtual, and changes in the distribution can have four different forms [63]: *abrupt*, *reoccurring*, *incremental*, and *gradual*, as illustrated in Figure 2.3. In streaming data, we may also have mixtures of these types.

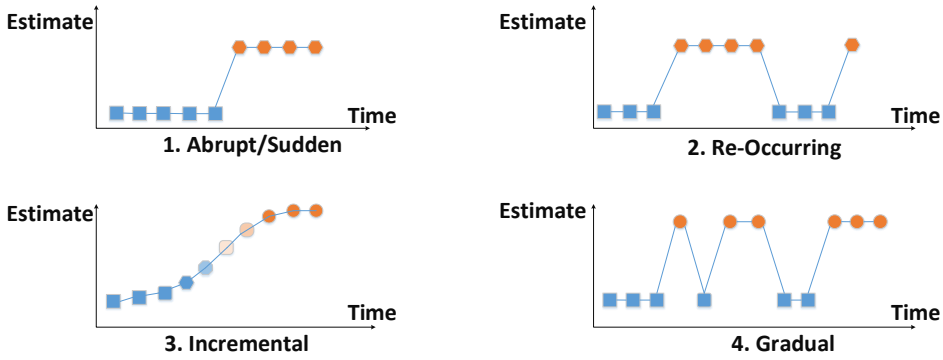


Figure 2.3: The four different types of concept drift.

2.3.2 Approaches to Change Detection

Techniques for change detection are generally based on one of the following approaches [63] (see also Table 2.3 for a summary):

1. *Sequential Analysis*: The cumulative sum (CUSUM) algorithm and its variant, the Page-Hinkley (PH) method [124], are the representative algorithms in this category. The main idea of this approach is to estimate a probability distribution value and to update this value when new data arrive. Concept drift occurs if there is a significant change in the values of the estimated parameters. The main advantage of sequential analysis algorithms is that they generally have low memory consumption. One of the disadvantages, however, is that the performance depends on the choice of parameters.

2. *Statistical Process Control*: The idea underlying this method is to estimate statistical information such as the error and standard deviation. By monitoring this error rate, detectors can determine that a concept drift has occurred if the error increases above a pre-specified threshold value. Examples of methods in this group are DDM [64], EDDM [13], ECDD [134], and AFF [22]. The main advantage of the methods in this category is that in addition to their efficiency and low memory consumption, they generally work well on streams with abrupt changes. On the other hand, slow, gradual change detection is a significant weakness of these approaches.

3. *Monitoring of Distributions using Two Different Time Windows*: In this approach, statistical tests are applied to the distributions of two windows in a stream. The first window is a fixed reference window used to summarize the information from past data, while the second is a sliding window used to summarize the information on the most recent data. Examples of representative algorithms in this category are ADWIN [18], HDDM [61], and SEQDRIFT2 [127]. The main advantage of the window-based methods is that they are able to identify more precise locations for change points. Their disadvantages are their high memory cost and the space requirements for processing the two windows.

4. *Contextual-Based*: The approaches in this category maintain a balance between incremental learning and weighting schema in the detection of concept drift with respect to the time of an estimated window. An example of a contextual approach is SPLICE [77]. The main advantages of methods in this category are that they can detect gradual and abrupt drift, and can control the number of errors. However, contextual-based methods are generally complex and have long execution times, thus making them less suitable for the mining of streaming data.

Hoeffding's Inequality for the Significance of Drift Assessment

The Hoeffding inequality [80] is derived from probability theory, and has been used in various studies to analyze data streams. Given a set of independent random variables, the Hoeffding inequality provides an upper bound on the probability that their sum deviates from an expected value. In a change detection problem, the Hoeffding inequality is used as a basis for assessing whether changes in a set of independent random values arriving in

Table 2.3: Summary of change detection methods

Category	Algorithms	Advantages	Disadvantages
Sequential analysis	CUSUM [124], PAGE-HINKLEY [124]	Low memory consumption.	Depends on the choice of parameters.
Statistical process control	DDM [64], EDDM [13], ECDD [134], AFF [22], RDDM [16]	Low memory consumption, work well on abrupt changes, and fast execution time.	Slow gradual changes.
Windows-based methods	Kifer et al. [88], ADWIN [18], HDDM [61], SEQDRIFT2 [127], ACWM [142]	Precise localization of change points.	Memory and space requirements for processing two windows.
Contextual approaches	SPLICE [77]	Gradual and abrupt drift, and control of the number of errors.	Complex and difficult to implement and long execution time.

a data stream are statistically significant. If the probability of a predefined condition is greater than a user-specified threshold, then a change is considered to have occurred in the data at this observation point. Hoeffding's inequality theorem is stated as follows [80].

Theorem 1 (Hoeffding's Inequality)**.** *Let U_1; U_2; ... ; U_n be a set of independent random variables bounded by the interval [0, 1], that is,* $0 \leq U_i \leq 1$*, where* $i \in \{1; \ldots; n\}$*. Let* $\overline{U}$ *denote the average of these random variables, that is,* $\overline{U} = \frac{1}{n}\sum_{i=1}^{n}(U_i)$*. We have:*

$$Pr(\overline{U} - E[\overline{U}] \geq \varepsilon) \leq e^{-2n\varepsilon^2}, \tag{2.7}$$

*where E[*X*] is the expected value of* X*.*

The inequality states that the probability of the estimation and true values differing by more than ε is bounded by $e^{-2n\varepsilon^2}$. In a symmetrical way, the inequality is also applied to the other side of the difference: $Pr(-\overline{U} + E[\overline{U}] \geq \varepsilon) \leq e^{-2n\varepsilon^2}$. As a result, a two-sided variant of the inequality is obtained:

$$Pr(|\overline{U} - E[\overline{U}]| \geq \varepsilon) \leq 2e^{-2n\varepsilon^2} \tag{2.8}$$

This inequality is true if all variables are bounded by the interval [0, 1]. More generally, if a variable U_i is bounded by an interval $[x_i, y_i]$, Hoeffding's inequality is generalized as follows:

$$Pr(|\overline{U} - E[\overline{U}]| \geq \varepsilon) \leq 2e^{\frac{-2n^2\varepsilon^2}{\sum_{i=1}^{n}(y_i - x_i)^2}} \tag{2.9}$$

This inequality (Theorem 2) can be used to assess the significance of changes in a stream of values, based on the following proposition.

Proposition 1. *Let $U_1; U_2; \ldots; U_{n_1+n_2}$ be a set of independent random variables bounded by the interval [0, 1]. These variables can be split into two windows using an index n_1 as the splitting point. This results in two windows, $\{U_1; \ldots; U_{n_1}\}$ and $\{U_{n_1+1}; \ldots; U_{n_1+n_2}\}$. Then, for an error $\varepsilon > 0$, the following inequality holds:*

$$Pr(\overline{U} - \overline{V} - (E[\overline{U}] - E[\overline{V}]) \geq \varepsilon) \leq e^{\frac{-2n_1 \times n_2\varepsilon^2}{n_1+n_2}}, \tag{2.10}$$

where $\overline{U} = \frac{1}{n_1}\sum_{i=1}^{n_1}(U_i)$ and $\overline{V} = \frac{1}{n_2}\sum_{i=n_1+1}^{n_1+n_2}(U_i)$.

If the two-sided variant of the inequality is considered:

$$Pr(|(\overline{U} - E[\overline{U}]) - (\overline{V} - E[\overline{V}])| \geq \varepsilon) \leq 2e^{\frac{-2n_1 \times n_2\varepsilon^2}{n_1+n_2}} \tag{2.11}$$

Consider a significant confidence level α (i.e., the probability of making an error), which controls the maximum false positive rate. The error ε can be estimated with respect to α as follows:

$$\varepsilon = \sqrt{\frac{n_1 + n_2}{2n_1 \times n_2} \ln \frac{2}{\alpha}} \tag{2.12}$$

Bonferroni Correction

The Bonferroni correction prevents an increase in the probability of incorrectly rejecting the null hypothesis when multiple hypotheses are tested [144]. In the context of the change detection problem, the Bonferroni correction is widely used with Hoeffding's inequality. Rather than using Eq. 2.12, the following formula is used:

$$\varepsilon_\alpha = \sqrt{2m\sigma^2 \ln \frac{2\ln(n)}{\alpha}} + \frac{2m}{3} \ln \frac{2\ln(n)}{\alpha} \tag{2.13}$$

where $n = n_1 + n_2$, $m = n_1^{-1} + n_2^{-1}$, and the variance σ^2 is defined as the sum of the squared distances of each sample in the distribution from the

mean, divided by the number of samples in the distribution. An efficient way of calculating the standard deviation for a set of numbers $(x_i)_{i=1}^{N}$ is as follows:

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N} x_i^2 - (\frac{\sum_{i=1}^{N} x_i}{N})^2 \tag{2.14}$$

2.3.3 Distance and Similarity

This section presents some metrics that are commonly used to measure the similarity between objects. The change detector is used to evaluate whether there is a significant difference between the similarities; based on this evaluation, a decision is made on whether or not a change has occurred.

Cosine Similarity

Consider two vectors $\mathbf{x} = (x_1, x_2, \ldots, x_n)$, $\mathbf{y} = (y_1, y_2, \ldots, y_n)$. The cosine similarity between $\mathbf{x}$ and $\mathbf{y}$ is defined by:

$$S_{cos}(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x} \times \mathbf{y}}{\|\mathbf{x}\| \times \|\mathbf{y}\|} = \frac{\sum_{i=1}^{n}(x_i \times y_i)}{\sqrt{\sum_{i=1}^{n} {x_i}^2}\sqrt{\sum_{i=1}^{n} {y_i}^2}} \tag{2.15}$$

The cosine similarity is a standard measure for calculating the similarity between vectors. However, a drawback of this approach is that it only considers the difference in the orientation of two vectors, while ignoring the difference in terms of the magnitude [79]. For example, a cosine similarity of 1 indicates that two vectors have the same orientation, although these vectors may or may not have the same magnitude. The magnitude of an itemset vector represents its utility. In order to take into account both the difference in the orientation and the magnitude, an efficient approach needs to calculate the similarity between two vectors $\mathbf{x}$ and $\mathbf{y}$ using an improved similarity measure that considers both the magnitude and orientation of two vectors.

Levenshtein Distance

The Levenshtein distance [101] is a metric used to measure the difference between two strings. Given two strings $x = x_1x_2 \ldots x_n$, and $y = y_1y_2 \ldots y_m$, the Levenshtein distance between x and y is denoted as $lev_{x,y}$, and is com-

puted as $lev_{x,y}(n, m)$, where:

$$lev_{x,y}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0 \\ \min \begin{cases} lev_{x,y}(i-1,j)+1 \\ lev_{x,y}(i,j-1)+1 \\ lev_{x,y}(i-1,j-1)+\mathbb{1}_{\{x_i \neq y_j\}} \end{cases} & \text{otherwise.} \end{cases} \tag{2.16}$$

$\mathbb{1}_{\{z\}}$ is binary indicator function. In other words, $\mathbb{1}_{\{z\}}$ is equal to 1 if z is TRUE; otherwise, $\mathbb{1}_{\{z\}}$ is equal to 0.

$$\mathbb{1}_{\{z\}} = \begin{cases} 1 & \text{if z is TRUE} \\ 0 & \text{otherwise.} \end{cases} \tag{2.17}$$

Euclidean Norm

Given a vector $\mathbf{x} = (x_1, \ldots, x_N)$ in a Euclidean space D of N dimensions $\mathbb{R}^N$, the Euclidean norm of $\mathbf{x}$ in D is computed by:

$$\| \mathbf{x} \|_2 = \sqrt{x_1^2 + x_2^2 + \cdots + x_N^2} = \sqrt{\sum_{i=1}^{N} x_i^2} \tag{2.18}$$

Let p be a real number, $p \geq 1$. The p-norm of $\mathbf{x}$ is:

$$\| \mathbf{x} \|_p = \sqrt[p]{|x_1|^p + |x_2|^p + \cdots + |x_N|^p} = \sqrt[p]{\sum_{i=1}^{N} |x_i|^p} = (\sum_{i=1}^{N} |x_i|^p)^{\frac{1}{p}} \tag{2.19}$$

The Euclidean norm is the p-norm when $p = 2$.

2.3.4 State of the Art in Change Detection

In the field of traditional frequent itemset mining, several algorithms have been proposed to identify concept drift in data streams [121, 90]. Ng et al. [121] proposed a test paradigm called the Algorithm for Change Detection (ACD) that detected changes in transactional data streams by considering the frequency of itemsets for its reservoir sampling process. ACD evaluates drift by performing reservoir sampling and applying three statistical tests. This method employs a bound based on Hoeffding's inequality to determine the number of transactions to be kept in each reservoir, and selects the transactions to fill each reservoir using a distance measure that is a function of the frequency of single items. A major limitation of this

approach is that high frequency items have more probability of being sampled. Recently, Koh [90] proposed the CD-TDS algorithm, which considers two types of changes in transactional data streams for frequent pattern mining. This method uses a graph to represent the relationships between items in transactions, and applies the Levenshtein distance to calculate the similarities between groups of frequent itemsets found at different times. A limitation of CD-TDS is that it detects local drifts for single items, rather than for itemsets. Although CD-TDS uses itemsets to detect global drifts, it does not provide information about which itemsets contribute the most to these global drifts.

Hoeffding's inequality [80] is one of the most common inequalities, and has been used to design several upper bounds for drift detection [61]. These upper bounds have been used in algorithms such as the Drift Detection Method Based on Hoeffding's Bounds (HDDM) [61], the Fast Hoeffding Drift Detection Method for Evolving Data Streams (FHDDM) [129], Hoeffding Adaptive Tree (HAT) [19], and the HAT + DDM + ADWIN [4] algorithm which extends the ADaptive sliding WINdow (ADWIN) algorithm [18], the Drift Detection Method (DDM) [64], and the Reactive Drift Detection Method (RDDM) [16], a recent drift detection algorithm based on DDM. Hoeffding's inequality has also been used to design several efficient approaches to determining the upper bounds for drift detection. However, the Hoeffding inequality has the disadvantage that the dependence on the underlying distribution is eliminated [161]. Although these algorithms are useful for detecting changes in streaming data, most of them assume that the data are independent and identically distributed. This assumption rarely holds in real-world streaming environments.

Independent and Identical Distribution

Consider a set of random generated data D, and two random variables x and y. x and y are independent, meaning that the probability of observing both x and y is simply the product of the probability of observing each event individually.

$$Pr(x, y) = Pr(x) \times Pr(y), \tag{2.20}$$

where the $Pr(z)$ notation is used to indicate the probability of observing z. An independent distribution also implies conditional independence: the probability of observing x does not depend on y.

$$Pr(x|y) = Pr(x) \tag{2.21}$$

Two random variables x, y are identically distributed if $Pr(x \leq z) = Pr(y \leq z), \forall z$. Most existing work is based on an assumption of an identical and

independent distribution (IID) for the data; however, as mentioned earlier, this assumption is too restrictive. Real-world streaming data are inherently dynamic, and are not identically and independently distributed [21, 178]. Temporal dependencies are also very common in data streams [178], and should therefore be considered when addressing this issue. Several learning algorithms that focus on non-stationary environments have been proposed to overcome the limitations of the IID assumption [104, 92], although the majority of existing approaches have assumed that data in a stream are independently but not identically distributed [21, 178]. Adaptive estimation is one such technique for handling the temporal dependencies of data. For example, the studies in [124, 132, 22, 8] employed an adaptive estimation methodology by introducing a so-called 'forgetting factor' [67, 130].

Simplex and the ℓ_1 Ball Projection

Consider a data space D of N dimensions, $D = \mathbb{R}^N$. Let $\mathbf{y}$ be a vector in the data space D, $\mathbf{y} = (y_1, y_2, \ldots, y_N) = (y_i)_{i=1}^N \in D = \mathbb{R}^N$. $\mathcal{S}_{\mathcal{CS}}$ is a convex set, and $\prod$ is a projection. A vector $\mathbf{x}$ is called an image of $\mathbf{y}$ after the projection $\prod$ subject to the constraint of $\mathcal{S}_{\mathcal{CS}}$, and is denoted by: $\mathbf{x} = \prod_{\mathcal{S}_{\mathcal{CS}}}(\mathbf{y})$.

Given a real value r, the simplex projection of a vector $\mathbf{y}$ is defined as follows:

$$\prod_{\Delta}(\mathbf{y}) = \arg\min_{\mathbf{x}\in\Delta} \frac{1}{2} \parallel \mathbf{x} - \mathbf{y} \parallel, \tag{2.22}$$

where $\parallel . \parallel$ is the Euclidean norm, and Δ is the simplex of the data space D with respect to r. $\Delta \subset D$, and is defined by:

$$\Delta = \{(x_1, x_2, \ldots, x_N) \in \mathbb{R}^N \Big| \sum_{i=1}^{N} x_i = r \wedge x_i \geq 0, \forall i = 1, \ldots, N\} \tag{2.23}$$

When the value of r is 1, the simplex Δ is called the standard or probability simplex. If the conditional constraints on the simplex Δ change to $\sum_{i=1}^{N} \mid x_i \mid \leq r$, we have a new sub space:

$$\mathcal{B} = \{(x_1, x_2, \ldots, x_N) \in \mathbb{R}^N \Big| \sum_{i=1}^{N} \mid x_i \mid \leq r\} \tag{2.24}$$

$\mathcal{B}$ is called the ℓ_1 ball sub space of D, and the ℓ_1 ball projection is as follows:

$$\prod_{\mathcal{B}}(\mathbf{y}) = \arg\min_{\mathbf{x}\in\mathcal{B}} \frac{1}{2} \parallel \mathbf{x} - \mathbf{y} \parallel, \tag{2.25}$$

2.4 Event Detection in Complex Data

In many real-world applications, generated data are commonly represented using a more complex structure such as a graph or multidimensional array, often referred to as a *tensor* [31]. Tensors and graphs have been used in several important domains, including geometry, physics, biology and computer science [176, 81, 145]. As a result of the growth in the number of applications involving tensors and graphs, combined with the increase in the range of interests of researchers in this field, numerous tensor- and graph-related approaches have been proposed, including tensor decomposition [150, 107], tensor factorization [171, 123, 126], and dense subgraph detection [95, 68, 99]. Some effective algorithms have used the greedy method [12] with a guarantee on the density of dense subgraphs, and have been applied in specific fields such as fraud detection, event detection, and genetics [52, 159, 82, 148, 138], among others. Dense subgraph detection and dense subtensor detection are core tasks in a wide-range of real-life applications [60, 136, 174, 143].

2.4.1 Dense Detection in Graph and Tensor Data

In this subsection, we introduce the background to the problem of dense subgraph and dense subtensor detection.

Dense Subgraph Detection

Definition 1 (Graph)**.** *Let G be an undirected graph that is composed of a pair $(V; E)$ of sets of vertices V and edges E. We denote the graph as $G(V; E)$. There is a weight a_i at each vertex v_i, and a weight c_{ij} on each edge e_{ij} between two vertices v_i and v_j in G.*

Definition 2 (Density of Graph)**.** *The density of G is denoted by $\rho(G)$ and is defined by:*

$$\rho(G) = \frac{\sum a_i + \sum c_{ij}}{|V|} = \frac{f(G)}{|V|}, \tag{2.26}$$

where $|V|$ is the number of vertices of G, and $f(G) = \sum a_i + \sum c_{ij}$, $f(G)$ is called the mass of graph G.

Definition 3 (Subgraph)**.** *Let G be an undirected graph that is composed of a pair $(V; E)$ of sets of vertices V and edges E. S is a subgraph of G if S is induced by a subset of the vertices of V and edges in E.*

Definition 4 (Weight of Vertex in Graph)**.** *Given a graph $G(V, E)$ with a weight a_i at vertex v_i, and a weight c_{ij} at the edge between 2 vertices v_i, v_j. The weight of vertex v_i in graph G is denoted by $w_i(G)$, and is defined by:*

$$w_i(G) = a_i + \sum_{v_j \in G \wedge e_{ij} \in E} c_{ij}. \tag{2.27}$$

Dense Subgraph Detection

Given an undirected graph $G = (V; E)$ and a density measure *df*, the problem of dense subgraph detection involves finding the subgraphs S induced by a subset of the vertices of V and edges in G to maximize the density of S.

Density Measures

To measure the density of a region S in a data D, four density measures are commonly used in benchmark evaluations for dense region detection, as follows:

1. Arithmetic average mass [27]: This density function is simply computed as follows:
$$df(S) = \frac{mass\ of\ S}{size\ of\ S} \tag{2.28}$$

2. Geometric average mass [27]: This metric was introduced under an observation from the problem in the context of sparse space [86].
$$df(S) = \frac{mass\ of\ S}{volume\ of\ S} = \frac{mass\ of\ S}{\sqrt[N]{\prod_{i=1}^{N} S_i}}, \tag{2.29}$$
where N is the number of dimensions of S, and S_i is the size along the i-th dimension of S.

3. Entry surplus [159]: This metric uses a parameter α to control the probability of each edge in S, to get better results in terms of a smaller diameter and higher density.
$$df_\alpha(S) = mass\ of\ S - \alpha \times \binom{size\ of\ S}{2} \tag{2.30}$$
The underlying idea of this metric is to express the edge surplus and then maximize the density with a smaller region.

4. Suspiciousness [85]: This metric was used in [85] in place of the arithmetic density. It is the negative log likelihood of the mass under an Erdös–Rényi model [51]. This metric is useful in fraud detection problems, and is computed by:

$$df_{Susp}(S \text{ over } D) = (\prod_{i=1}^{N} S_i) \times D_{KL}(df(S) \parallel df(D)), \tag{2.31}$$

where $D_{KL}(x \parallel y)$ is the Kullback-Leibler(KL) divergence [94] of Poisson(x) from Poisson(y) [73]. The Kullback–Leibler divergence of two given probability distributions P from Q on a space $\mathcal{X}$ is the expectation of the logarithmic difference between the probabilities, and is defined by:

$$D_{KL}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log \frac{P(x)}{Q(x)} \tag{2.32}$$

Dense Subtensor Detection

In the following, we present some fundamental preliminaries for the dense subtensor detection problem, based on [149, 147].

Definition 5 (Tensor)**.** *A tensor T is a multidimensional array data. The order of T is the number of its ways. Given a N-way tensor, there are multiple spaces on each way, each of which is called a slice.*

Definition 6 (SubTensor)**.** *Given an N-way tensor T, Q is a subtensor of T if it is composed of a subset s of the set of slices S of T, and there is at least one slice on each way of T. Intuitively, Q is the part of T that remains after we remove all slices in S but not in s.*

Definition 7 (Entry of a Tensor)**.** *E is an entry of an N-way (sub)tensor T if it is a subtensor of T and is composed of exactly N slices.*

Definition 8 (Size of a (Sub)Tensor)**.** *Given a (sub)Tensor Q, the size of Q is the number of slices making up Q.*

Definition 9 (Density)**.** *Given a (sub)tensor Q, the density of Q, denoted by $\rho(Q)$, is computed as: $\rho(Q) = \frac{f(Q)}{size\ of\ Q}$, where $f(Q)$ is the mass of the (sub)tensor Q, and is calculated as the sum of every entry value of Q.*

Definition 10 (Weight of a Slice in a Tensor)**.** *Given a tensor T, the weight of a slice q in T is denoted by $w_q(T)$, and is defined as the sum of the entry values composing the intersection of T and q.*

Definition 11 (D-Ordering)**.** *An ordering π on a (sub)tensor Q is a D-Ordering if*

$$\forall q \in Q, q = \underset{p \in Q \wedge \pi^{-1}(p) \geq \pi^{-1}(q)}{argmin} w_p(\pi_q), \tag{2.33}$$

where

$$\pi_q = \{x \in Q | \pi^{-1}(x) \geq \pi^{-1}(q)\}, \tag{2.34}$$

$\pi^{-1}(q)$ indicates the index of the slice q in ordering π, and $w_p(\pi_q)$ is the weight of p in π_q. Intuitively, the D-Ordering is the order in which we select and remove the minimum slice sum at each step.

Mining of Dense Subtensors

Given a tensor T, the problem of dense subtensor detection involves finding subtensors $Q \in T$ that maximize the density of Q.

Several methods of dense subtensor detection have been proposed using the same min-cut mechanism as for dense subgraph detection [148, 147, 174]. These methods adapt the theoretical results from dense (sub)graph detection, i.e., [10, 9, 159], to tensor data by considering more than two dimensions. The algorithms utilize a greedy approach to guarantee the density of the estimated subtensors, and this has been shown to yield high levels of accuracy in practice [85]. However, the same guarantee as in the original work was employed, with no improvement in density guarantee.

2.4.2 Related Work on Dense Region (Subtensor, Subgraph) Detection

Finding the densest subtensor or subgraph is generally an NP-complete or NP-hard problem [70, 11], and the hardness of this problem varies with the choice of constraints. Due to the complexity of the exact algorithm, it is infeasible for use with big data or in dynamic environments such as streaming. Approximation methods are therefore commonly used to detect the densest regions [12, 27, 14]. GREEDY is an efficient approximation algorithm that finds the optimal solution in a weighted graph [12]. A further analysis of the GREEDY algorithm [27] has shown that the problem can be solved using a linear programming technique. The authors proposed a greedy 2-approximation for this optimization problem with a density guarantee of the dense subgraph greater than half of the maximum density in the graph. The process used by the greedy approximation algorithm is as follows [12, 27]. The vertex with the minimum weighted degree is iteratively removed from the currently remaining graph until all vertices have been removed. Finally,

the subgraph with the highest density is chosen from among the estimated subgraphs.

Several algorithms using this method have been applied in fields such as fraud detection, event detection, and genetics applications [159, 136, 82, 148, 138], among others. The common aspect connecting these works is that they use the greedy 2-approximation to find a dense subgraph to optimize an objective density for an interest density measure. Inspired by the theoretical solutions for graphs, numerous approaches have been proposed to detect dense subtensors by using the same min-cut mechanism [149, 147]. Various algorithms have been proposed that extend prior work on dense (sub)graph detection to tensor data for use in the areas of network attack detection and fraud detection [85, 148, 147]. However, the density guarantee in these approaches is the same as in the original work, with no improvement, and this also applies to recent algorithms such as *ISG+D-Spot* [174] and *BFF* [143]. For an N-way tensor, the guarantee is a fraction of the highest density with the number of ways of the tensor N. Although *ISG+D-Spot* converts an input tensor to a form of graph to reduce the number of ways, it drops all edges with weights less than a certain threshold, meaning that the density guarantee is lost.

The greedy 2-approximation has been utilized in many algorithms with both graph and tensor data [154, 138, 137, 143]. Despite this, most current works, including [82, 120, 138, 147, 174] can only roughly provide a guarantee of half (or $\frac{1}{N}$ for a subtensor) the density of the densest subregion. None of the existing approximation schemes provides a better guarantee than the baseline algorithms [12, 27], and such schemes can only provide a loose theoretical density detection guarantee.

2.4.3 Multiple Dense Subtensor Estimation and Related Work

Dense subtensor detection is a well-studied area with a wide range of applications, and numerous efficient approaches and algorithms have been proposed. Existing algorithms for dense subtensor detection are generally efficient, and can perform well in many applications. However, the main drawback of these algorithms is that most can estimate only one subtensor at a time, with a low guarantee of the density of the suntensor. Although some methods can estimate multiple subtensors, they can only give a guarantee on the density with respect to the input tensor for the first estimated subtensor. M-Zoom [146] and M-Biz [147] are among the current state-of-the-art dense subtensor detection algorithms. They extend approaches for dense (sub)graph detection, such as in [27, 62], to tensors by considering more dimensions for a specific problem to obtain highly accurate algorithms.

They also utilize a greedy approach to provide a local guarantee of the density of the estimated subtensors. M-Zoom and M-Biz are able of maintaining k subtensors at a time. Each time a search is performed, a snapshot of the original tensor is created, and the density of the estimated subtensor in each single search is guaranteed locally on the snapshot. Hence, M-Zoom and M-Biz provide only a density guarantee for the current intermediate tensor, rather than the original input tensor. A newer approach called DenseAlert [149] was developed to detect an incremental dense subtensor for streaming data. Despite its efficiency, DenseAlert can estimate only one subtensor at a time, and can only provide a low density guarantee for the estimated subtensor. It may therefore miss a very large number of other interesting subtensors in the stream. Extensive studies have shown that DenseAlert, M-Zoom, and M-Biz generally outperform most other tensor decomposition methods [91, 177] in terms of efficiency and accuracy. Nevertheless, an important drawback of these methods is that they can provide only a loose theoretical guarantee of density detection, and the results and efficiency are primarily based on heuristics and empirical observations. More importantly, these methods do not provide any analysis of the properties of the multiple estimated subtensors.

2.5 Sketching Using Multiple Weighted Factors with Concept Drift

The storage and processing of high volumes of streaming data with high velocity are often infeasible, due to limitations on memory and computational infrastructure. Normally, data in a stream can be accessed only once, thus making efficient processing of streams a challenging but crucial task. A possible solution involves sketching of the streaming data. Sketching is an effective approximation method that maps a stream to a maintainable form, while still retaining the characteristics of the stream with a high level of accuracy. The development of efficient sketching techniques has attracted much research attention over recent decades [165, 167, 164]. In addition to being used to visualize statistical information for the data, *histogram* estimation is a technique that is commonly used to sketch the underlying distribution of the data [142]. An important limitation of most existing histogram-based methods is that they were developed based on the assumption that histograms were drawn from a non-changing or static data distribution [72, 153]. In practice, streaming data are inherently dynamic and evolve over time, resulting in dynamic changes in the underlying data distribution, i.e., concept drift [63].

To address issues related to concept drift, the concept of forgetting factor has been introduced to determine the significance of data based on the time at which they occur in the stream [142, 168, 140]. However, most of the current work (e.g., [142] and [168]) assumes that the forgetting factor is constant at each point of observation and over the whole process. This is too restrictive, since in many real-world applications, we cannot use the same factor for each previous data item at a given observation point. In the following, we present fundamental definitions for terms related to the consistent weighted sampling [113, 84, 105] used in our solution involving sketching of data stream with concept drift. We assume two data vectors (weighted sets) of k elements $X, Y \in \mathbb{R}^k$, and present definitions as follows.

Definition 12 (Uniformity [113, 84, 105])**.** *Let (i, S_i) be a sample of X, where $1 \leq i \leq k$ and $0 \leq S_i \leq X_i$. A process is known as uniformity sampling if the sample is distributed uniformly over the pairs.*

Definition 13 (Consistency [113, 84, 105])**.** *Assume that X dominates Y, i.e., $Y_i \leq X_i$, for all i and $1 \leq i \leq k$. If a sample (i, S_i) is sampled from X and satisfies $S_i \leq Y_i$, then (i, S_i) is also sampled from Y.*

Definition 14 (Consistent weighted sampling [113, 84, 105])**.** *Consistent weighted sampling is a sampling process that satisfies the properties of both the uniformity and consistency.*

The min-max similarity between X, Y is defined by [84, 105]:

$$SIM_{min-max}(X,Y) = \frac{\sum \min(X_i,Y_i)}{\sum \max(X_i,Y_i)}$$

Let $SE(X)$ and $SE(Y)$ denote the samples of data vectors X and Y, respectively, using a consistent weighted sampling schema. The collision probability between the two vectors X, Y is exactly its min-max similarity [84, 105]:

$$Pr(SE(X) = SE(Y)) = SIM_{min-max}(X,Y).$$

Assume $SE(X) = \{i_x^*, s_x^*\}$, and $SE(Y) = \{i_y^*, s_y^*\}$ are two CWS samples of X and Y, respectively. Theoretical results for consistent weighted sampling show that:

$$Pr(i_x^* = i_y^*) \simeq Pr(\{i_x^*, s_x^*\} = \{i_y^*, s_y^*\}) \tag{2.35}$$

Given a data stream S, let t be the current time point, v_t the current incoming item at an instant in time (order) t with timestamp $\mathcal{T}_t$, and $V_t = (v_1, v_2, \ldots, v_t)$ the current sub-stream. The task of sketching a stream S at a point of time t involves maintaining a parameter sketch SK (s elements) such that SK is a compact representation of the current sub-stream V_t, and SK preserves the interest (similarity) measure of V_t.

2.5.1 Histograms

For a data stream, a histogram is a data summarization method, in which data are hashed into a set of buckets. Each bucket maintains the frequency of the hashed data. One common method for estimating the histogram of a data stream uses count-min sketching, originally proposed in [33], due to its efficiency and simplicity. A histogram of a data stream using a count-min sketch is created using a matrix $\Theta \in \mathbb{R}^{n \times m}$, where n is the number of random hash functions h_i, $i = 1, 2, \ldots, n$, and m is the number of ranges such that each hash function h_k maps an incoming data item v in the stream to a range from 1 to m. When a new data item v arrives in the stream, n hash

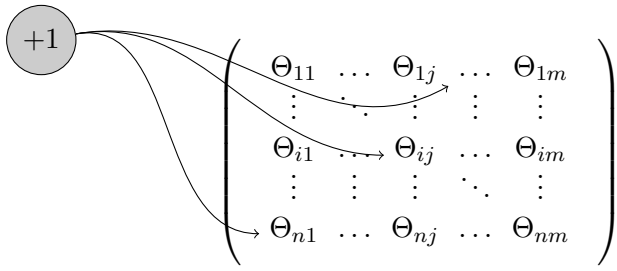


Figure 2.4: Example of an $\mathbb{R}^{n \times m}$ histogram.

functions h_i, $i = 1, 2, \ldots, n$, are used to hash the value v to a corresponding range (a set of m columns of matrix Θ). The value of the corresponding cell of matrix Θ is then incremented, i.e., $\Theta(i, h_i(v)) = \Theta(i, h_i(v)) + 1$.

2.5.2 Forgetting Factor and Elastic-Based Models

In this subsection, we first briefly introduce the forgetting factor, which is commonly used in weighting the importance of data in streaming data, then we present an elastic-based statistical model which we use in our studying of highly correlated features of evolving data.

Forgetting Factor

A forgetting factor [67, 130] was proposed to reduce the impact of noise in streaming data and weigh the importance of the data in the stream [168, 142, 140]. Given a factor λ, $0 \leq \lambda \leq 1$, and two observations at time t and $t-1$, the importance of a data item at these observation points is denoted as $W(t)$ and $W(t-1)$, and is given by:

$$W(t-1) = W(t) \times f(\lambda, \Delta(t)), \tag{2.36}$$

where $\Delta(t)$ is the timespan between observations, and f is a weight function of the variables λ and $\Delta(t)$ that is normally less than one.

Elastic Net

Elastic net regularization [179] is an improvement on the Lasso regularization scheme [155]. Lasso regularization was formulated for the least squares model, and utilized the ℓ_1-norm to minimize the least squares loss function to enhance the prediction accuracy of a statistical model. However, using only the ℓ_1 norm tends to select one variable from a group of highly correlated variables, and to set less important coefficients to zero. To overcome this drawback in the context of highly correlated variables, another term of 2-norm is added to enforce the hierarchy constraint, which helps to force the coefficients closer to the average value rather than to zero. The problem involves solving the following expression for $\mathbf{x}$:

$$\min_{\mathbf{x}}(\frac{1}{2} \parallel M\mathbf{x} - C \parallel_2^2 + \lambda \parallel \mathbf{x} \parallel_1 + \frac{1}{2}\beta \parallel \mathbf{x} \parallel_2^2), \tag{2.37}$$

where M, C, λ and β are parameters, and $\parallel . \parallel$ is the Euclidean norm.

2.5.3 k-NN Classification and the Hamming Distance

When sketching a data stream, one important task is to maintain a huge amount of data using a compact representation that is able to preserve the similarities between data points. The following paragraphs describe the use of the Hamming distance to measure the distance between data points, and discuss methods for classifying data into the corresponding bucket (classification) using distance measures.

Hamming Distance

The Hamming distance is a metric for computing the distance between two strings [74]. It is used in information theory, and is computed based on the number of positions with the same symbol in two strings. It is only defined for strings of the same length, and is different from the Levenshtein distance in that the latter is normally used to compare strings of different lengths.

k-Nearest Neighbors

k-nearest neighbors (k-NN) is a method of classifying a data point based on information on its k nearest neighbors. The k-NN classifier relies on a distance metric to compute the similarities between data points. This method selects the k nearest neighbors of a data point based on a predefined similarity. In the classification problem, the label for a data point is predicted

based on votes from its k nearest neighbors (e.g., the most frequent class). The value of k (the number of nearest neighbors) depends on the application and the data, and the value of k can be selected using heuristic methods.

2.5.4 Related Work

Sketching is a hashing technique [30] that has been extensively studied and widely applied to represent a dataset using an approximate compact representation. Numerous algorithms for sketching have been proposed, including Count-Min (CM) [33], Count-Min with conservative update (CM-CU) [71], and Augmented sketching [135]. However, due to the skewness and the dynamic nature of data in a stream, these sketching methods are inefficient when applied to real data streams [34, 172], since there is often a need to estimate the frequency when considering the weight of data in a stream.

In order to preserve the characteristics of the data, a sketch needs to simulate the data distribution of a given dataset in a form that is as close as possible to the real distribution. Several algorithms have been proposed for sketching datasets [25, 83, 139, 84], one of the most well-known of which is locality-sensitive hashing (LSH) [83], a data-independent method that can guarantee the collision probability between similar data points. Examples of LSH-based methods are SimHash [139] and MinHash [25], which differ in terms of method of computing similarities. SimHash [139] uses the cosine distance to measure the similarity between data files, while in MinHash (or minwise hashing) [25], the Jaccard similarity is used to estimate the similarity between two sets. MinHash-based algorithms are generally efficient, and are more efficient than SimHash when the cosine similarity is used, although one drawback is that they consume a great deal of memory. Furthermore, minwise hashing uses a large number of permutations of the data, which in turn degrades the performance of the algorithm since minwise hashing is normally adopted in the context of binary or high-dimensional data. To address the issue of the cost of processing data, Weighted Minwise Hashing (WMH) [84] was developed, inspired by the idea of using densification and one permutation [106]. WMH uses a "rotation" scheme to densify the estimated sparse sketches using one permutation hashing that assigns new values to all the empty buckets. As a result, WMH greatly reduces the computational cost compared to MinHash-based schemes. It is also much simpler, significantly faster, and more memory efficient than the original approach, especially for very sparse datasets [30].

Histograms have long been seen as one of the most important statistical tools for providing information about a data distribution [131]. There are various algorithms for estimating the histograms generated from streaming

data. A few of these algorithms use a dynamic forgetting factor, including the most recent one, HistoSketch [168], which applies a Weighted Sampling method [84, 105] and the Hamming distance. Every time a new data item arrives in the stream, HistoSketch uses a constant decay factor to decrease the importance of outdated data in order to handle the issue of concept drift. Although several adaptive estimation approaches have been proposed for the efficient detection of concept drift [22, 142], a drawback of all of these algorithms is that they compute a decay factor at every observation, and consider data independently. This means that they decrease the importance of all outdated data by that constant factor, and then use the same factor on all data.

Part II

Pattern Discovery and Change Detection in Evolving Data

This part presents a study investigating the dependencies of data and proposes statistical hypotheses to detect changes. We explore the impact of data structures, correlations and dependencies between data features on the performance of various methods of event detection. The first chapter in this part introduces a new approach to solve the problem of detecting patterns from an unknown distribution. The next chapter describes an efficient algorithm for detecting both local and global changes in the structure of high utility patterns. Global changes are determined based on a statistical hypothesis of a distance measure between single patterns. The last chapter presents a method that utilizes the temporal dependencies between data in a statistical hypothesis model to detect changes in streaming data.

Chapter 3

Summarizing a Stream for High Utility Pattern Detection

The efficiency of an algorithm (e.g., for pattern detection) depends strongly on the characteristics of the data. Efficient methods are normally designed to work well with certain specific characteristics; however, in some cases, the characteristics of the data are unknown beforehand, and a robust solution needs to take into account the different and unknown distribution of data for avoiding bias. The work presented in this chapter addresses research questions RQ1 to RQ3 [44], i.e., *How can we design efficient methods to work well with a variety of different characteristics of data? How can we avoid bias in the dense and sparse characteristics of data? How can we construct efficient structures and simpler models with limited computing resources (i.e., memory)*?

3.1 Motivation

Utility-lists were introduced in the HUI-Miner [110] algorithm to discover high utility itemsets. HUI-Miner was shown to be up to 100 times faster than several state-of-the-art algorithms. In this approach, a utility-list is associated to each itemset, and utility-lists of itemsets are built without scanning the database by joining the utility-lists of some of their subsets. The algorithm can directly calculate the utility of itemsets and reduce the search space without having to maintain a set of candidates in memory or to repeatedly scan the database. The simplicity of the utility-list structure and the high performance of utility-list based algorithms have led to the development of numerous utility-list based algorithms for HUIM and variations of the HUIM problem such as closed high utility itemset mining [162, 38],

top-k high utility itemset mining [158, 100, 45], high utility sequential pattern mining [160], and on-shelf high utility itemset mining [59, 37], among others [57, 175, 56]. Although the introduction of the utility-list structure has been a breakthrough in the field of HUIM, the utility-list structure still has to be improved. In particular, it can be observed that the amount of memory required by utility-lists can be quite large.

Contributions

The main contributions of this chapter are fourfold as follows.

- A novel Utility-List Buffer structure is proposed. It is based on the principle of buffering utility-lists to decrease memory consumption. A Utility-List Buffer consists of multiple *segments*, which are reused to store utility-list information.
- An efficient join operation is designed to create utility-lists segments in a Utility-List Buffer in linear time, to decrease the time required for utility-list construction.
- An efficient algorithm named ULB-Miner (Utility-List Buffer Miner) is proposed to mine HUIs efficiently using the designed Utility-List Buffer structure and several implementation optimizations.
- An extensive experimental study is conducted in order to evaluate the efficiency of the proposed utility-list buffer structure and ULB-Miner algorithm on both sparse and dense datasets having various characteristics. In these experiments, the performance of ULB-Miner is compared with state-of-the-art algorithms with or without the novel utility-list buffer structure. Our results show that the proposed ULB-Miner algorithm outperforms the previous state-of-the-art utility-list based HUIM algorithms. Moreover, our experiments show that algorithms employing the novel structure are up to 10 time faster than when using standard utility-lists and consumes up to 6 times less memory. Also, the proposed technique performs quite well on both dense and sparse datasets.

Organization

The rest of this chapter is organized as follows. In Section 3.2, we briefly review the related literature. We define the problem of mining high utility itemsets and introduce the preliminaries of this problem in Section 3.3. In

Section 3.4, we present the novel utility-list buffer structure, its construction and join operations, and the ULB-Miner algorithm. In Section 3.5, we report on and discuss the experimental results. Finally, in Section 3.6, we conclude the chapter.

3.2 Related Work

Many algorithms have been designed to discover HUIs [112, 7, 110, 157, 57, 152]. To reduce the search space and mine HUIs efficiently, these algorithms have included various methods to overestimate the utility of itemsets. Several high utility itemset mining algorithms discover high utility itemsets using the TWU measure and a two-phase approach. This includes algorithms such as Two-Phase [112], UP-Growth+ [157], PB [98] and BAHUI [152]. In the first phase, the algorithms overestimate the utility of itemsets to obtain a set of candidate HUIs using the TWU measure and other strategies. Then, in the second phase, they scan the database again to calculate the utility of these candidates and filter those that are not HUIs. Although these algorithms are complete as they can find the whole set of HUIs, the two-phase approach can lead to considering and maintaining a very large number of candidate itemsets in memory. The cost of scanning the database for each itemset in the second phase to calculate their utility is also very costly. As a result, these algorithms can be slow and consume a huge amount of memory.

In recent years, to avoid the drawbacks of the two-phase approach, algorithms have been proposed to mine high utility itemsets using a single phase. These algorithms can directly calculate the utility of itemsets in memory without having to repeatedly scan the database or maintain candidates in memory. Moreover, they utilize tighter upper-bounds and more efficient strategies to reduce the search space, compared to two-phase algorithms. The concept of single phase algorithm was introduced in the HUI-Miner algorithm [110] by using a novel structure called utility-list. This structure stores all the information needed to calculate the utility of itemsets and reduce the search space, without repeatedly scanning the database. To discover HUIs, the HUI-Miner algorithm first constructs a utility-list for each item by scanning the database. Then, HUI-Miner recursively builds utility-lists of larger itemsets by joining the utility-lists of some of their subsets, i.e., without scanning the database again. The HUI-Miner algorithm is a complete algorithm as it can enumerate all high utility itemsets with their utility values using the utility-list structure. In terms of performance, it was shown that HUI-Miner outperforms the state-of-the-art two-phase HUIM algorithms [110]. Nonetheless, the performance of HUI-Miner can

still be improved. An important observation is that the join operation for obtaining the utility-lists of itemsets is costly in terms of runtime. To reduce the number of join operations performed by HUI-Miner, Fournier et al. designed the Faster High-Utility Itemset Mining (FHM) algorithm [57]. FHM applies a strategy to eliminate low utility itemsets using information about item co-occurrences. For each itemset eliminated using this strategy, the join operation does not need to be applied, thus reducing the execution time. It was shown that this pruning strategy can greatly reduce the number of join operations, and that FHM [57] can be up to six times faster than HUI-Miner.

Creating the utility-lists of itemsets using the join is costly. It requires a significant amount of memory, since an algorithm has to maintain many utility-lists in memory during the search for HUIs. Moreover, in terms of execution time, the complexity of building a utility-list is also high [57]. In general, it requires to join three utility-lists of smaller itemsets. Recently, improved versions of the HUI-Miner algorithm called HUP-Miner [93] and FHM [57] have been proposed by introducing additional search space pruning strategies and optimizations. It was shown that these algorithms can be up to 6 times faster than HUI-Miner and are the state-of-the-art algorithms for HUIM. Although some algorithms [57, 162, 158, 45] have introduced strategies to reduce the number of join operations, this operation is repeatedly performed to mine high utility itemsets, and this high cost has a negative impact on the performance, especially when the number of items is huge or a database contains long transactions. Hence, joining utility-lists remains the main performance bottleneck in terms of execution time, and storing utility-lists remains the main issue in terms of memory consumption [57].

3.3 Preliminaries

Definition 15 (Utility of an item in a transaction)**.** *Let there be an item i and a transaction T_d such that $i \in T_d$. The utility of* i *in T_d is the product of the internal utility (purchase quantity) of item* i *in T_d by the external utility (unit profit) of i, that is $u(i, T_d) = q(i, T_d) \times p(i)$.*

For example, in the database of Table 2.1, $u(a, T_1) = 1 \times 5 = 1$, and $u(c, T_1) = 1 \times 1 = 1$.

Definition 16 (Utility of an itemset in a transaction)**.** *For an itemset* X *and a transaction T_d, the utility of X in T_d is a positive number defined as $u(X, T_d) = \sum_{i \in X} u(i, T_d)$.*

For instance, consider the utility of itemset ac in transaction T_3 for the database of Table 2.1. The utility of ac in T_3 is calculated as $u(ac, T_3) = 1 \times 5 + 2 \times 2 = 9$. Similarly, the utility of bc in transaction T_2 is calculated as $u(bc, T_2) = 4 \times 2 + 1 \times 3 = 11$.

Definition 17 (Transaction utility and total utility)**.** *The utility of a transaction T_d is the sum of the utilities of items appearing in that transaction, that is $TU(T_d) = u(T_d, T_d)$. The total utility of a database D is the sum of the utilities of all transactions, that is $TUD(D) = \sum_{T_d \in D} TU(T_d, T_d)$.*

For example, in Table 2.1, $TU(T_1) = 8$, $TU(T_2) = 27$, $TU(T_3) = 30$, $TU(T_4) = 20$, $TU(T_5) = 11$. The total utility of database D is $TUD(D) = (TU(T_1) + TU(T_2) + TU(T_3) + TU(T_4) + TU(T_5)) = 8 + 27 + 30 + 20 + 11 = 96$.

Definition 18 (Utility and relative utility of an itemset)**.** *Let there be a database D and an itemset X. The utility of X in D is defined as $u(X) = \sum_{X \subseteq T_d \wedge T_d \in D} u(X, T_d)$. The relative utility of X in D is defined as $ru(X) = u(X)/TUD(D)$.*

For instance, the utility of the itemset ac in the database of Table 2.1 is $u(ac) = u(ac, T_1) + u(ac, T_2) + u(ac, T_3) = 6 + 16 + 6 = 28$, while the relative utility of ac in that database is $ru(ac) = 28/96 = 0.29$.

Definition 19 (Low utility itemset and high utility itemset)**.** *Let the minimum utility threshold (abbreviated as minutil) be a positive number specified by the user such that $0 < minutil < TUD(D)$. Consider an itemset X. It is said to be a* high utility itemset *(HUI) if its utility is no less than minutil. Otherwise, X is said to be a* low utility itemset*.*

Definition 20 (High utility itemset mining)**.** *Given a minimum utility threshold minutil and a database D, the problem of high utility itemset mining is to enumerate all high utility itemsets appearing in D.*

Note that the problem of high utility itemset mining can also be defined in terms of the relative utility of itemsets. Given a relative minimum utility threshold $r\_minutil = minutil/TUD(D)$, an itemset X is a high utility itemset if and only if $ru(X) \geq r\_minutil$.

The TWU measure was introduced and used as an upper-bound on the utility. The TWU measure is defined as follows and has the following important properties [112].

Definition 21 (Transaction-weighted Utilization)**.** *Let there be an itemset X and a database D. The Transaction-weighted Utilization(TWU) [112] of* X *in D is denoted as $TWU(X)$ and defined as:*

$$TWU(X) = \sum_{T_d \in D \wedge X \subseteq T_d} TU(T_d). \tag{3.1}$$

Property 5 (Overestimation [112])**.** *The utility of an itemset X is less than or equal to its TWU, that is $TWU(X) \geq u(X)$.*

For instance, consider the transactions T_1, T_2, and T_3 in the database of the running example. Their TWU values are 8, 27, and 30, respectively. The TWU of the item a is calculated as $TWU(a) = TU(T_1) + TU(T_2) + TU(T_3) = 8 + 27 + 30 = 65$. The following property has been used by several HUIM algorithms to reduce the search space.

Property 6 (Search space reduction using the TWU [112])**.** *For an itemset X, if $TWU(X) < minutil$, it follows that X and its supersets are low utility itemsets.*

For example, the transaction-weighted utilization of item f is $TWU(f) = TU(T_3) = 5 + 4 + 1 + 12 + 3 + 5 = 30$. Table 3.1 shows the transaction utilities of all transactions in D and the TWU values of each item.

Table 3.1: The TU and TWU values of transactions for the running example

Item Name	a	b	c	d	e	f	g
TWU	65	61	96	58	88	30	38
TID	T_1	T_2	T_3	T_4	T_5		
TU	8	27	30	20	11		

The proposed algorithm relies on the novel utility-list buffer structure inspired by the utility-list structure [110] to mine high utility itemsets in a single phase. The next paragraphs present definition of the utility-list structure and its key properties [110].

Definition 22 (Utility-list)**.** *Let $\succ$ be a total order on items from I, and X be an itemset appearing in a database D. The* utility-list *of X is denoted as $ul(X)$. It contains a tuple $(tid, iutil, rutil)$ for each transaction T_{tid} where X appears ($X \subseteq T_{tid}$). The iutil element of a tuple for a transaction T_{tid} stores the utility of X in the transaction T_{tid}, i.e., $u(X, T_{tid})$. The rutil element of a tuple stores the value $\sum_{i \in T_{tid} \wedge i \succ x \forall x \in X} u(i, T_{tid})$, which is called the remaining utility of X [110].*

Example 2. *In the running example, the utility-list of the item a is {(T1, 5, 3)(T2, 10, 17)(T3, 5, 25)}. The utility-list of the item e is {(T2, 6, 5)(T3, 3, 5)(T4, 3, 0)}. The utility-list of the itemset ae is {(T2, 16, 5),(T3, 8, 5)}.*

Two important properties of utility-lists have been proposed to determine the utility of an itemset and to reduce the search space, respectively [110].

Property 7 (Calculating the utility using the sum of iutil values [110])**.** *The utility of an itemset X (denoted as u(X)) can be calculated by performing the sum of the iutil values in the utility-list ul(X). If that sum is less than the minutil threshold, X is a low utility itemset. Otherwise, it is a high utility itemset [110].*

Property 8 (Pruning using an utility list's iutil and rutil values [110])**.** *Let X and Y be two itemsets. It is said that Y is an* extension *of X if Y can be obtained by adding an item c to X, where $c \succ i$, $\forall i \in X$. The sum of the iutil and rutil values in the utility-list ul(X) is an upper-bound on the utility of Y and any other transitive extension of X. As a consequence, if this sum is less than the minutil threshold, it follows that any itemset that is a transitive extension of X must be a low utility itemset, and thus be pruned.*

3.4 Model and Solution

This section presents our method to address the above discussed drawback of the state-of-the-art methods.

3.4.1 The Proposed Utility-list Buffer Method

As proposed in the HUI-Miner algorithm [110], the utility-list of an itemset Pxy can be constructed without accessing the database by joining the utility-lists of some subsets of Pxy. For instance, consider some itemsets Px, Py, and Pxy, where Px and Py are extensions of an itemset P obtained by appending an item x and an item y, respectively. To build the utility-list of the itemset Pxy, Algorithm 1 [110] is applied. The algorithm first considers each element in the utility-list $ul(x)$. For each such element, the algorithm verifies if there exists an element having the same transaction identifier in $ul(y)$. If such an element is found, the algorithm applies a binary search on the utility-list of the itemset P to check if an element in the utility-list of P has the same transaction identifier. The time complexity of this comparison of utility-lists is $\mathcal{O}(m \log nz)$, where m, n, and z are

the number of entries in $ul(x)$, $ul(y)$, and $ul(P)$, respectively. In terms of space complexity, a utility-list has a size proportional to $\mathcal{O}(n)$ in the worst case, where n is the number of transactions. The worst case occurs when a utility-list has an entry for each transaction of the database. The overall worst-case time complexity is thus roughly $\mathcal{O}(n^3)$.

Algorithm 1 The traditional utility-list construction procedure

Input:
ul(P) : the utility-list of itemset P;
ul(Px): the utility-list of itemset Px;
ul(Py): the utility-list of itemset Py;
Output: *ul(Pxy)*: the utility-list of itemset Pxy;

```
1:  ul(Pxy) = NULL;
2:  for each (tuple ex ∈ ul(Px)) do
3:      if (∃ey ∈ ul(Py) and ex.tid=ey.tid) then
4:          if (ul(P) is not empty) then
5:              Search element e ∈ ul(P) such that e.tid = ex.tid;
6:              exy ⟵ (ex.tid; ex.iutil + ey.iutil - e.iutil; ey.rutil);
7:          else
8:              exy ⟵ (ex.tid; ex.iutil + ey.iutil; ey.rutil);
9:          ul(Pxy) ⟵ ul(Pxy) ∪ exy;
10: return ul(Pxy);
```

The proposed method is based on the following observations. Joining utility-lists is costly both in terms of runtime and memory consumption. In utility-list-based algorithms, memory has to be allocated to store each utility-list. Since millions of itemsets are often considered by HUI (High Utility Itemset) mining algorithms, the memory used for storing utility-lists can be quite large. Moreover, because utility-lists can contain many entries, the time requires for allocating and reusing memory for utility-lists can be quite important. In addition, a related issue is that a utility-list can be kept in memory during a long period of time by utility-list-based algorithms, even if the corresponding itemset is identified as not being a HUI and/or is not extended by the search procedure to find HUIs. This can lead to high peaks of memory usage. In conclusion, there is an important issue with how memory is managed by the state-of-the-art utility-list-based algorithms. Our experimental evaluation in Section 3.5 will also show this in more details.

To address this issue, this section proposes a data structure named

utility-list buffer, designed to both quickly access information stored in utility-lists and more efficiently manage the memory used for storing the information of utility-lists. The proposed utility-list buffer structure is designed for replacing traditional utility-lists in any utility-list-based algorithms. As it will be shown in the experimental evaluation, using the utility-list buffer structure leads to considerably lower memory usage and faster execution times for utility-list-based algorithms.

3.4.2 The Utility-list Buffer Structure

The utility-list buffer structure is proposed to tackle the aforementioned limitations of one-phase utility-list-based algorithms for mining high utility itemsets. The utility-list buffer structure is introduced by the following definitions and properties. Then, an example will be given to illustrate the definitions.

Definition 23 (Utility-list buffer structure)**.** *Let I be the set of items in a database D. Let Tid_D be the set of transaction identifiers in the database D. The utility-list buffer structure for the database D is denoted as* UTLBuf*. The structure is designed like a memory pipeline to store information about itemsets that would be normally stored in their utility-lists. The utility-list buffer of a database stores a set of tuples of the form $(tid \in Tid_D, iutil \in \mathbb{R}, rutil \in \mathbb{R})$. These tuples called data segments, which store the tuples normally contained in traditional utility-lists. To quickly access the information stored in the utility-list buffer, a set of index segments are created, where an index segment* SUL*(X) indicates where the information about an itemset X is stored in the utility-buffer. Index segments allow fast accessibility of the data stored in the utility-buffer and are described next.*

Definition 24 (Summary of Utility-list)**.** *The index segment of an itemset X in a database D, also called the summary of utility-list of itemset X, is denoted as* SUL(X)*. It is defined as a tuple having the form (X,* StartPos*,* EndPos*,* SumIutil*,* SumRutil*). The* SumIutil *element stores the sum of the iutil values in $ul(X)$, that is $\sum ul(X).iutil$. The* SumRutil *element stores the sum of rutil values in the utility-list of X, that is $\sum ul(X).rutil$. The* StartPos *and* EndPos *elements respectively indicate the start index and end index of the data segments in the utility-list buffer structure* UTLBuf*, where the information that would be normally contained in the utility-list of X is stored.*

Definition 25 (Summary List)**.** *Let I be the set of items in a database D.*

A structure called Summary List *is further defined. It is a memory pipeline denoted as* SULs_D*, and defined as* SUL$s_D = \{SUL(X), X \subseteq I\}$.

The proposed utility-list buffer structure is used as follows by the proposed algorithm. When the algorithm considers an itemset X from the search space as a potential HUI and as an itemset that could be extended to find other HUIs, the algorithm stores the utility-list of X in the *UTL-Buf* structure by temporally inserting its information in the data segments of *UTLBuf* from the *StartPos* to *EndPos* positions. Then, when needed, the algorithm accesses this information by reading the values in the *UTL-Buf* from the *StartPos* to *EndPos* positions. Thanks to the utility-buffer structure, data can be quickly accessed. For efficient memory management, the temporary memory that is allocated for an itemset X in the *UTLBuf* structure is reused for storing the utility-lists of other itemsets when it is found that the utility-list of the itemset X is not needed anymore by the search process. In this case, the memory is recalled and reused for other candidate itemsets (this idea will be described in more details in Subsection 4.4).

In terms of implementation, we implement the proposed structures as follows. Four array lists are created, named *TIDs*, *Iutils*, *Rutils* and *SULs*. The three first lists store the information of the *UTLBuf* structure, and the fourth list is the *SULs* structure. These lists are initialized as empty and their size is increased when they are full, and more space is needed. Lists are used for storing the utility-lists of itemsets, and when the utility-list of an itemset is not needed, the memory is reused to store other utility-lists. This reduces the time for allocating memory and the overall memory usage for mining HUIs.

The proposed algorithm first creates the utility-list of all single items according to the total order $\succ$ by performing a database scan. For example, consider the utility-list of the item f. In transaction T_3, we have that $u(f, T_3) = 5$ and $ru(f, T_3) = 25$. The item f only appears in the transaction T_3. Hence, the summary of f is stored in the *SULs* list and contains the following information: the item is f, its start position index in the lists is 0, its information ends at position index 1, the sum of its utilities is 5, and the sum of its remaining utilities is 25. In the following, we use a notation $SULi$ to present an item of *SULs* for short.

$$SULi = \begin{pmatrix} Item \\ StartPos \\ EndPos \\ SumIutil \\ SumRutil \end{pmatrix} \tag{3.2}$$

The state of the utility-list buffer after inserting the item f is depicted in Figure 3.1. Thereafter, the other items are inserted in the same manner. The resulting state of the utility-list buffer is depicted in Figure 3.2, the first three rows are lists of TIDs, Iutils, and Rutils. In this figure, it can be seen that a utility-list segment is used for each item. Accessing a utility-list stored in the utility-buffer is efficient thanks to the *SULs* structure. For example, assume that the algorithm is currently processing the itemset $X = \{a\}$. To access its utility-list, the summary information of $\{a\}$ is obtained from the *SULs*. After the summary information of $\{a\}$ is obtained, its utility-list $UL(\{a\})$ is read in *UTLBuf* from the $SULs(\{a\}).StartPos$ to $SULs(\{a\}).EndPos$ positions (in red color in Figure 3.2).

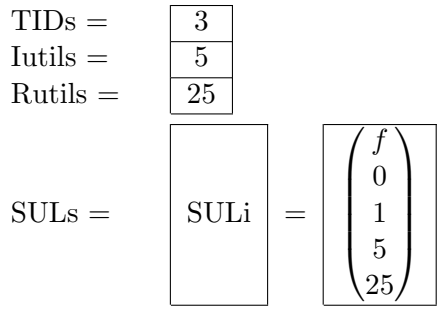


Figure 3.1: The utility-list buffer after inserting the item f

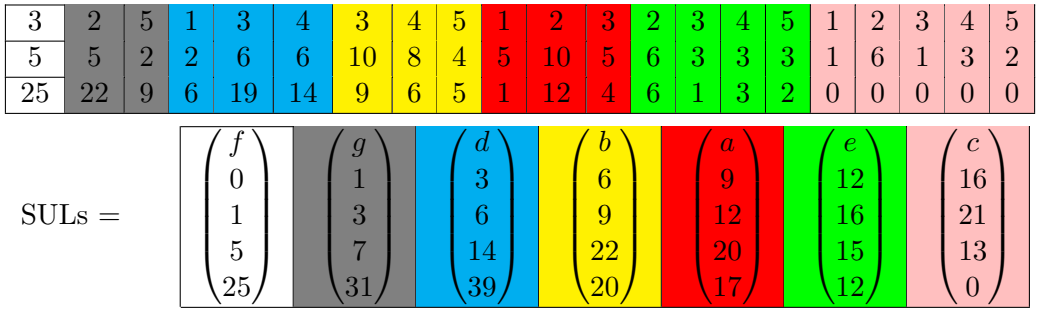


Figure 3.2: The utility-list buffer after inserting all single items

3.4.3 An Efficient Utility-list Segment Construction Method

The previous subsection has explained how the proposed data structures are used to store the utility-lists of itemsets containing a single item. This subsection explains the more general case where itemsets can have two or

more items.

As it has been pointed out in traditional utility-list-based algorithms [110, 57], the utility-list of a 2-itemset xy can be constructed without scanning the database by joining (intersecting) the utility-lists of its items x and y. Moreover, the utility-list of any k-itemset $\{i_1 \dots i_{k-1} i_k\}$ ($k \geq 3$) can be obtained by intersecting the utility-lists of three itemsets: $\{i_1 \dots i_{k-2} i_{k-1}\}$, $\{i_1 \dots i_{k-2}\}$ and $\{i_1 \dots i_{k-2} i_k\}$. The basic procedure for intersecting utility-lists was proposed in the HUI-Miner algorithm [110]. This procedure is given in Algorithm 1, where the utility-list of an itemset Pxy is built by intersecting the utility-lists of the itemsets Px, Py, and P. P is the prefix itemset, x and y are items. For each element in the utility list $ul(x)$, the procedure checks if an element has the same transaction identifier in the utility-list $ul(y)$. If yes, then a binary search is performed on the utility-list of P to find an element with the same transaction identifier. Hence, the time complexity of this procedure is $\mathcal{O}(sxlog(sy))$, where sx and sy are respectively the number of entries in $ul(x)$ and $ul(y)$.

Although this algorithm is useful for constructing utility-lists, it cannot be directly applied to utility-lists stored in the proposed utility-buffer structure. Thus, an adapted utility-list segment construction procedure is proposed and depicted in Algorithm 2. This procedure constructs a utility-list in the next free data segments of the utility-list buffer and updates the Summary List $SULs$ structure to allow the quick retrieval of the utility-list from the buffer when needed.

Since transaction identifiers (Tids) in utility-lists are ordered in ascending order, an efficient way of identifying transactions that are common to two utility-lists $ul(x)$ and $ul(y)$ are to read the two utility-lists at the same time by reading the Tids sequentially in each utility-list. The complexity of this search method is $\mathcal{O}(m + n)$, which is less than $\mathcal{O}(mlogn)$ for the basic utility-list construction method. Based on this observation, we introduce an improved construction procedure named ULB-Construct, and it is presented in Algorithm 3.

3.4.4 High Utility Itemset Miner Employing the Utility-list Buffer

Having presented the proposed utility-buffer structure and how utility-lists are constructed and stored in that structure, this subsection proposes a novel algorithm named ULB-Miner for discovering all high utility itemsets using that structure.

After constructing the initial utility-list buffer from an input database,

Algorithm 2 The basic utility-list segment construction procedure

Input:
PPosition : the *StartPos* of itemset P;
PxPosition: the *StartPos* of itemset Px;
PyPosition: the *StartPos* of itemset Py;
Output: *PxyPosition*: the *StartPos* of itemset Pxy;

1: $PxyPosition = NULL$;
2: $countX = SULs(Px).EndPos$ - $SULs(Px).StartPos$;
3: **for** ($i = 0$; $i < countX$; i++) **do**
4: **if** ($\exists j \in [SULs(Py).StartPos, SULs(Py).EndPos]$ and $TIDs[SULs(Px).StartPos+i]=TIDs[j]$) **then**
5: **if** ($PPosition \geq 0$) **then**
6: Search index $p \in [SULs(P).StartPos, SULs(P).EndPos]$ such that $TIDs[p] = TIDs[SULs(Px).StartPos+i]$;
7: $TIDs[PxyPosition + p] = TIDs[PxPosition + i]$;
8: $Iutils[PxyPosition + p] = Iutils[PxPostion + i] + Iutils[PyPostion + j]$ - $Iutils[PPosition + p]$;
9: $Rutils[PxyPosition + p] = Rutils[PyPostion + j]$;
10: **else**
11: $TIDs[PxyPosition + p] = TIDs[PxPosition + i]$;
12: $Iutils[PxyPosition + p] = Iutils[PxPostion + i] + Iutils[PyPostion + j]$;
13: $Rutils[PxyPosition + p] = Rutils[PyPostion + j]$;
14: Update *SULs* of Pxy;
15: **return** *PxyPosition*;

the algorithm can efficiently mine all high utility itemsets by employing the utility-list buffer. The proposed approach for mining HUIs is inspired by the HUI-Miner [110] and FHM [57] algorithms, but adapted to work with the novel utility-buffer structure. In particular, it integrates the novel ULB-construct procedure, described in the previous subsection, that constructs utility-list segments in linear time. The main procedure of ULB-Miner is shown in Algorithm 4. The input is a transaction database D and the *minutil* threshold, and the output is the high utility-itemsets. The main procedure performs the following steps. The algorithm first scans the database to calculate the TWU of all items (line 1). Then, based on these TWU values, the set I^* is created, which contains all items having a TWU greater than or equal to the *minutil* threshold (line 2). The TWU values of items are used to build a total order $\succ$ on items, which is the ascending

Algorithm 3 ULB-Construct: the efficient utility-list segment construction procedure

Input:
UTLBuf, SULs;
Itemsets P, Px; Py;
Output: Updated *UTLBuf, SULs* with itemset Pxy

1: Let $PPnt$, $PxPnt$, $PyPnt$ are three pointers that initially point to $UTLBuf$ at positions $SULs(P).StartPos$, $SULs(Px).StartPos$, and $SULs(Py).StartPos$, respectively.
2: **while** ($PxPnt$ not reach $SULs(Px).EndPos$ **and** $PyPnt$ not reach $SULs(Py).EndPos$) **do**
3: **if** ($TIDs[PxPnt] < TIDs[PyPnt]$) **then**
4: Shift $PxPnt$ to the right by 1;
5: **else if** ($TIDs[posX] > TIDs[posY]$) **then**
6: Shift $PyPnt$ to the right by 1;
7: **else**
8: **if** ($SULs[P]$ is not $NULL$) **then**
9: **while** ($PPnt$ not reach $SULs(P).EndPos$ **and** $TIDs[PPnt] \neq TIDs[PxPnt]$) **do**
10: Shift $PPnt$ to the right by 1;
11: $UTLBuf.TIDs[TIDs.count{+}{+}] = TIDs[PxPnt]$;
12: $UTLBuf.Iutils[Iutils.count{+}{+}] = Iutils[PxPnt] + Iutils[PyPnt] - Iutils[PPnt]$;
13: $UTLBuf.Rutils[Rutils.count{+}{+}] = Rutils[PyPnt]$;
14: Shift both $PxPnt$ and $PyPnt$ to the right by 1;
15: Update $SULs[Pxy]$;

order of TWU values (line 3). The algorithm then scans the database again (line 4) to reorder items in transactions according to that total order. At the same time, the utility-list buffer of all single items $i \in I$ and the Estimated Utility Co-occurrence Structure (EUCS) [57] are built. The EUCS stores the TWU values of all pairs of items. It will be discussed in more details in the next subsection. After that the algorithm starts a recursive depth-first search by invoking the Search procedure (line 5).

The Search procedure is presented in Algorithm 5. It performs the following operations. For each extension Px of P, if the sum of the *iutil* values of Px is no less than *minutil*, then Px is a high utility itemset based on Property 7. Hence, the itemset Px is output (lines 2-4). Then, if the sum of the *SumIutil* and *SumRutil* values of Px is greater than or equal

Algorithm 4 The ULB-Miner algorithm

Input:
D: a transaction database;
minutil: a user-defined threshold;
Output: The set of high utility itemsets;

1: Scan D to calculate the TWU of single items;
2: Let I^* be the list of single items i such that $TWU(i) \geq minutil$;
3: Let $\succ$ be the total order of TWU ascending values on I^*;
4: Scan D again to build the initial *utility-list buffer* $UTLBuf$, and $SULs$ of item $i \in I^*$ and build the $EUCS$;
5: **Search** ($\emptyset$, I^*, *minutil*, $EUCS$, $UTLBuf$, $SULs$);

Algorithm 5 The Search Procedure

Input:
P: an itemset;
ExtensionsOfP: a set of extensions of P;
minutil: the user-specified utility threshold;
$EUCS$: the $EUCS$ structure;
utility-list buffer $UTLBuf$: the utility-list buffer structure;
SULs: The summary list;
Output: The set of high utility itemsets;

1: **for each** *itemset* $Px \in ExtensionsOfP$ **do**
2: **if** ($SULs(Px).SumIutil \geq minutil$) **then**
3: output Px;
4: **if** ($SULs(Px).SumIutil + SULs(Px).SumRutil \geq minutil$) **then**
5: $ExtensionsOfPx \leftarrow \emptyset$;
6: **for** (**each** *itemset* $Py \in ExtensionsOfP$ *such that* $y \succ x$) **do**
7: **if** ($\exists(x, y, c) \in EUCS$ such that $c \geq minutil$) **then**
8: $Pxy \leftarrow Px \cup Py$;
9: **if** (**ULBReusingMemory-Construct**($UTLBuf$, $SULs$, P, Px, Py) **then**
10: $ExtensionsOfPx \leftarrow ExtensionsOfPx \cup Pxy$;
11: **Search** (Px, $ExtensionsOfPx$, *minutil*, $EUCS$, $UTLBuf$, $SULs$);

to *minutil*, the extensions of Px are considered for further exploration (line 5), based on Property 8. This process is done by combining Px with each extension Py of P such that $y \succ x$ to produce a larger itemset itemset Pxy (line 8). The utility-list segment of Pxy is then constructed by calling an

improved version of the ULB-Construct procedure, which will be presented in the next subsection (Algorithm 6). This procedure joins the utility-list segments of P, Px and Py (line 9). Then, a recursive call to the Search procedure with Pxy is done to calculate the utility of that itemset and recursively explore its extensions (line 11).

As other utility-list based algorithms for mining high utility itemsets [110, 57], the Search procedure starts from single items and then recursively explores the search space of itemsets by appending single items, while reducing the search space using Properties 7 and 8. It thus can be easily seen that this process is correct and complete to discover all high utility itemsets.

3.4.5 Implementation Optimizations

The Estimated Utility Co-Occurrence Structure (EUCS) [57] is a very useful structure for pruning the search space. The EUCS has been designed to avoid performing join operations to construct utility-lists of itemsets when some specific conditions are met. It was demonstrated that this structure and the corresponding Estimated Utility Co-occurrence Pruning (EUCP) strategy can considerably reduce the number of join operations for HUI mining using utility-lists. Hence in the proposed framework, this structure and its search space pruning strategy are also used to reduce the search space and increase the performance of the proposed algorithm. This structure is used in line 7 of Algorithm 5.

Moreover, to obtain better performance for utility-list construction, an approach is proposed in [45] for abandoning utility-list construction early named EA (Early Abandoning) strategy. This strategy and its stopping criterion are designed and employed during the construction of utility-lists of all candidate itemsets to avoid completely constructing utility-lists. The utility-list construction process is immediately stopped if a specific condition is met. This strategy can reduce the runtime and memory consumption of the algorithms considerably. Therefore, EA is also implemented in the *UTLBuf* framework. Detail of how the EA strategy is implemented in the *UTLBuf* framework is shown in Algorithm 6 using the variable *EACriterion*.

Finally, a novel optimization is proposed to reuse memory in the utility-buffer. It is based on the following observation. In utility-list based algorithms, the utility-list of an itemset containing more than one item is constructed by intersecting the utility-lists of some of its subsets. For instance, the utility-list of an itemset Pxy, $ul(Pxy)$, can be obtained by intersecting the utility-lists of itemsets P, Px and Py. However, after constructing the utility-list of Pxy, it is possible that Pxy is considered to not be a HUI according to Property 7, and also to not be useful for generating larger

HUIs according to Property 8. Hence, the memory allocated for storing the utility-list of Pxy is wasted and could be reused for storing other utility-list(s). This is a serious problem because the construction of utility-lists is a process that is repeatedly performed by the search procedure. To save memory, we propose the following strategy for memory re-utilization. If an itemset Pxy is not a candidate for exploring the search space, then the memory allocated for storing its utility-list will be recalled and reused for the next potential candidates that will be considered by the search procedure. All the memory used for Pxy will be reused and new memory is only allocated when the utility-buffer is full. The pseudo-code of the improved ULB-Construct procedure integrating this strategy is shown in Algorithm 6.

3.4.6 An Illustrative Example

To give a better understanding of how the proposed ULB-Miner algorithm works, and at the same time show the benefits of the designed utility-list buffer structure, this subsection provides a detailed example. In this example, ULB-Miner is applied on the database D shown in Table 2.1 with $minutil = 35$ and the external utilities of items are shown in Table 2.2.

Step 1. The database D is scanned to calculate the TWU of single items. The resulting TWU values of items are shown in Table 3.1. The set of single items I^* sorted by ascending TWU values and having $TWU \geq 35$ is $\{g, d, b, a, e, c\}$. Item f is dismissed because $TWU(f) = 30 < 35 = minutil$.

Step 2. The initial $UTLBuf$ and $SULs$ structures for items in I^* are constructed. The result is shown in Figure 3.3.

Step 3. The Search procedure is invoked to perform the recursive search.

(a) The procedure explores the search space starting from item g. Because $SULs(g).SumIutil = 7 < minutil = 35$, g is not a high utility itemset. But $SULs(g).SumIutil + SULs(g).SumRutil = 7 + 31 = 38 > minutil$. Thus, extensions of g should be considered as potential high utility itemsets.

(b) The algorithm appends each item y to g such that $y \succ g$ and $y \in I^*$ to form larger itemsets. The algorithm first considers appending d to g to form the larger itemset gd. Because g and d never appear together (an empty utility-list is constructed), the itemset gd is not further considered.

Algorithm 6 ULBReusingMemory-Construct: Construction procedure for reusing memory

Input: *UTLBuf*, *SULs*, Itemsets P, Px, and Py;
Output: Updated *UTLBuf*, *SULs* with itemset Pxy

1: Let $PPnt$, $PxPnt$, $PyPnt$ are three pointers that initially point to *UTLBuf* at positions $SULs(P).StartPos$, $SULs(Px).StartPos$, and $SULs(Py).StartPos$, respectively.
2: Let $EACriterion = SULs[Px].SumIutil + SULs[Py].SumIutil + SULs[Px].SumRutil + SULs[Py].SumRutil$
3: Let $insertionPosition = SULs.Last.endPos$;
4: **while** ($PxPnt$ not reach $SULs(Px).EndPos$ **and** $PyPnt$ not reach $SULs(Py).EndPos$) **do**
5: **if** ($TIDs[PxPnt] < TIDs[PyPnt]$) **then**
6: Shift $PxPnt$ to the right by 1;
7: Substract $EACriterion$ by ($Iutils[PxPnt] + Rutils[PxPnt]$)
8: **else if** ($TIDs[posX] > TIDs[posY]$) **then**
9: Shift $PyPnt$ to the right by 1;
10: Substract $EACriterion$ by ($Iutils[PyPnt] + Rutils[PyPnt]$)
11: **else**
12: **if** ($SULs[P]$ is not $NULL$) **then**
13: **while** ($PPnt$ not reach $SULs(P).EndPos$ **and** $TIDs[PPnt] \neq TIDs[PxPnt]$) **do** Shift $PPnt$ to the right by 1;
14: **if** ($insertionPosition \geq UTLBuf.TIDs.size()$) **then**
15: $UTLBuf.TIDs[TIDs.count{+}{+}] = TIDs[PxPnt]$;
16: $UTLBuf.Iutils[Iutils.count{+}{+}] = Iutils[PxPnt] + Iutils[PyPnt]$ - $Iutils[PPnt]$;
17: $UTLBuf.Rutils[Rutils.count{+}{+}] = Rutils[PyPnt]$;
18: **else**
19: $insertionPosition{+}{+}$ *//Reused Memory*
20: $UTLBuf.TIDs[insertionPosition] = TIDs[PxPnt]$;
21: $UTLBuf.Iutils[insertionPosition] = Iutils[PxPnt] + Iutils[PyPnt]$ - $Iutils[PPnt]$;
22: $UTLBuf.Rutils[insertionPosition] = Rutils[PyPnt]$;
23: Shift both $PxPnt$ and $PyPnt$ to the right by 1;
24: **if** ($EACriterion < minutil$) **then return** *false*;
25: Update $SULs[Pxy]$;
26: **return** *true*;

(c) Then, the algorithm considers appending b to g to create the itemset gb. The utility-list of bd is inserted into the utility-buffer $UTLBuf$ as shown in Figure 3.4 (cells filled with white color). The sum of the $Iutils$ and $RUtils$ values of gb is $6 + 5 = 11 < minutil$. Hence, the itemset gb is not considered as a candidate by the search procedure. Note that at this point, previous utility-list-based algorithms would allocate new memory for storing the utility-lists of the following candidates. The proposed method will instead reuse the memory allocated for the utility-list of gb for storing the utility-lists of the following candidates.

(d) The algorithm next considers the itemset ga. The state of the utility-list buffer $UTLBuf$ after inserting the utility-list of ga is shown in Figure 3.5. The sum of the $Iutils$ and $RUtils$ values of ga is $15 + 12 = 27 < minutil$. Thus, ga will not be considered by the search procedure to generate further extensions. This memory will be reused for storing the utility-lists of the following candidates.

(e) The following item e is appended to itemset g to form the itemset ge. The algorithm inserts the utility-list of ge into the utility-buffer. The resulting state of the buffer is shown in Figure 3.6. The itemset ge is not extended by the search procedure because the sum of the $Iutils$ and $RUtils$ values of ge is $11 + 5 + 6 + 2 = 24 < minutil$. This memory will thus be reused to store the utility-lists of the following candidates.

(f) Then, the item c is appended to g to create the itemset gc. The state of the utility-list buffer after inserting the utility-list of gc is shown in Figure 3.7. The itemset gc is also not a high utility itemset due to its low utility.

Step 4. The search for high utility itemsets is then continued with other items until no more itemsets can be generated. The result is the set of all high utility itemsets found in the dataset D. This set is $\{dbec : 40, dbe : 36\}$, where the number besides each itemset indicates its utility.

In the above example, the proposed algorithm relying on the novel utility-list buffer allocates only 2 entries in the utility-buffer for storing the utility-lists of extensions of the item g. Previous utility-list-based algorithms such as HUI-Miner and FHM would utilize 6 entries to store the

2	5	1	3	4	3	4	5	1	2	3	2	3	4	5	1	2	3	4	5
5	2	2	6	6	10	8	4	5	10	5	6	3	3	3	1	6	1	3	2
22	9	6	19	14	9	6	5	1	12	4	6	1	3	2	0	0	0	0	0

SULs =

g	d	b	a	e	c
0	2	5	8	11	15
2	5	8	11	15	20
7	20	16	20	15	13
31	33	20	17	12	0

Figure 3.3: The initial utility-list buffer: TIDs, Iutils, Rutils, and SULs

2	5	1	3	4	3	4	5	1	2	3	2	3	4	5	1	2	3	4	5	5
5	2	2	6	6	10	8	4	5	10	5	6	3	3	3	1	6	1	3	2	6
22	9	6	19	14	9	6	5	1	12	4	6	1	3	2	0	0	0	0	0	5

Figure 3.4: The utility-list buffer after inserting the utility-list of gb

2	5	1	3	4	3	4	5	1	2	3	2	3	4	5	1	2	3	4	5	2
5	2	2	6	6	10	8	4	5	10	5	6	3	3	3	1	6	1	3	2	15
22	9	6	19	14	9	6	5	1	12	4	6	1	3	2	0	0	0	0	0	12

Figure 3.5: The utility-list buffer after inserting the utility-list of ga

2	5	1	3	4	3	4	5	1	2	3	2	3	4	5	1	2	3	4	5	2	5
5	2	2	6	6	10	8	4	5	10	5	6	3	3	3	1	6	1	3	2	11	5
22	9	6	19	14	9	6	5	1	12	4	6	1	3	2	0	0	0	0	0	6	2

Figure 3.6: The utility-list buffer after inserting the utility-list of ge

2	5	1	3	4	3	4	5	1	2	3	2	3	4	5	1	2	3	4	5	2	5
5	2	2	6	6	10	8	4	5	10	5	6	3	3	3	1	6	1	3	2	11	0
22	9	6	19	14	9	6	5	1	12	4	6	1	3	2	0	0	0	0	0	4	0

Figure 3.7: The utility-list buffer after inserting the utility-list of gc

utility-lists, due to the lack of a mechanism for reusing memory. If we consider the full search space for the previous example, the proposed algorithm only needs 33 entries in the utility-buffer and reuses 39 times some existing entries to store utility-lists. This simple example shows that the proposed utility-list buffer structure is useful for mining high utility itemsets while reusing memory.

3.5 Evaluation

We performed a series of large scale experiments to evaluate the performance of the proposed ULB-Miner algorithm employing the designed utility-list buffer structure.

3.5.1 Experimental Setup

The algorithms were implemented by extending the SPMF open-source Java data mining library [53]. The source code was compiled using the J2SDK 1.7.0, and the memory measurements were done using the standard Java API. The experiments were run on a computer equipped with an Intel core i3 processor 2.4 GHz and 4 GB of RAM, running the Windows 7 operating system.

Table 3.2: Characteristics of the datasets

Dataset	#Transactions	#Distinct items	Avg. trans. length
Connect	67,557	129	43
Chainstore	1,112,949	46,086	7.2
Chess	3196	75	37
Foodmart	4141	1559	4.4
Kosarak	990,000	41,270	8.1
Retail	88,162	16,470	10.3

Datasets

Both real and synthetic datasets having varied characteristics were used in the experiments. These datasets are standard benchmark datasets used to evaluate HUIM algorithms. The characteristics of these datasets are described in Table 3.2, where #Transactions, #Distinct items and Avg. trans. length indicate the number of transactions, the number of distinct items and the average transaction length, respectively. These datasets were selected because they are standard benchmark datasets and they have varied characteristics.

We used two real-world customer transaction datasets named Chainstore[1] and Foodmart[2]. Chainstore is a very large dataset consisting of

[1] `http://cucis.ece.northwestern.edu/projects/DMS/MineBenchDownload.html`
[2] `https://www.microsoft.com/en-us/download/details.aspx?id=51958`

transactions from a Californian retail store, while Foodmart is a small dataset of customer transactions obtained from the Microsoft Food-Mart 2000 database. Retail[3] is a sparse dataset containing customer transactions from a Belgian retail store. Kosarak[4] is a very sparse dataset with moderately short transactions. Lastly, two dense datasets named Chess[5] and Connect[5] were used. Although these two datasets are not retail data, they are often used in the pattern mining literature as benchmark datasets to evaluate the performance on dense data. Chess is especially a quite challenging dataset for most mining algorithms because it contains many long itemsets. The Chainstore and Foodmart datasets already contain real unit profits and purchase quantities. For other datasets, external utilities of items are generated between 1 and 1000 by using a log-normal distribution and quantities of items are generated randomly between 1 and 5, as the settings of previous studies [110, 57].

3.5.2 Running Time

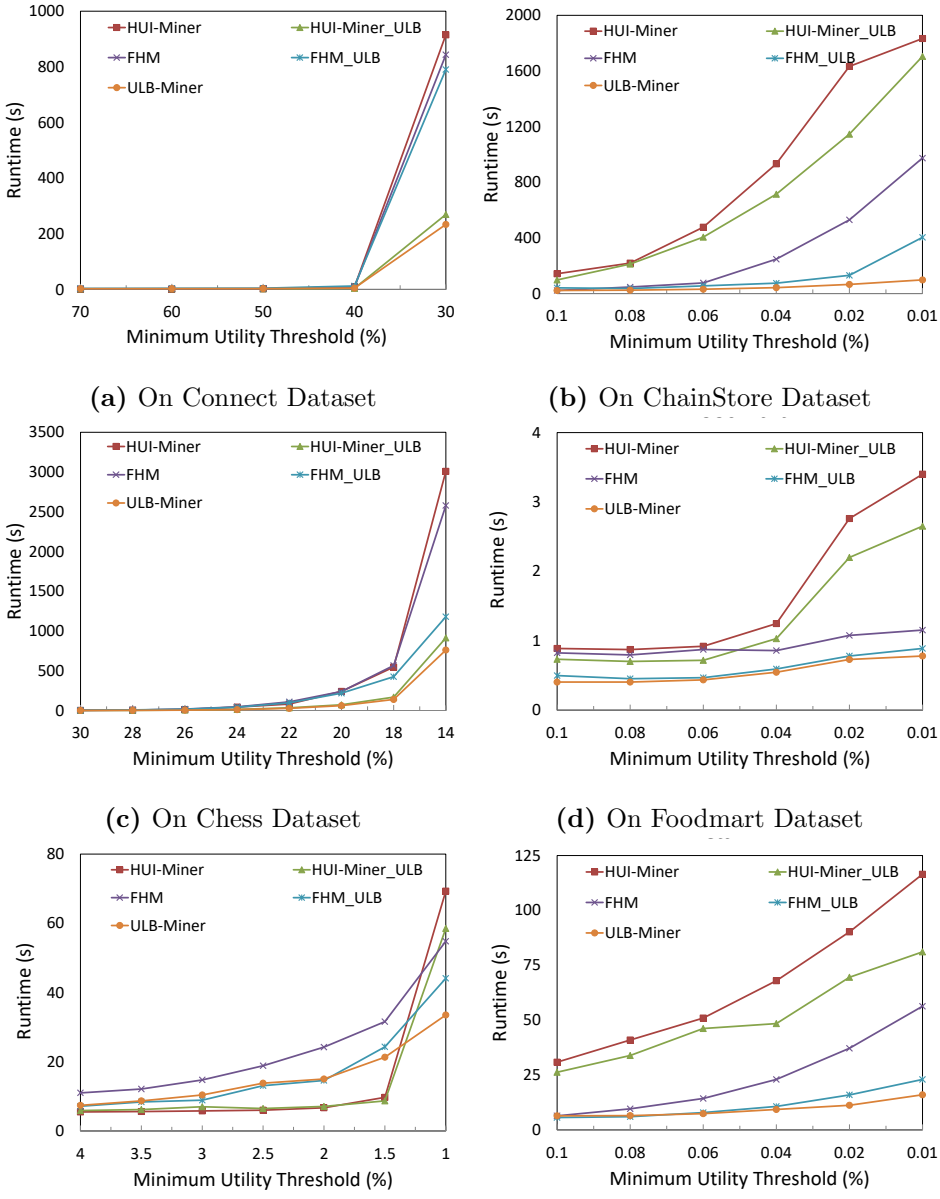
The performance of ULB-Miner is compared with two state-of-the-art HUI mining algorithms, namely HUI-Miner and FHM. These algorithms were chosen since they are state-of-the art HUIM algorithms. These algorithms are also based on the traditional utility-list structure. Moreover, we also prepared two improved versions of HUI-Miner and FHM, named HUI-Miner_ULB and FHM_ULB, respectively. These versions employ the proposed utility-list buffer structure and the basic utility-list segment construction procedure (Algorithm 2).

We ran the compared algorithms on each dataset while decreasing the *minutil* threshold until the algorithms became too long to execute, ran out of memory or a clear winner was observed. For each dataset, we recorded the execution time and memory consumption. The comparison of execution times is shown in Figure 3.8. As presented in these figures, the HUI-Miner_ULB and FHM_ULB versions are faster than the original implementations of these algorithms on all datasets. Especially, when *minutil* is decreased, there is a big gap between the runtimes of the original and improved versions. The proposed ULB-Miner algorithm is faster than the compared algorithms when *minutil* is small on the Kosarak dataset. For the remaining datasets, the proposed algorithm is the fastest for all *minutil* values. The compared algorithms are one-phase algorithms employing the traditional utility-list structure, which perform the costly utility-list intersection

[3] `http://fimi.cs.helsinki.fi/data/`

[4] `http://fimi.cs.helsinki.fi/data/`

[5] `http://www.philippe-fournier-viger.com/spmf/index.php?link=datasets.php`



(a) On Connect Dataset

(b) On ChainStore Dataset

(c) On Chess Dataset

(d) On Foodmart Dataset

(e) On Kosarak Dataset

(f) On Retail Dataset

Figure 3.8: Runtime comparison on different datasets

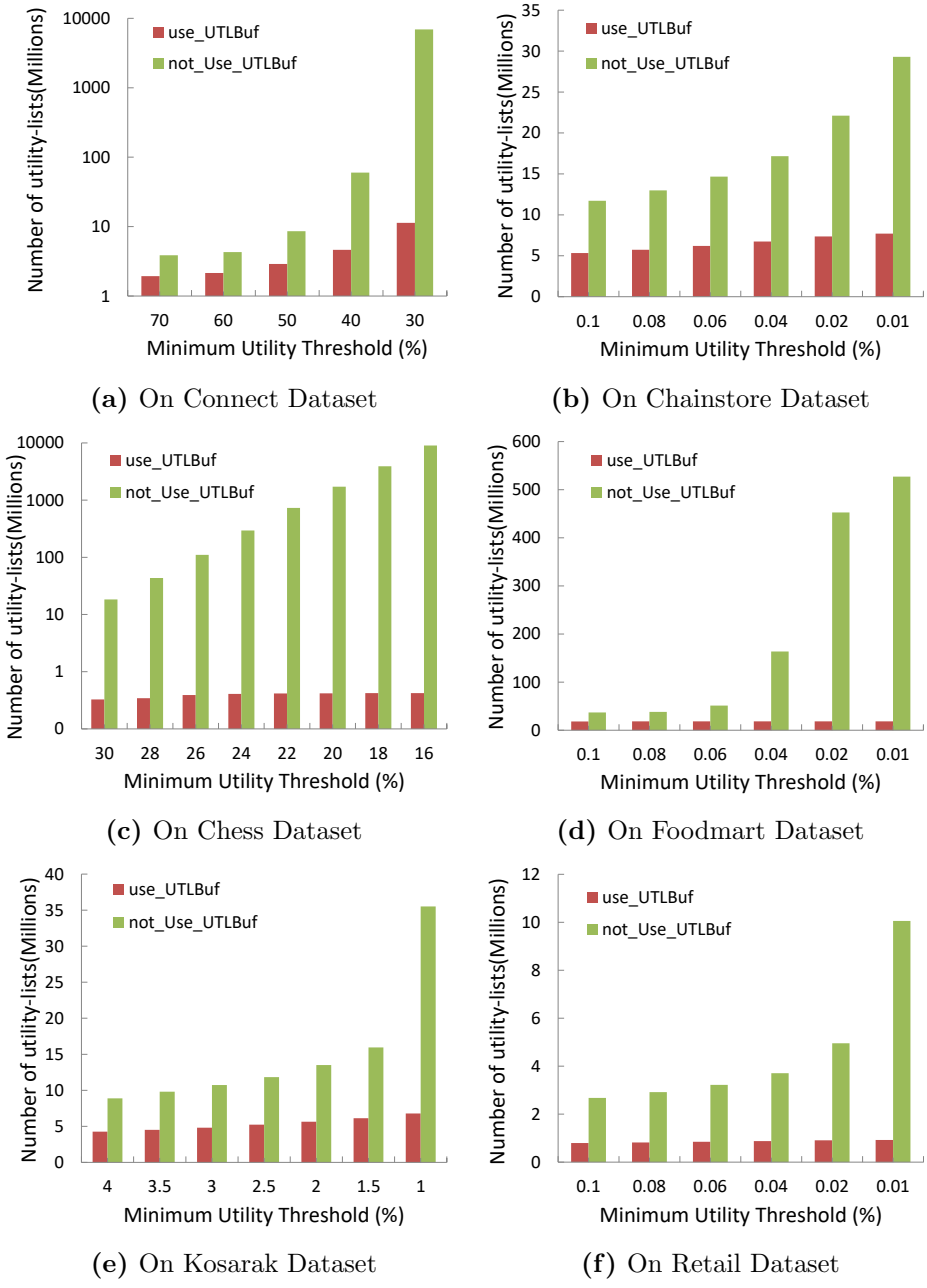
operation. To reduce this cost, ULB-Miner employs the designed efficient utility-list segment construction method to quickly search for transactions that are common to two utility-list segments. This considerably reduces its execution time.

3.5.3 Memory Consumption

Table 3.3 compares the peak memory usage of the algorithms on the six datasets when the *minutil* threshold is set to the smallest values used in the previous experiment. All memory measurements were done using the standard Java API. By observing these results, it is found that the proposed utility-list buffer structure reduces the memory consumption of the HUI-Miner and FHM algorithms on all datasets. The FHM_ULB and ULB-Miner algorithms consume almost the same amount of memory on the experimental datasets because they employ similar strategies. Both FHM_ULB and ULB-Miner consume less memory than FHM. The best results are obtained on the Chess and Connect datasets. There, the memory consumption is reduced by up to 4 and 6 times, respectively. This can be explained as follows. These datasets are dense with long transaction and many items. As a result, the algorithms generate a huge amount of candidates. But the proposed ULB-Miner algorithm reuses most of the memory for storing utility-lists thanks to its utility-buffer structure, and it thus have a low memory consumption. Similar results are also obtained when comparing the HUI-Miner and HUI-Miner_ULB algorithms. HUI-Miner_ULB consumes less memory than HUI-Miner on all datasets. The best result is obtained on the Chess dataset. Here, the gap in terms of memory usage is clear and large. The gap shrinks a bit due to the EUCS structure. But it is an acceptable trade-off when considering the runtime performance. On overall, the results depicted in Table 3.3 show that the proposed utility-list buffer structure is efficient in terms of memory consumption. In some cases, the proposed method can reduce memory consumption by up to 6 times.

3.5.4 Comparison on the Number of Utility-lists

To analyze in more details the memory consumption of the proposed algorithm, we performed an experiment to compare the number of utility-lists created by allocating new memory when using the designed utility-list buffer structure and when not using that structure. For this experiment, a version of the proposed ULB-Miner that employ the traditional utility-list structure [110] was prepared (i.e., that does not use the novel utility-list buffer



(a) On Connect Dataset

(b) On Chainstore Dataset

(c) On Chess Dataset

(d) On Foodmart Dataset

(e) On Kosarak Dataset

(f) On Retail Dataset

Figure 3.9: Comparison of the number of utility-lists created by allocating new memory when using or not using the utility-list buffer structure

Table 3.3: Comparison of peak memory usage (MB)

Dataset	HUI-Miner	HUI-Miner_ULB	FHM	FHM_ULB	ULB-Miner
Connect	452.6	399.9	1516.9	398.1	**368.6**
Chainstore	1367.3	**1021.1**	2792.7	2396.9	2402.7
Chess	752.4	**140.1**	1319.7	209.7	208.3
Foodmart	423.4	257.2	68.3	**40.1**	41.3
Kosarak	1060.2	**910.1**	1270	1030	1015.7
Retail	803.53	**442.1**	670.22	544.7	544.6

structure). Then, the number of utility-lists generated by allocating new memory was measured for both versions of the algorithm on each dataset.

Figure 3.9 shows the comparison. As presented in this figure, employing the designed utility-list buffer structure can greatly reduce the number of utility-lists created by allocating new memory during the mining process, especially for dense and long transaction datasets such as Chess. When the *minutil* threshold is set to small values, the difference in terms of number of generated utility-lists becomes clear and large. The reason is that for these datasets, there are many extensions for each considered itemsets. Hence, the number of utility-lists generated during the process of itemset extension is huge if the traditional utility-list structure is used. Fortunately, using the proposed utility-list buffer reduces the need to allocate new memory for utility-lists during the search by reusing the memory used for storing previously generated utility-lists.

3.5.5 Scalability Evaluation

Lastly, we performed experiments to evaluate the scalability of the proposed algorithm on a synthetic dataset named T10I4NXKDYK, where the number of transactions Y and the number of items X were varied. The dataset was generated using the IBM Quest synthetic data generator [5], where the numbers after T, I, N, and D represent the average transaction size, average size of maximal potentially frequent patterns, number of items, and the number of transactions, respectively. For this experiment, the *minutil* threshold was set to 0.05%, the number of items was varied from 2K to 10K, and the number of transactions was varied from 100K to 500K. Results are shown in Figure 3.10a and Figure 3.10b, respectively. As can be observed from these figures, the proposed algorithm has almost constant scalability when the number of items increases, and it has linear scalability when the number of transactions increases.

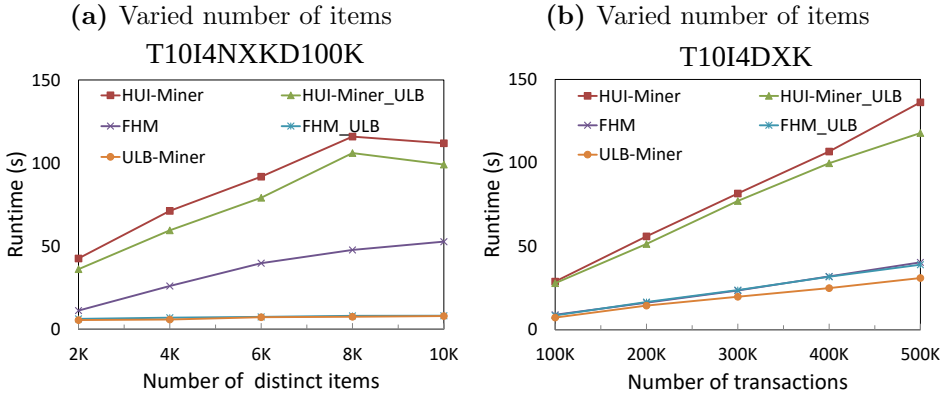


Figure 3.10: Scalability of the compared algorithms for different parameter values

3.6 Conclusion

This chapter has presented a novel structure named utility-list buffer for reusing the memory for storing utility-lists. We have proposed an algorithm for high utility itemset mining named ULB-Miner. This algorithm integrates the utility-list buffer structure with an efficient method for constructing utility-list segments to reduce the time and the memory usage required for mining high utility itemsets.

We have performed an extensive experimental study on six real-life datasets to compare the performance of ULB-Miner with the state-of-the-art algorithms HUI-Miner and FHM, which both employ traditional utility-lists. The results show that the proposed utility-list buffer structure and its construction method increase the effectiveness of HUI mining both in terms of execution time and memory consumption. The peak memory usage was reduced by up to six times, and execution times was reduced by up to 10 times.

Chapter 4

Detecting High Utility Drift in Quantitative Data Streams

The underlying distribution of data objects in a stream generally changes over time. Thus, the development of efficient methods and algorithms for analyzing transaction streams is an important research problem. An important challenge when analyzing data streams involves extracting interesting events (such as patterns, changes, and network attacks), and understanding these activity trends. The work presented in this chapter addresses research questions RQ1, RQ2 and RQ5 [50], i.e., *How can we design efficient methods to work well with a variety of different characteristics of data? How can we identify hypotheses for changes in the underlying distribution of streaming data and target concept drift over time? How can we simulate the correlations between single items in the global structure of the distribution?*

4.1 Motivation

As with general data streams, streaming transactional data is generally infinite and changes continuously. This combined with the high data generation speed makes mining of a stream of transactional data to discover patterns more challenging than mining a static database. Thus, developing efficient methods and algorithms for analyzing transaction streams is an important research problem [29, 114]. Nevertheless, most studies on this topic, including [103, 173], have focused on adapting traditional data mining techniques to streams and improving their efficiency to deal with streaming data. Note, however, that the underlying distribution of data objects in a stream generally changes over time [15], thus making such approaches unsuitable. At the same time, detecting changes, called *concept drifts*, is crucial because

it allows to discover the latest trends in a stream. A concept drift mainly refers to a significant decrease or increase in the distribution of data objects in a data stream with respect to a given measure [66].

In recent years, incremental and online learning have attracted the attention of many researchers to detect changes due to their numerous real-life applications, including market basket analysis, image processing, outlier detection, and climate monitoring [42, 116]. An important challenge of analyzing data streams is that trends may emerge, or remain steady over time, and that the streams often contain noise. In other words, to allow decision-makers to quickly react to changes, it is necessary to design efficient algorithms that can detect and monitor these changes in real-time. Nevertheless, although monitoring changes in data streams is widely recognized as important, most existing algorithms have mainly focused on discovering frequent patterns with changing frequencies, rather than considering changes in terms of other meaningful measures, such as the profit generated by the sale of items. To the best of our knowledge, only few approaches have been proposed to detect changes in the utility (profit) distribution of itemsets, where transactions are treated as streaming data. Monitoring such fluctuations in profit is necessary and important in many real-life applications including online retail stores and monitoring stock exchanges.

The work presented in this chapter is motivated by the need to address the limitations due to the lack of approaches that fully study the issues with concept drifts in high utility itemsets in data streams. We propose an efficient algorithm called HUDD-TDS (High Utility Drift Detection in Transactional Data Streams), with which we introduce several novel ideas to detect drifts efficiently.

Contributions

Overall, the main contributions of this chapter can be summarized as follows:

1. We introduce the task of detecting both local and global drifts by considering the utility measure and the recency of transactions in streams, defined as follows:

    - A *local utility drift* is a change in the utility distribution of an itemset (e.g., the utility of an itemset has recently considerably increased or decreased).
    - A *global utility drift* is a change in the total utility distribution of all itemsets (e.g., the sales of products in a retail store have

globally considerably increased or decreased).

2. We propose an efficient algorithm for the drift detection task in 1). The proposed algorithm relies on probability theory and statistical testing to identify changes in the utility distribution of itemsets in a quantitative data stream. Moreover, our approach takes into account the evolving behavior of the streams, and we employ a fading function to identify recent trends. Such a fading function is specifically useful in weighing the importance of transactions according to their age.

3. We introduce a new distance measure function called *Dmo* to compare high utility itemsets for detecting drifts. Although *Dmo* is based on the cosine similarity, which is a standard measure for calculating the similarity between vectors, it is more general in that *Dmo* not only considers the difference of vectors in terms of orientation (i.e., vector angles) but also magnitude. As discussed, this is necessary to address our problem.

4. We conduct an extensive experiment to evaluate the proposed method HUDD-TDS, showing the feasibility, effectiveness, and efficiency of our HUDD-TDS algorithm.

Organization

The remainder of this chapter is organized as follows. Section 4.2 briefly reviews the related work. Section 4.3 defines the problem of drift detection and introduces necessary preliminaries. Section 4.4 presents the proposed approach for detecting changes in the utility distribution of itemsets in a quantitative transaction data stream. Section 4.5 presents results from an extensive experimental evaluation to evaluate the performance of the proposed algorithm. Finally, Section 4.6 concludes the work.

4.2 Related Work

Detecting concept drifts is an important research problem that has applications in many domains such as flow prediction in industrial systems [128] and information filtering [89]. Numerous approaches have been proposed to detect changes in the distribution of data objects in data streams. Techniques for drift detection [63] are generally based on one of the following approaches: sequential analysis [64], statistical process control [23], comparison of two consecutive time windows [3], and contextual approaches [89].

The Hoeffding's Inequality has been used to design several approaches for determining the upper bounds for drift detection. Such upper bounds have been used in algorithms such as the Fast Hoeffding Drift Detection Method for Evolving Data Streams (FHDDM) [129], Hoeffding Adaptive Tree (HAT) [19], and the HAT+DDM+ADWIN [4] algorithm which extends ADaptive sliding WINdow (ADWIN) algorithm [18] and the Drift Detection Method (DDM) [64]. ADWIN is one of the most popular change detection algorithms, and it uses sliding windows to maintain the distribution and detect changes, whilst the DDM uses an online learning model to control the online error-rate and detect changes. Frías-Blanco et al. [61] proposed online and non-parametric drift detection methods using several bounds based on Hoeffding's Inequality. The algorithm can detect concept drifts based on the movements of distribution averages in streaming data. The algorithm uses counters to maintain information for detecting drifts. The time complexity of this approach is constant ($\mathcal{O}(1)$ for processing each data point in the stream).

On one hand, to discover high-utility patterns in data streams, some algorithms have been proposed [173, 102, 40]. These studies generally extend traditional HUIM methods to increase their efficiency in a streaming context. Nevertheless, they are not designed to detect changes or drifts. As mentioned earlier, customer transactions in retail stores can be seen as a stream of data, because customers continuously purchase products in the stores. This also means that the data is not static and is often impossible to store in memory due to its large volume. Moreover, in a streaming context, the distribution of data and the transactional behavior of customers can change and evolve over time. To the best of our knowledge, no work has been proposed to detect drifts for high utility itemset mining in streams of quantitative transactions, while considering the importance of utility over time.

On the other hand, in traditional frequent itemset mining, several algorithms have been proposed to identify concept drifts in data streams [121, 90]. Ng et al. [121] proposed a test paradigm named Algorithm for change detection (ACD) for detecting changes in transactional data streams by considering the support of itemsets for reservoir sampling. ACD evaluates drifts by performing reservoir sampling and applying three statistical tests. The ACD method employs a bound based on Hoeffding's Inequality to determine the number of transactions to be kept in each reservoir. ACD selects transactions to fill its reservoir using a distance measure that is a function of the support of single items. A major limitation of this approach is that high frequency items have more chance of being sampled. However, in terms of

utility (profit), high frequency items are often low utility itemsets in real-life customer transactions. Thus, this approach may find many patterns that are frequent but do not necessarily yield a high profit. Recently, Koh [90] proposed the CD-TDS algorithm, which considers two types of changes in transactional data streams for frequent pattern mining. A local drift, is a change in the frequency of single items, whilst a global change indicates a major change in the set of discovered frequent itemsets. The CD-TDS method uses a graph to represent the relationships between items in transactions, and the Levenshtein distance to calculate the similarity between sets of frequent itemsets found at different times. A limitation of CD-TDS is that it detects local drifts for single items, and not for itemsets. CD-TDS consider itemsets to detect global drifts but do not provide information about which itemsets contribute the most to these global drifts. Moreover, CD-TDS considers drifts in terms of pattern frequencies, but it does not take into account changes in terms of utility. In addition, another limitation of the CD-TDS algorithm is that it treats all transactions as equally important. However, in real-life data streams, the most recent transactions are generally the most important transactions, as they provide information about recent trends.

4.3 Preliminaries

This section introduces preliminaries related to high utility itemset mining and drift detection. In the following paragraphs, we present some basic definitions that we will use in this chapter. The notations used in this chapter are summarized in Table 4.1. Let there be a set of items $I = \{i_1, i_2, \ldots, i_m\}$ representing products sold in a retail store. For each item $i_j \in I$, the external utility of i_j is a positive number representing its unit profit (or more generally, its relative importance to the user). The external utility of an item i_j is denoted as $p(i_j)$. Let there be an infinite sequence of increasing positive integers $1 \leq x1 < x2 < x3 \ldots$. A streaming quantitative transactional database D is an infinite sequence of transactions $D = \{T_{x1}, T_{x2}, T_{x3}, \ldots\}$, where for each transaction $T_d \in D$, the relationship $T_d \in I$ holds. Moreover, for each transaction $T_d \in D$, d is a unique integer that is said to be the TID (Transaction IDentifier) of T_d, and represents its observation time. Thus, for two transactions T_a and T_b, if $a < b$, this indicates that transaction T_a has occurred before T_b. Assume the stream D does not contain two transactions with the same observation time[1]. Consider two observation times a and b, a

[1]If two transactions are simultaneous, a total order on these transactions can be obtained by incrementing the observation time of one of those transactions by a small value.

Table 4.1: Table of notations

Symbols	Description	Symbols	Description
D	Quantitative database for transaction stream	T_d	A specific transaction in D having transaction IDentifier d
S	Sequence of transactions	I	Set of items in the database D
X	An itemset	$p(i)$	External utility of single item i
$q(i, T_d)$	Internal utility of an item i in a transaction T_d	W_{ab}	A data window containing transactions from the observation time a to the observation time b
$u(X, T_d)$	Utility of itemset X in transaction T_d	$TUD(W)$	Total utility of window W in a stream D
$u(X, W)$	Utility of itemset X in window W	μ	Population mean
H	Hypothesis	u_t	Value in the stream at the observation time t
λ	Decay factor	$d^{\lambda}(t)$	Decay function of factor λ and time t
$U(T_i, t, \lambda)$	Utility of transaction T_i in the stream at observation time t	$U_S(X, T_i, t, \lambda)$	Utility of itemset X in transaction T_i at the observation time t
$HS_W^{n,\lambda}$	Set of all high utility itemsets in W of stream at observation time n	$\overline{U}$	Average of the sequence variables U_i
$\|W\|$	Cardinality: The number of members in W	α	Confidence level of hypothesis
ε	Error value	σ^2	Variance of utility in a window
Pr	Probability	E	Expectation value of variable
$\overrightarrow{Root_S}$	*Root* vector of stream S	$\overrightarrow{X}$	Vector of itemset X
$\|\overrightarrow{X}\|$	Euclidean norm, or Euclidean length, or magnitude of vector $\overrightarrow{X}$	$S_{cos}(\overrightarrow{x}, \overrightarrow{y})$	Cosine similarity between $\overrightarrow{x}$ and $\overrightarrow{y}$
$S_{mo}(\overrightarrow{x}, \overrightarrow{y})$	Similarity between $\overrightarrow{x}$ and $\overrightarrow{y}$	$D_{mo}(\overrightarrow{x}, \overrightarrow{y})$	Distance between $\overrightarrow{x}$ and $\overrightarrow{y}$
DIS_{HS}	Sum of the movements of high utility itemsets in the set HS to the root vector		

data window W_{ab} is the finite sub-sequence of D containing all transactions from the observation time a to the observation time b. Formally, $W_{ab} = \{T_{y1}, T_{y2}, \ldots, T_{yn}\}$, for all integers $y1, y2, \ldots, yn$ such that $a \leq y1 < y2 < \cdots < yn \leq b$, and $T_{y1}, T_{y2}, \ldots, T_{yn}$ appear in D. The internal utility of an item i_j in a transaction T_d is denoted as $q(i_j, T_d)$. It is a positive number representing the purchase quantity of item i_j in T_d. A set of items $X = \{i_1, i_2, \ldots, i_l\} \subseteq I$ containing l items is said to be an itemset of length l, or alternatively, an l-itemset. For example, Figure 4.1 shows the first four transactions of a streaming quantitative transactional data stream, which will be used as running example. In this stream, the set of items I in D is $\{a, b, c, d, e, g\}$. The external utilities of these items are respectively 5, 2, 1, 2, 3, and 1.

TID	**Transaction**	**Transaction utility**
1	(a,1), (c,1), (d,1)	8
2	(a,2), (c,6), (e,2), (g,5)	27
3	(b,4), (c,3), (d,3), (e,1)	20
4	(b,2), (c,3), (e,2), (g,2)	15

Item	a	b	c	d	e	g
External utility	5	2	1	2	3	1

Figure 4.1: Four transactions of a quantitative transactional stream (top) and the corresponding external utilities of items (bottom).

Example 3. *Consider the stream of Figure 4.1. The utility of itemset c in transaction T_1 is $u(c, T_1) = 1 \times 1 = 1$. The utility of itemset ac in T_1 is $u(ac, T_1) = u(a, T_1) + u(c, T_1) = 1 \times 5 + 1 \times 1 = 5 + 1 = 6$. The utility of transaction T_1 is $TU(T_1) = u(a, T_1) + u(c, T_1) + u(d, T_1) = 5 + 1 + 2 = 8$. Consider the window $W = \{T_1; T_2\}$ and that* minutil *is set to 22. The set of high utility itemsets in that window W is {ac:22, ace:22}, where the number beside each itemset indicates its utility.*

Definition 26 (Utility of a transaction in a stream)**.** *Let* S *= (T_{x1}, T_{x2}, …, T_{xn}) be a sequence of transactions, where the notation T_i denotes the transaction that occurred at time* i*. At the time of observation* t*, the utility of a transaction T_i in the stream S is denoted as $U(T_i, t, \lambda)$ and defined as: $U(T_i, t, \lambda) = U(T_i, i) * d^{\lambda}(\triangle T)$, where $\triangle T = t - i \geq 0$ and $U(T_i, i)$ is the utility of transaction T_i at time* i*, it is equal to $TU(T_i)$.*

Detecting drifts [63] in a stream can be done based on the following definitions. Generally, let there be a stream S that is a sequence of values $\{u_1;$

$u_2; \ldots ; u_t; u_{t+1}; \ldots ; u_n\}$. Let μ_1 and μ_2 respectively denote the population means of two samples of instances U_1 and U_2, where $U_1 = \{u_1; u_2; \ldots; u_t\}$ and $U_2 = \{u_{t+1}; \ldots; u_n\}$. Detecting drifts using a statistical test can be done by comparing two hypotheses. The null hypothesis is that the population means of the two samples have the same distribution, that is H_0: $\mu_1 = \mu_2$. The alternative hypothesis H_1 is that $\mu_1 \neq \mu_2$. A statistical test is then applied to determine if the null hypothesis holds for a significance level α. The rule to accept the H_1 hypothesis is $Pr(|\mu_1 - \mu_2| \geq \varepsilon) \geq \alpha$, where ε is a user-defined positive number. A drift is said to occur at an observation time t if the population of the sample at time t is significantly different from that of the preceding observation time. In that case, t is said to be a drift point.

The problem of detecting drifts is to find the observation times where there are significant differences in the data distribution for a given measure (e.g., the support) with respect to the preceding observation times. Although several papers, e.g., [61, 90] have proposed approaches for drift detection, none have considered detecting drifts in the utility distribution of patterns including the utility distribution of items in patterns as well as the utility distribution of patterns in the whole set of patterns. However, the utility is a more useful measure compared to the support measure as it measures the profit generated by patterns, rather than simply measuring the number of transactions containing the items without considering their purchase quantities.

To allow discovering more useful patterns and drifts, this work adapts the concept of drift to the utility measure to propose the problem of drift detection for high utility itemset mining in an evolving data stream. It consists of finding all observation times where there are significant differences in the utility distribution of itemsets, while also considering the recency of transactions. Finding such drifts provides information that allows decision-makers to quickly react to changes in customer behavior that influence profitability.

4.4 Model and Solution

This section introduces the High Utility Drift Detection in Transactional Data Stream (HUDD-TDS) algorithm to detect changes in the utility distribution of high utility itemsets in a stream of quantitative customer transactions. HUDD-TDS can detect both local and global changes by considering the utility measure, the recency of transactions, and the correlative conjunction of the utility distributions of itemsets in streams.

It gives most importance to the most recent transactions since in prac-

tice, i.e., customer transaction data streams, recent trends are considered as the most valuable. In particular, the proposed approach uses a fading function to assign a weight to each transaction that is inversely proportional to its age. The next subsections describe the proposed approach in details.

4.4.1 A Fading Function for High Utility Itemset Mining in a Stream

An important characteristic of customer transaction data streams is that trends found in a stream change with time, as the behaviors of customers vary. In a data stream, data points arrive at a high speed. As previously explained, the importance of data points (transactions) can be viewed as inversely proportional to their age. Hence, a transaction that occurred a long time ago (before the current observation time) should be considered as less important than a recent transaction. To model the varying importance of transactions with respect to the observation time, a decay (fading) function is used in the proposed algorithm. At the time of observation, the utility values in a transaction that occurred at a time t are multiplied by a decay factor calculated by a decay function $d^{\lambda}(t)$. The calculated decay factor is a value $d^{\lambda}(t) \in [0, 1]$. The decay function $d^{\lambda}(t)$ is a user-defined function that is inversely proportional to the elapsed time. The decay function takes a positive constant λ as parameter, called the decay constant, which let the user indicate how fast the importance of transactions should decrease with respect to time.

Definition 27 (A fading function to consider the recency of transactions)**.** *Let* S $= (T_{x1}, T_{x2}, \ldots, T_{xn})$ *be a sequence of transactions, and* $d^{\lambda}(T)$ *be the user-defined decay function. The utility of an itemset* X *in a transaction* T_i *at the observation time* $t \geq i$ *is denoted as* $U_S(X, T_i, t, \lambda)$*, and defined as* $U_S(X, T_i, t, \lambda) = U_S(X, T_i, i, \lambda) \times d^{\lambda}(\triangle T) = u(X, T_i) \times d^{\lambda}(\triangle T)$*, where* $U_S(X, T_i, i, \lambda)$ *is the utility of itemset* X *in* T_i *at observation time* i*, and* $u(X, T_i)$ *is the utility of* X *in* T_i *as defined in traditional high utility mining (without applying the decay function).*

Example 4. *Consider the stream database of Figure 4.1. Suppose that the decay function is* $d^{\lambda}(\triangle T) = 2^{\frac{-\triangle T}{2}}$*. The utility of itemset* ac *in* T_1 *at observation time* t $= 2$ *is* $U_S(ac, T_1, 2, \lambda) = 6 \times 2^{\frac{-1}{2}} = 4.24$*. The utility of itemset* ac *in* T_1 *at time* t $= 3$ *is* $U_S(ac, T_1, 3, \lambda) = 6 \times 2^{\frac{-2}{2}} = 3$*. Consider the window* $W = \{T_1; T_2\}$*, and that* minutil *is set to 22. The set of high utility itemsets in* W *when considering transaction recency is* $\{ace{:}22\}$*. The itemset ac is not high utility because its utility is* $U_S(ac, T_1, 2, \lambda) + U_S(ac, T_2, 2, \lambda) = 4.24 + 16 = 20.24 <$ minutil.

There are two important differences between mining high utility itemsets in a stream and in a static transaction database. First, not all data points from a stream can be stored and kept in memory due to the limited amount of memory of a computer and the infinite nature of a stream. For this reason, data points can only be read once. Second, an important issue is that the utility of a transaction should decrease according to a decay function that is inversely proportional to its age. To consider the decay function when mining high utility itemsets in a quantitative transactional data stream, we redefine the problem of HUIM as follows. Let $W = (T_{y1}, T_{y2}, \ldots, T_{ym})$ be a window containing m transactions at the observation time ym of a sequence of transactions S. Let there be a threshold value θ defined by the user. An itemset X is a high utility itemset in W if the sum of its utilities in W is not less than θ, that is: $\sum_{T_i \in W}(U_S(X, T_i, ym, \lambda)) \geq \theta$. In the following, the notation $HS_W^{n,\lambda}$ is used to denote the set of all high utility itemsets found in a window W of a transactional data stream at observation time n. Formally, $HS_W^{n,\lambda} = \{X, \sum_{T_i \in W}(U_S(X, T_i, n, \lambda)) \geq \theta\}$.

4.4.2 A Hoeffding Bound to Assess the Significance of Drifts

Having explained how we apply fading, this section proposes a bound to detect utility drifts. The proposed bound is based on the Hoeffding Inequality [80] from the probability theory. This inequality has been used in various studies to analyze data streams [129, 18, 61, 90]. Given some independent random variables, the Hoeffding Inequality provides an upper bound on the probability that their sum deviates from its expected value. In this work, the Hoeffding Inequality is used as the basis for assessing if changes are statically significant in a flow of independent random transactions arriving in a data stream. If the probability of a predefined condition is greater than a user-specified threshold, the proposed approach considers that there is a change in the data. The Hoeffding's Inequality theorem states as follows [80].

Theorem 2 (Hoeffding's Inequality)**.** *Let U_1; U_2; ... ; U_n be independent random variables bounded by the interval [0, 1], that is $0 \leq U_i \leq 1$, where $i \in \{1; \ldots; n\}$. Let $\overline{U}$ denote the average of the random variables, that is $\overline{U} = \frac{1}{n}\sum_{i=1}^{n}(U_i)$. We have:*

$$Pr(\overline{U} - E[\overline{U}] \geq \varepsilon) \leq e^{-2n\varepsilon^2}, \tag{4.1}$$

where E[X] is the expected value of X.

The inequality states that the probability that the estimation and true values differ by more than ε is bounded by $e^{-2n\varepsilon^2}$. Symmetrically, the inequality is also valid for the other side of the difference: $Pr(-\overline{U} + E[\overline{U}] \geq \varepsilon) \leq e^{-2n\varepsilon^2}$. As a result, a two-sided variant of the Inequality is obtained:

$$Pr(|\overline{U} - E[\overline{U}]| \geq \varepsilon) \leq 2e^{-2n\varepsilon^2} \tag{4.2}$$

This Inequality is true if all variables are bounded by the [0, 1] interval. More generally, if a variable U_i is bounded by an interval $[x_i, y_i]$, the Hoeffding's Inequality is generalized as follows:

$$Pr(|\overline{U} - E[\overline{U}]| \geq \varepsilon) \leq 2e^{\frac{-2n^2\varepsilon^2}{\sum_{i=1}^{n}(y_i - x_i)^2}} \tag{4.3}$$

The Hoeffding's Inequality (Theorem 2) can be used to assess the significance of changes in a stream of values, based on the following proposition.

Proposition 2. *Let $U_1; U_2; \ldots; U_n$ be independent random variables bounded by the [0, 1] interval. These variables can be split into two windows using an index m as splitting point. This results in two windows, $W_1 = \{U_1; \ldots; U_m\}$ and $W_2 = \{U_{m+1}; \ldots; U_n\}$, such that $1 \leq m < n$. Then, for an error $\varepsilon > 0$, the following inequality holds:*

$$Pr(\overline{U} - \overline{V} - (E[\overline{U}] - E[\overline{V}]) \geq \varepsilon) \leq e^{\frac{-2\varepsilon^2|W_1|\cdot|W_2|}{|W_1|+|W_2|}}, \tag{4.4}$$

where $|W_1|$ and $|W_2|$ are the size of W_1 and W_2, respectively. Moreover, $\overline{U} = \frac{1}{|W_1|}\sum_{i=1}^{m}(U_i)$ and $\overline{V} = \frac{1}{|W_2|}\sum_{i=m+1}^{n}(U_i)$.

If the two-sided variant of the inequality is considered:

$$Pr(|(\overline{U} - E[\overline{U}]) - (\overline{V} - E[\overline{V}])| \geq \varepsilon) \leq 2e^{\frac{-2\varepsilon^2|W_1|\cdot|W_2|}{|W_1|+|W_2|}} \tag{4.5}$$

Based on proposition 2, consider a significant confidence level α (probability of making an error), that controls the maximum false positive rate. The error ε can be estimated with respect to α as follows.

$$\begin{aligned} \alpha &= 2e^{\frac{-2\varepsilon^2|W_1|\cdot|W_2|}{|W_1|+|W_2|}} \Rightarrow \frac{2\varepsilon^2|W_1|\cdot|W_2|}{|W_1|+|W_2|} = \ln\frac{2}{\alpha} \\ \Rightarrow \varepsilon &= \sqrt{\frac{|W_1|+|W_2|}{2|W_1|\cdot|W_2|}\ln\frac{2}{\alpha}} \end{aligned} \tag{4.6}$$

To assess the significance of changes in a stream of values, we use Eq. 4.6 to obtain the cut point value ε_α based on the predefined α threshold. An

observation time m is said to be a *distribution change point* if there is a significant difference between the averages of values in the windows W_1 and W_2, that is the difference is not less than ε_α. Furthermore, if that condition holds, we also say that there is a change and that the windows W_1 and W_2 are different. This can be expressed as a statistical test with bounding probability of errors. Let the null hypothesis be H_0: $(\overline{U} - E[\overline{U}]) = (\overline{V} - E[\overline{V}])$, stating that the distributions of the two windows are equal (assuming independent random variables). The H_0 hypothesis is compared with the alternative hypothesis H_1: $(\overline{U} - E[\overline{U}]) \neq (\overline{V} - E[\overline{V}])$ with the rule $|(\overline{U} - E[\overline{U}]) - (\overline{V} - E[\overline{V}])| \geq \varepsilon$ to reject H_0. HDDM [61] proposed a minor improvement of the Hoeffding's Inequality by using two counters instead of three counters for maintaining the left, right and the total mean at the cut point. The proposed method inherits this improvement to detect a global drift. This improvement is derived from the Hoeffding's Inequality as follows.

Proposition 3. *Let $U_1; U_2; ...; U_n$ be independent random variables bounded by the [0, 1] interval. These variables can be split into two windows at an index* m *to obtain $W_1 = \{U_1; \ldots; U_m\}$ and $W_2 = \{U_{m+1}; \ldots; U_n\}$, such that $1 \leq m < n$. Then, for an error $\varepsilon > 0$, the following inequality is obtained:*

$$Pr(\overline{U} - \overline{V} - (E[\overline{U}] - E[\overline{V}]) \geq \varepsilon) \leq e^{\frac{-2nm\varepsilon^2}{(n-m)}}, \tag{4.7}$$

where $\overline{U} = \frac{1}{m}\sum_{i=1}^{m}(U_i)$ and $\overline{V} = \frac{1}{n}\sum_{i=1}^{n}(U_i)$.

The estimated error ε with respect to α is:

$$\begin{aligned} \alpha &= 2e^{\frac{2nm\varepsilon^2}{(n-m)}} \Rightarrow \frac{-2nm\varepsilon^2}{(n-m)} = \ln\frac{2}{\alpha} \\ \Rightarrow \varepsilon &= \sqrt{\frac{n-m}{2nm}\ln\frac{2}{\alpha}} \end{aligned} \tag{4.8}$$

4.4.3 Two Mechanisms to Detect Utility Changes

This subsection presents the proposed approach to detect changes in the distribution of high utility itemsets in a stream of customer transactions. In general, change detection consists of detecting each observation time where there is a significant difference in the distribution of the data. In this work, the object of change analysis is the sets of high utility itemsets mined from consecutive windows of transactions in a stream. As mentioned above, two kinds of changes are considered in this work: local and global utility changes. A local utility change is a drift in the utility distribution of a high utility

itemset. A global utility change is a drift in the overall utility distribution of all high utility itemsets. The next paragraphs describe mechanisms to detect these two types of changes.

Local Utility Change Detection

Let $HS^{x,\lambda}_{W_1}$ and $HS^{y,\lambda}_{W_2}$ be two sets of high utility itemsets mined at two different observation times in a transactional data stream. Local utility change detection consists of comparing the utility of each high utility itemset X in $HS^{x,\lambda}_{W_1}$ and $HS^{y,\lambda}_{W_2}$. For an itemset X, if there is a significant difference in terms of utility for the two observation times, it is called a local drift of X at the change point from $HS^{x,\lambda}_{W_1}$ to $HS^{y,\lambda}_{W_2}$. The theoretical background of our method is probability theory, statistical testing and the Hoeffding's Inequality. To detect a drift, existing methods such as CD-TDS [90] and ADWIN [18] apply the Bonferroni correction with the Hoeffding's Inequality. The reason is that the Bonferroni correction prevents an increase of the probability of incorrectly rejecting the null hypothesis when testing multiple hypotheses.

In this work, the proposed approach applies the Bonferroni correction with the Hoeffding's Inequality to detect local drifts for high utility itemsets in two consecutive windows. For the two sets of high utility itemsets $HS^{x,\lambda}_{W_1}$ and $HS^{y,\lambda}_{W_2}$, let n_1 and n_2 be the number of sampled transactions from the stream that were used to obtain the sets $HS^{x,\lambda}_{W_1}$ and $HS^{y,\lambda}_{W_2}$ at their respective observation times. For the sake of brevity, the notation HS_1 and HS_2 will be used to refer to these sets in the following. There is a local change for an itemset X from HS_1 to HS_2 if its utility distribution difference in HS_1 and HS_2 is not less than a cut value:

$$\begin{aligned} \varepsilon_\alpha &= \sqrt{\frac{2(n_1+n_2)}{n_1 n_2}\sigma^2 \ln\frac{\ln(n_1+n_2)}{\alpha}} \\ &\quad + \frac{2(n_1+n_2)}{3n_1n_2}\ln\frac{2(n_1+n_2)}{\alpha} \\ &= \sqrt{2m\sigma^2\ln\frac{2\ln(n)}{\alpha}} + \frac{2m}{3}\ln\frac{2\ln(n)}{\alpha}, \end{aligned} \tag{4.9}$$

where $n = n_1 + n_2$, $m = n_1^{-1} + n_2^{-1}$, α is a user-defined confidence level, and σ^2 is the observed variance of the utility of the itemset in the window W formed by joining W_1 and W_2.

The variance σ^2 is defined and computed as the sum of the squared distances of each sample in the distribution from the mean, divided by

the number of samples in the distribution. In the proposed algorithm, an efficient way of calculating the standard deviation for a set of numbers is proposed and as follows.

$$\sigma^2 = \frac{1}{N}\sum X^2 - (\frac{\sum X}{N})^2 \tag{4.10}$$

Global Utility Change Detection

Global utility change detection aims at detecting changes by comparing the utility distributions of high utility itemsets mined in two consecutive windows. For each itemset X, the utility of X is a function of the utility distributions of the items that it contains. Let $I = \{i_1, i_2, \ldots, i_n\}$ be the set of all items that appear in a customer transaction data stream. A high utility k-itemset X can be represented as $(\{i_{j1} : u_{j1}\}, \{i_{j2} : u_{j2}\}, \ldots, \{i_{jk} : u_{jk}\})$, where the notation i_{jx} represents an item in X, and the number beside each item indicates the utility contributed by that item to the utility of X. Because of differences in the utility distributions of items in high utility itemsets, this work proposes a custom distance measure to efficiently detect drifts for high utility itemsets. The distance of each itemset to a reference point is calculated. Then, the distance of a set of high utility itemsets to the reference point is calculated as the sum of the distances of high utility itemsets to that point. The proposed measure is inspired by the observation that an itemset X with its utility can be presented as a vector, and that the distance between two itemsets can thus be calculated using vector distance measures. Formally, the distance between high utility itemsets is computed based on the following definitions.

Definition 28 (Root vector)**.** *Let* $S = (T_1, T_2, \ldots, T_n, \ldots)$ *be a sequence of transactions, and* $I = \{i_1, i_2, \ldots, i_n\}$ *be a set of* n *items in* S*, such that* $i_j \prec i_{j+1}$ *with* $1 \leq j < n$ *and* $\prec$ *be any total order on items from* I*. The Root vector in* S *is denoted as* $\overrightarrow{Root_S} = \overrightarrow{i_1 i_2 \ldots i_n} = \{1, 1, \ldots, 1\}$*. It is a vector described with* n *properties, and the value of each property is equal to 1 unit.*

Definition 29 (Vector of a high utility itemset)**.** *Let* $I = \{i_1, i_2, \ldots, i_n\}$ *be a set of items that appear in a set of transactions. Moreover, let there be a high utility* k*-itemset* $\mathrm{X} = \{i_{j1} i_{j2} \ldots i_{jk}\}$ *such that the utility distribution of its items is* $(\{i_{j1} : u_{j1}\}, \{i_{j2} : u_{j2}\}, \ldots, \{i_{jk} : u_{jk}\})$*. The vector of the high utility itemset* X *is denoted as* $\overrightarrow{X}$ *and defined as* $\overrightarrow{X} = \{U_1, U_2, \ldots, U_n\}$*,*

where

$$U_j = \begin{cases} 0, & \text{if } i_j \notin X \\ u(i_j), & \text{otherwise.} \end{cases} \tag{4.11}$$

Distance. The first step for calculating the distance in the proposed approach is to calculate the similarity between two vectors using the cosine similarity. Consider two vectors for some high utility itemsets X and Y, defined as $\overrightarrow{X} = \{X_1, X_2, \ldots, X_n\}$, $\overrightarrow{Y} = \{Y_1, Y_2, \ldots, Y_n\}$, respectively. The cosine similarity between $\overrightarrow{X}$ and $\overrightarrow{Y}$ is:

$$\begin{aligned} S_{cos}(\overrightarrow{X}, \overrightarrow{Y}) &= \frac{\overrightarrow{X} \cdot \overrightarrow{Y}}{\|\overrightarrow{X}\| \cdot \|\overrightarrow{Y}\|} \\ &= \frac{\sum_{i=1}^{n}(X_i \cdot Y_i)}{\sqrt{\sum_{i=1}^{n} {X_i}^2}\sqrt{\sum_{i=1}^{n} {Y_i}^2}} \end{aligned} \tag{4.12}$$

The cosine similarity is a standard measure for calculating the similarity between vectors. However, a drawback of this measure is that it only considers the difference in orientation of two vectors, while ignoring their difference in terms of magnitude [79]. For example, a cosine similarity of 1 indicates that two vectors have the same orientation. But these vectors may or may not have the same magnitude. Meanwhile, magnitude of an itemset vector represents its utility. To consider not only the difference in terms of orientation but also in terms of magnitude, the proposed approach calculates the similarity between two vectors $\overrightarrow{X}$ and $\overrightarrow{Y}$ using an improved similarity measure denoted as $S_{mo}(\overrightarrow{X}, \overrightarrow{Y})$, and defined as follows.

$$S_{mo}(\overrightarrow{X}, \overrightarrow{Y}) = S_{cos}(\overrightarrow{X}, \overrightarrow{Y}) \times (1 - \frac{abs(\|\overrightarrow{X}\| - \|\overrightarrow{Y}\|)}{max(\|\overrightarrow{X}\|, \|\overrightarrow{Y}\|)} \tag{4.13}$$

Similarly to the cosine similarity, the proposed S_{mo} similarity measure assigns a value of 1 to two vectors having the same orientation and magnitude. However, in the proposed approach, the distance between vectors must be calculated rather than the similarity. Thus, based on S_{mo}, a distance measure D_{mo} is defined as: $D_{mo}(\overrightarrow{X}, \overrightarrow{Y}) = 1 - S_{mo}(\overrightarrow{X}, \overrightarrow{Y})$. In the following, this measure is called the *movement* between two vectors. To check if there is a global drift between two sets of high utility itemsets HS_1 and HS_2, the proposed approach computes the sum of the movements of high utility itemsets in the two sets.

$$DIS_{HS} = \sum_{X \subseteq HS} D_{mo}(\overrightarrow{X}, \overrightarrow{Root}) \tag{4.14}$$

Example 5. *Consider the stream of Figure 4.1 and the decay function* $d^{\lambda}(\triangle T) = 2^{\frac{-\triangle T}{2}}$. *Furthermore, suppose that* minutil *is set to 22 and that the window size is set to 2. Consider the windows* $W_1 = \{T_1; T_2\}$ *and* $W_2 = \{T_3; T_4\}$. *By taking transaction recency into account, the set of high utility itemsets in* W_1 *is* $\{ace{:}22\}$. *The vector of the itemset* ace *is* $\overrightarrow{\text{ace}} = \{10, 0, 6, 0, 6, 0\}$. *The set of high utility itemsets in* W_2 *is* $\{bce{:}22.9\}$. *The utility of itemset bce is computed as the sum of its utilities in* T_3 *and* T_4, *that is* $13 + 7 \times \sqrt{2} = 22.9$. *The vector of the itemset bce is* $\overrightarrow{\text{bce}} = \{0, 9.66, 5.12, 0, 8.12, 0\}$. *The distance of the vector of itemset* ace *to the* root *vector is computed as* $D_{mo}(\overrightarrow{ace}, \overrightarrow{root}) = 1 - S_{mo}(\overrightarrow{ace}, \overrightarrow{root}) = 1 - 0.128 = 0.872$, *where* $S_{mo}(\overrightarrow{ace}, \overrightarrow{root})$ *is computed by Eq. 4.13. In a similar way, the distance of the vector of itemset* bce *to the* root *vector is computed as* $D_{mo}(\overrightarrow{bce}, \overrightarrow{root}) = 1 - 0.1234 = 0.8766$.

The *total distribution* is the sum of the distances of all high utility itemset vectors to the root vector. After that, the algorithm uses a statistical test (the A-test [61]) with the designed Hoeffding bound to determine if a global drift occurred at the current observation time. Note that it is also possible to detect drifts that represent increasing or decreasing trends by using the one-side or two-side variant of the Hoeffding's Inequality. We also consider increasing or decreasing trends and report it in Evaluation section of this chapter.

4.4.4 The Change Detection Algorithm

In the proposed approach, high utility itemsets are obtained at different observation times using windows on the stream. A set of high utility itemsets found at a given time is eventually replaced by a newer set, as time passes. Another important characteristic of a data stream is that data points arrive at a very high speed. Thus, an algorithm would be inefficient if it checks for changes for each new data point in the stream. The solution to this problem is to let the user set a parameter that determines the frequency of checks. On one hand, frequently checking for changes decreases the efficiency of the algorithm. On the other hand, rarely checking for changes results in higher efficiency but increases the risk of missing drift points as the windows may be too large. In the proposed method HUDD-TDS (High Utility Drift Detection in Transactional Data Stream), the checking frequency is a time interval length, and it is set by the user. The detailed pseudocode of the proposed method is presented in Algorithm 7.

The Algorithm 7 takes a stream of quantitative customer transactions as input. When a new transaction T_i is read from the stream, the transaction

Algorithm 7 HUDD-TDS: High Utility Drift Detection in a Utility Data Stream

Input: Stream of transactions with utility.
Output: Utility changes in the stream.

Init() Initialize variables: *minutil*, *interval*, *window size*, *confidence*, etc.
for (each transaction T_i arriving on a stream $T_{x1}; T_{x2}; \ldots; T_{xi}; \ldots$) **do**
    if (Memory is full) **then**
        Remove the oldest transactions from memory
    Add T_i to the limited memory
    if (Observation time i % check_interval = 0) **then**
        $checkpoints[i].HUIs \longleftarrow$ **HUI-Discovery**(i) using utility-list based mining method
        $checkpoints[i].distance$ = sum of the distance of each itemset X in $checkpoints[i].HUIs$
        if (*IsGlobalDrift*(checkpoints)) **then**
            Update last index where a drift is detected
            Output global drift
        if (*IsLocalDrift*(checkpoints)) **then**
            Output local drift

Algorithm 8 HUI-Discovery: Mine all high utility itemsets

Input: Observation time t and transactions.
Output: Set of high utility itemsets.

Init() Initialize all variables
$HUIsSet \leftarrow \phi$
Scan transactions in memory to create a window W of a predefined length, ending at observation time t
for (each transaction T in W) **do**
    if (is fading) **then**
        $T.utility = T.utility \times decay$ function d^{λ}
        for (each item it in T) **do**
            Update the utility of it in the transaction by multiplying it with the *decay* function d^{λ}
while (Found high utility itemset X) **do**
    $X.distance = Distance(\overrightarrow{X}, \overrightarrow{Root})$
    $HUIsSet$.add(X)
return *HUIsSet*

Algorithm 9 IsGlobalDrift: Check if there is a global change in a utility stream

Input: List of distance values at checkpoints starting from the last change index: d_1; d_2; ... ; d_n.

Output: State with trend.

1: $\overline{U}_{drift}$: average statistic computed from d_1; d_2; ... ; d_{drift}
2: $\overline{V}$: average statistic computed from d_1; d_2; ... ; d_n
3: $\varepsilon_{\overline{U}_{drift}}$, $\varepsilon_{\overline{V}}$: error bounds by Hoeffding's Inequality
4: **for** (each d_i in the list of distance values) **do**
5: Update $\overline{U}_{drift}$, $\overline{V}$, $\varepsilon_{\overline{U}_{drift}}$, $\varepsilon_{\overline{V}}$
6: **if** ($\overline{U}_{drift} + \varepsilon_{\overline{U}_{drift}} \geq \overline{V} + \varepsilon_{\overline{V}}$) **then** *//This is an increasing trend*
7: Update cut point: $\overline{U}_{drift} = \overline{V}$ and $\varepsilon_{\overline{U}_{drift}} = \varepsilon_{\overline{V}}$
8: **if** ($\overline{U}_{drift}$ - $\varepsilon_{\overline{U}_{drift}} \leq \overline{V}$ - $\varepsilon_{\overline{V}}$) **then** *//This is a decreasing trend*
9: Update cut point: $\overline{U}_{drift} = \overline{V}$ and $\varepsilon_{\overline{U}_{drift}} = \varepsilon_{\overline{V}}$
10: **if** (The hypothesis H_0 is rejected, $|\overline{U}_{drift}$ - $\overline{V}| \geq \varepsilon$ as Eq. 4.8) **then**
11: Output drift with trend (increasing or decreasing) & return drift
12: **else**
13: Output Stable State

Algorithm 10 IsLocalDrift: Check if there is a local change in a utility stream

Input: List of checkpoints.

Output: Drift state with itemset.

1: **for** (each check point cp in list *checkpoints*) **do**
2: **for** (each itemset X in cp.HUIs) **do**
3: Split *checkpoints* into two different observations $HS^{n,\lambda}_{W_1}$ and $HS^{m,\lambda}_{W_2}$ at *cp*
4: Calculate the variance of X, σ^2 by Eq. 4.10
5: Calculate the epsilon cut point, ε_α by Eq. 4.9 with Bonferroni correction
6: **if** (The rule to reject the hypothesis H_0, $|\overline{X_1}$ - $\overline{X_2}| \geq \varepsilon_\alpha$) **then**
7: Output local drift of itemset X

is temporarily stored in the limited amount of available memory (line 5). If the memory is full then the oldest transaction(s) are removed to free space for new transaction(s) T_i (line 4). To detect drift, the user must indicate the time interval length at which the algorithm should check for drifts (line

6). When the drift detection algorithm is called, it applies a procedure named HUI-Discovery (Algorithm 8) to mine all high utility itemsets in the window ending at the current observation time (line 7). Then, the similarity and distance of itemsets and the global distance are calculated (line 8). Thereafter, global and local checks are performed (lines 9 and 12) to detect significant changes in the utility distribution.

The HUI-Discovery procedure (Algorithm 8) mines high utility itemsets in a stream. The procedure first creates the window W ending at the current observation time t (line 3). Then, for each transaction in the window W, the procedure multiplies the utilities by the decay factor calculated by the fading function. This decreases the utility of items in the transaction as a function of its age. Then, the HUI-Discovery procedure applies a traditional HUI mining algorithm to extract each HUI in the current window (lines 9 to 11). For each itemset found (line 9), the utility distribution of each item is calculated to construct the itemset's vector and then calculate its distance to the root vector (line 10).

The IsGlobalDrift procedure (Algorithm 9) detects global changes in a stream of values. It is based on probability theory, statistical testing, and the Hoeffding's Inequality. Lines 6-7 detect a cut point for an increasing trend of values, while lines 8-9 detect a cut point for a decreasing trend of values. If a change in the data distribution rejects the null hypothesis H_0 at line 10, a drift is said to occur at the cut point, and it is reported to the user (line 11).

The IsLocalDrift procedure (Algorithm 10) is an algorithm to detect local changes in the utilities of itemsets in a stream. Similar to the IsGlobalDrift procedure, IsLocalDrift is also based on probability theory, statistical testing, and the Hoeffding's Inequality. At line 5, the procedure calculates the epsilon cut point with Bonferroni correction to prevent increasing of incorrectly rejecting a null hypothesis when multiple hypotheses are tested.

Complexity. The proposed method uses a time interval length m to select checkpoints at which drift detection is performed. Thus, the time complexity of the proposed approach is $\mathcal{O}(\frac{n}{m})$, where n is the space size of the stream. The HUI-Discovery algorithm is implemented using a traditional utility-list based algorithm for mining high utilty itemsets. In the worst case, the time complexity of a utility-list based algorithm is $\mathcal{O}(2^{|I^*|})$, where I^* is the set of remaining items in the processing window which have transaction weighted utilities no less than the threshold value. The HUI-Discovery procedure requires only to scan each window twice. It employs an efficient structure *EUCS* and an improved utility list construction method [45] with complexity of $\mathcal{O}(|W|)$, where $|W|$ is the window size. The

IsGlobalDrift and IsLocalDrift procedures have $\mathcal{O}(1)$ space and time complexity at each checkpoint.

4.5 Evaluation

This section presents an experimental evaluation of the performance of our approach.

Experimental Setup

The experiments were carried out on a computer running the Windows 10 operating system, having a 64 bit Intel i7 2.6 GHz processor, and 16 GB of RAM. The algorithms were implemented in Java. In all the experiments, the exponential decay function $2^{-\frac{T-t}{|W|}}$ was used, excepts experiments in Subsection 4.5.5 studying influence of the decay function. The window size was set to the half-life of the decay function (the time needed for the decay function to decrease a utility value by half). The interval parameter is used to monitor changes at each checkpoint. If its value is large, the number of checkpoints is small and the overlap between windows is small. This, in turn, means that the true positive and accuracy values are high, but it may miss some changes. Moreover, the global confidence level controls the error rate. When the value is high, the number of changes will increase with a high error rate. Setting the interval and confidence level follows the approach proposed in the literature [80, 20, 16] and is application-dependent. The utility threshold influences the number of itemsets and is also dataset-dependent. If the threshold is set to a small value, the number of high utility itemsets may reach millions. If the threshold is large, few high utility itemsets are obtained. Here, the threshold has been set empirically.

In the following evaluations, we employed several various settings to show the feasibility of our method. In addition, we carried out experiments to evaluate the effects of different parameter values.

4.5.1 Datasets

In order to show the generality, feasibility, and applicability of our approach, we performed the evaluation both on synthetic and real-world datasets.

Real Datasets

For the real-world experiments, we used the datasets named Chainstore, Accidents, and Kosarak. These datasets are standard benchmark datasets

for utility mining, which were obtained from the SPMF open-source data mining library website[2]. The Chainstore dataset contains real internal and external utilities. For the two other datasets, the internal and external utilities have been generated using a Gaussian distribution in the [1,10] and [1,5] interval, respectively.

Chainstore contains customer transactions from a retail store. The dataset was transformed from the NU-Mine Bench software, and is provided on the SPMF website. Chainstore contains 1,112,949 transactions, with an average transaction length of 7.26 items, and 46,086 distinct items. For this dataset, the checkpoint *interval*, utility *threshold*, and global *confidence* level have been set to 10,000, 600,000, and 0.99, respectively.

Accidents is a traffic accident dataset, often used as a benchmark dataset. It contains 340,183 transactions, 468 distinct items, and having an average transaction length of 33.8 itemsets. For experiments carried out on this dataset, the checkpoint *interval* has been set to 10,000. *window size*, utility *threshold* and global *confidence* level have been initially set to 10,000, 900,000, and 0.99, respectively. These values have then been changed to 15,000, 1,300,000, and 0.8 in the second experiment.

Kosarak is a click-stream dataset of an online Hungarian news portal. It contains 990,000 transactions with 41,270 items, where transactions contain 8.1 items on average. For the experiments on this dataset, the parameters were set to: *threshold* = 200,000, *time interval* = 20,000, and *window size* = 50,000.

Table 4.2: Drift detection in Chainstore

Window size	Check points	Rise	Fall
40,000	108	35	36
50,000	107	38	35

Table 4.3: Drift detection in Accidents

Window size	Check points	Rise	Fall
10,000	34	18	15
15,000	33	14	18

[2]http://www.philippe-fournier-viger.com/spmf/

Table 4.4: Drift detection in Kosarak

α	Check points	Rise	Fall
0.9	47	7	6
0.5	47	4	5

Generated Synthetic Datasets

Because of the lack of appropriate datasets with ground truth, we also performed experiments on synthetic datasets including evaluations influence of parameters, performance of the proposed methods, and comparison to other drift detectors. We used the java open source data mining library SPMF to generated three synthetic datasets, namely StreamT, StreamX, and StreamY. Each dataset contains 50k transactions. These datasets were concatenated multiple times to create a long stream. The reason for using different synthetic datasets and concatenating to form a stream is that the transition points of the resulting stream are known, and can thus be used to evaluate the ability to perform drift detection. Characteristics of the three synthetic datasets are as follows:

StreamT: This dataset contains 50k transactions and 10 distinct items. The average transaction length is 5.5 items. Internal and external utility values were generated in the [1,10] and [1,5] intervals, respectively.

StreamX: This dataset has 50k transactions. The number of distinct items is 15 and the average transaction length is 7.98 items. The [1,15] and [1,10] intervals were used to generate the internal and external utility values of items, respectively, using a Gaussian distribution.

StreamY: This dataset also contains 50k transactions. The average transaction length is 8.02 items, and 20 distinct items are used in this dataset. The internal and external values are generated with the same interval as in *StreamX*.

The stream considered in the following experiments is a concatenation of these three synthetic datasets, obtained by repeating the following pattern 25 times: *StreamT* + *StreamT* + *StreamX* + *StreamY*. This results in a data stream containing 5 million transactions.

Existing Artificial Datasets

Besides our generated synthetic datasets using SPMF library above, three widely-used synthetic data streams, namely Mixed, Sine, and Circles [129, 61], are used in our evaluation in performance evaluation of our drift detec-

tion method. Each stream dataset contains 100,000 instances. The characteristics of these datasets are as follows [64]:

- *Mixed*: This dataset contains four attributes, including two Boolean attributes (v, w) and two numeric attributes (x, y) in the [0, 1] interval. If two of three conditions are satisfied: $v, w, y < 0.5 + 0.3 \times sin(3\pi x)$, the instance is classified as positive. The Mixed dataset contains abrupt concept drifts. Drifts occur at every 20,000 instances with a transition length $\xi = 50$.

- *Sine1*: There are two attributes x and y that are uniformly distributed in the [0, 1] interval. If all points are below the piecewise function $y = sin(x)$, they are classified as positive. The classification is reversed after a change. The Sine1 dataset contains abrupt concept drifts. Drifts occur at every 20,000 instances with a transition length $\xi = 50$.

- *Circles*: This dataset uses four circles to simulate drift concepts. The radius of the circles are 0.15, 0.2, 0.25, and 0.3, respectively. Each instance has two numeric attributes (x, y) on the [0, 1] interval. If an instance is inside the circles, it is classified as positive. The Circles dataset contains gradual concepts drifts. Drifts occur at every 25,000 instances with a transition length $\xi = 500$.

4.5.2 Performance on Real Datasets

Tables 4.2 - 4.4 show the results of drift detection on the Chainstore, Accidents, and Kosarak datasets for various parameter values. In these tables, the columns indicate the window size, the number of check points, the number of increasing drifts that has been detected (denoted as Rise), and the number of decreasing drifts that has been detected (denoted as Fall), respectively. The results show that number of drift points vary slightly as parameters are changed. The number of detected drifts also varies slightly when we change the confidence level α, the size of the window, and the utility threshold to monitor the drifts. For the real world datasets, there is no ground truth to evaluate the detected concept drifts. Therefore, there is no baseline for testing true positive, false negative and delay detection. In this evaluation, the number of drift points is reported, as well as the corresponding trend (increasing or decreasing). Figures 4.2 - 4.5 show visualizations of the utility distribution movements for the three datasets. It can be observed that the utility distribution underlying the data changes gradually and continuously varies for the Chainstore dataset, as new transactions arrive. When the window size is increased, the number of high utility itemsets

and the total utility distribution in each window increases. However, the number of states in the Chainstore stream and the utility have only a slight change. Hence, the number of hits (change report) is stable. For the Accidents datasets, more abrupt changes occurred. The dataset containing the largest percentage of stable states is the Kosarak dataset. On this dataset, we varied the confidence level, which is a probability that influences the prediction failure rate. The results in Table 4.4 show the impact of varying the confidence level. As we can observe in this table, when the confidence value increases, the number of hits increases in both rise and fall states.

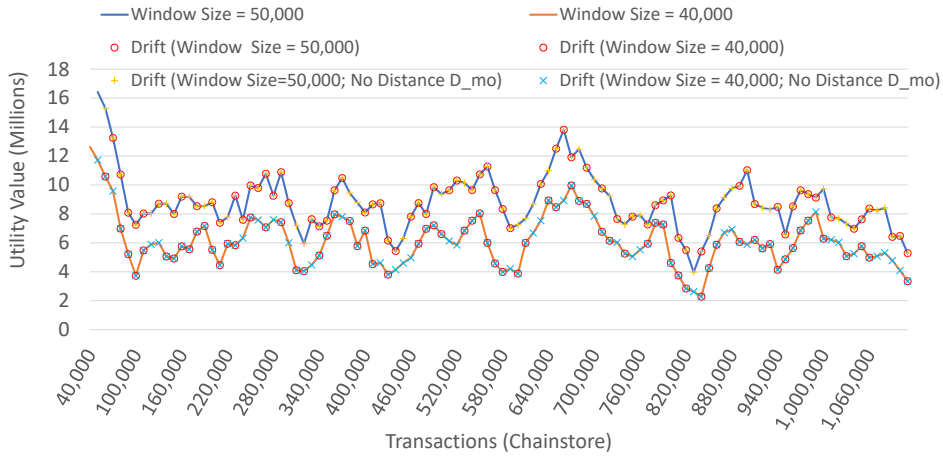


Figure 4.2: Utility distribution on Chainstore.

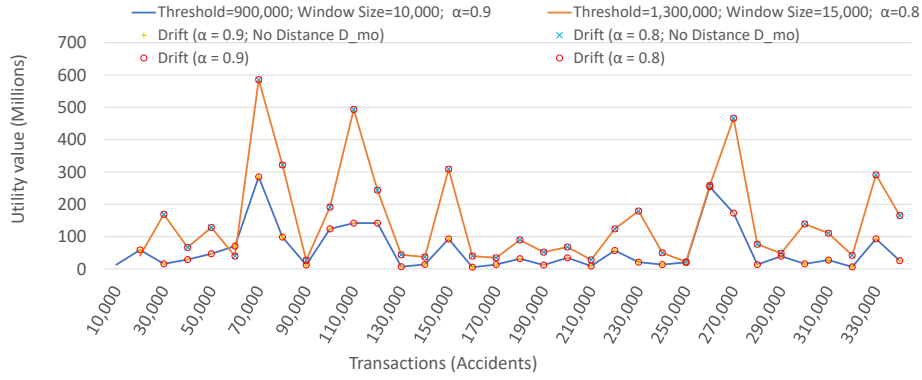


Figure 4.3: Utility distribution on Accidents.

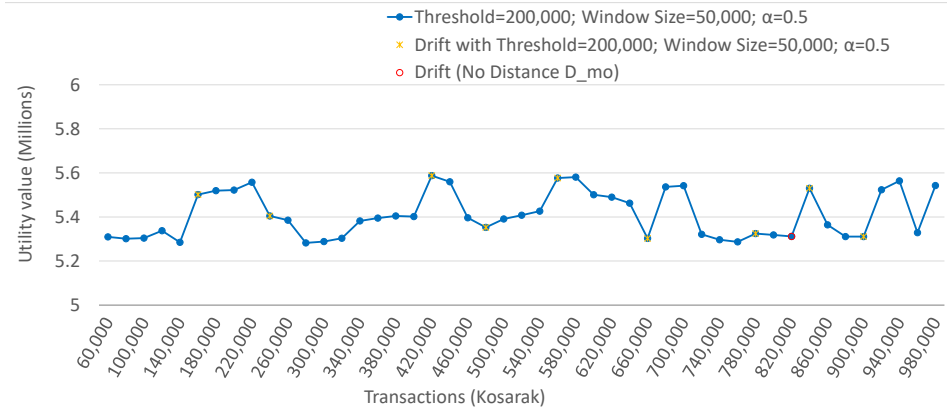


Figure 4.4: Utility distribution on Kosarak and drift points with $\alpha = 0.5$.

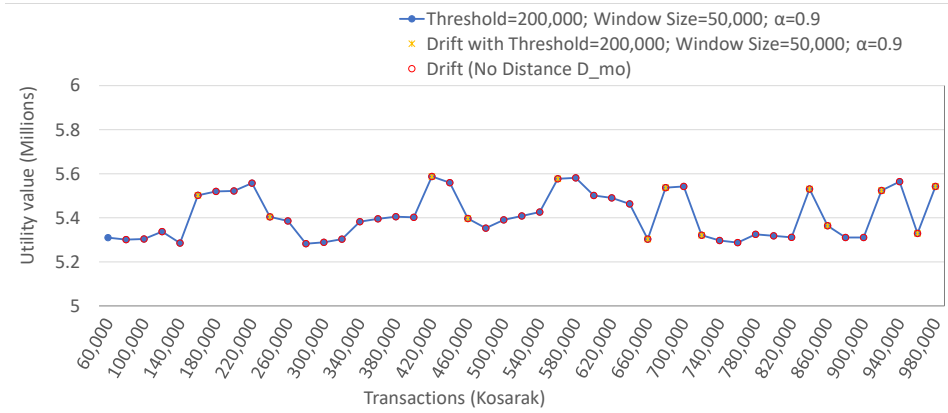


Figure 4.5: Utility distribution on Kosarak and drift points with $\alpha = 0.9$.

4.5.3 Influence of the Confidence Level

We carried out an experiment on the synthetic datasets to evaluate the influence of difference confidence level values. The utility threshold was set to 1,500,000. Both the window size and check point interval were set to 50,000 transactions to ensure that a checkpoint was located at every dataset transition. The confidence level was varied from 0.55 to 1.0.

Table 4.5 shows results when the confidence level α is varied. The result shows that the designed drift detector has a high true positive rate, and low false positive and false negative rates. In particular, when the confidence level α is increased, the accuracy of the approach increases. Generally, for the 1,000 tests performed on the synthetic data streams (a test is performed

Table 4.5: Detection on synthetic datasets

α	FP rate(%)	TP rate(%)	FN rate(%)	Rise	Fall	Time Avg (s)
0.55	0.0	66.2	49.01	25	24	2.773
0.60	0.0	66.2	49.01	25	24	2.725
0.65	0.0	66.2	49.01	25	24	2.721
0.70	0.0	66.2	49.01	25	24	2.709
0.75	0.0	100.0	0.0	25	49	2.721
0.80	0.0	100.0	0.0	25	49	2.690
0.85	0.0	100.0	0.0	25	49	2.570
0.90	0.0	100.0	0.0	25	49	2.583
0.95	0.0	100.0	0.0	25	49	2.580
1.00	0.0	100.0	0.0	25	49	2.563
Summary (tests)				Accuracy rate (%)		Time Avg (s)
1000	0.0%	86.5%	27.7%	89.6%		2.664

every 50k transactions), the true positive, false positive and false negative rates, and the accuracy are 86.5%, 0.0%, 27.7%, and 89.6%, respectively. This result can be explained as follows. When the confidence level is high, at each checkpoint, the detector checks the utility distribution of two successive windows and it reports concept drifts more accurately than for lower confidence levels where the detector probes concept drifts more tightly. In terms of runtime, the proposed approach is very fast, running in less than three seconds for processing windows of 50k transactions. Hence, the results from this experiment show that the detector can be used in an online setting to detect drifts, and that it can quickly adapt itself to changes.

4.5.4 Influence of the Observation Times

In the preceding subsection, an experiment was performed where the transition points were known, and where the proposed algorithm was applied exactly at these transition points, while varying the confidence level α. This section describes a follow-up experiment where the proposed algorithm is not applied exactly at the transition points to see the influence of the observation times. Instead, the checkpoint are gradually moved away from the transition points in the data stream. For this experiment, the confidence level was set to 0.9. The observation times where the change detection algorithm applying was shifted 8 times forward by 25 transactions. As a result,

Table 4.6: Detection on synthetic datasets with shift point

Shift	FP rate(%)	TP rate(%)	FN rate(%)	Rise	Fall	Time Avg (s)
25	0.0	100.0	0.0	25	49	2.744
50	0.0	100.0	0.0	25	49	2.707
75	0.0	100.0	0.0	25	49	2.733
100	0.0	100.0	0.0	25	49	2.768
125	0.0	100.0	0.0	25	49	2.748
150	0.0	100.0	0.0	25	49	2.771
175	0.0	100.0	0.0	25	49	2.726
200	0.0	100.0	0.0	25	49	2.754
Summary (tests)				Accuracy rate (%)		Time Avg (s)
800	0.0%	100.0%	0.0%	100.0%		2.744

windows considered by the algorithm may contain transactions from two datasets. After each shift, the proposed algorithm was executed 100 times for 100 checkpoints.

Table 4.6 shows the results of this experiment, where checkpoints are shifted away from the real transition points of datasets. It is observed that when the shift is increased from 25 to 200 transactions, the detector can detect changes in the utility distribution without making any mistakes. The false positive and false negative rates of the proposed method are in that case equal to zero. The proposed drift detector has high true positive rate and accuracy. In a stream, data evolves and changes over time. Detecting changes within an acceptable delay is crucial. As shown in our experiments, when the number of shifted transactions was non-zero, our detector was able to detect exactly all changes in the utility distribution of the datasets under an acceptable delay, which was defined as 200 transactions for these experiments.

4.5.5 Influence of the Decay Function

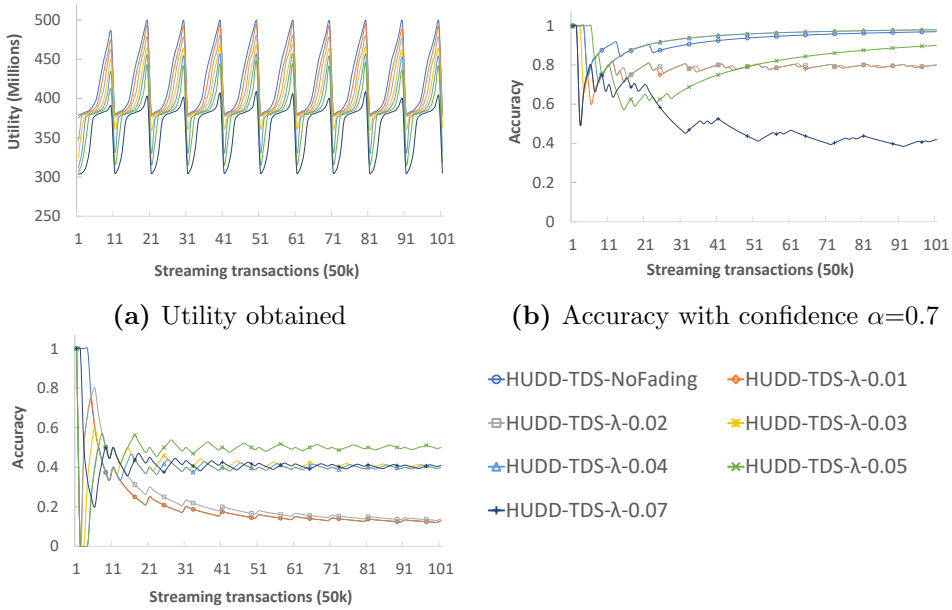
We generated a synthetic stream containing 101 instances. Each instance in the stream has 50k transactions. Thus the stream has 5.05 million transactions. We use the dataset StreamT as our first seed. Each instance in the stream was seeded from its previous instance as follows. We randomly sampled a 10% of transactions in the seed. We added noise to these samples by increasing utility value to 5%. In the stream, the instances at positions

of 11, 21, 31, ... and so on were reset seeded from the StreamT. The utility threshold was set to 1,200,000. In the previous experiments, the window size was used as the half-life of the decay function. To evaluate the influence of the decay function, we multiplied the decay function with a constant λ. The value of λ was set to 0.01, 0.02, 0.03, 0.04, 0.05, and 0.07, respectively. We recorded experiment results of the proposed algorithm without using decay fading and with using different decay values as described above. Figure 4.6a is the utility distribution of high utility itemsets in the stream. Figures 4.6b - 4.6c show the accuracy of the proposed algorithm with the confidence level value set to 0.7 and 0.8 respectively. The results show that the accuracy is gradually stable, and it is affected by decay value. From Figure 4.6a, we can observe that the utility changes slightly when we vary the decay factor. If λ's value is high, the importance of old transactions quickly decreases. Therefore, the weights of past transactions have a minor influence on the utility of itemsets, and the total utility distribution changes more smoothly than for small λ values. When α=0.7, the best accuracy of the proposed detector was obtained with λ=0.03 or λ=0.04. While the best accuracy of the proposed detector was obtained with λ=0.05 if α=0.8. The value of decay function is application-specific and can be chosen either by using a heuristic method.

4.5.6 Evaluation on a Random Stream

The synthetic data stream used in the previous experiments is a repeated concatenation of the following sequence of datasets: *StreamT* + *StreamT* + *StreamX* + *StreamY*. To more extensively evaluate the performance of the designed method, an additional experiment was performed where we concatenated several datasets in different ways to generate a synthetic stream. In this experiment, streams are generated by combining a sequence of datasets formed as $Dataset_1 - Dataset_2 - \cdots - Dataset_k$, where $Dataset_i$ $(1 \leq i \leq k)$ is randomly selected among *StreamT*, *StreamX* and *StreamY*. Each stream includes 100 random instances of datasets and has 5 million transactions. Four random data streams were generated. The designed algorithm was run with a confidence level set to 0.8 and 0.9, and a minimum utility threshold set to 1,500,000 and 1,700,000.

Figures 4.7a - 4.7d show drift points detected by our detector for the four random data streams. In these figures, a red point indicates an observation time where a change in the utility distribution of high utility itemsets was found. At each checkpoint, the proposed algorithm examines the movement in utility distributions in the window consisting of the transactions since the last estimated change point. If the movement is significantly different,



(a) Utility obtained

(b) Accuracy with confidence α=0.7

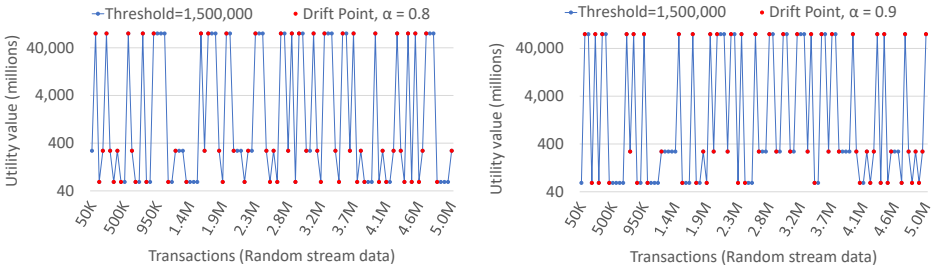
(c) Accuracy with confidence α=0.8

Figure 4.6: Influence of the decay function.

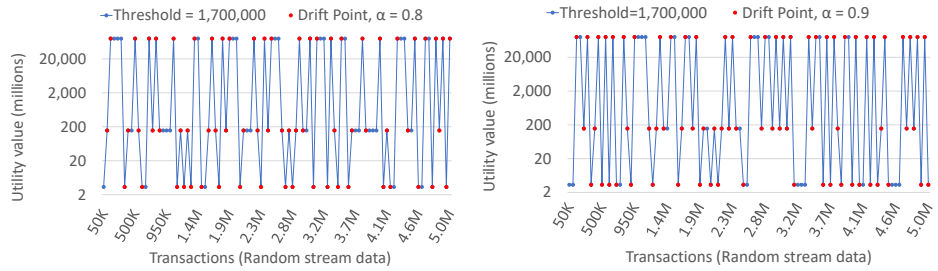
the algorithm marks the checkpoint as a drift point. That checkpoint is then remembered as the last estimated change point. Results have shown that the proposed algorithm can detect exactly all the drift points at the checkpoints, where there is a significant change at the transitions between successive windows.

4.5.7 Evaluation of the Drift Detector for Classification

This subsection presents experiments to evaluate our proposed detector in terms of detection delay, true positive (TP), true negative (TN), false negative (FN), and accuracy with a base incremental classifier applied on a stream. We used Natïve Bayes (NB) and Hoeffding Tree (HT) classifiers as base learners because they are usually used as benchmark classifiers in the literature [18, 61]. Moreover, we compared the results with several state-of-the-art drift detectors, namely EDDM [13], ECCD [134], SeqDrift2 [127], and RDDM [16]. All experiments were performed using the MOA framework [20] with parameters set to the default values for all the compared algorithms, as recommended in the original papers. Note that our drift detector uses a statistical test [61] as global detector. However, the original



(a) *Threshold* = 1,500,000 and $\alpha = 0.8$ **(b)** *Threshold* = 1,500,000 and $\alpha = 0.9$



(c) *Threshold* = 1,700,000 and $\alpha = 0.8$ **(d)** *Threshold* = 1,700,000 and $\alpha = 0.9$

Figure 4.7: Drift points on Random Streams.

detector tests to reject the null hypothesis with an error bound ε of the streaming values from the [0; 1] interval. The proposed detector considers the error bound ε on a real value interval of the streaming data.

In streaming data, to evaluate the measures of concept drift detector, such as true positive, true negative, false negative, and accuracy, the *acceptable delay* length metric [129] is often adopted. Given a threshold Δ, if a detector can detect a change within a delay Δ from the true change point, it is considered as a true positive. Mixed and Sine1 contain abrupt concept drifts, while Circles contains gradual drifts. Therefore, in this experiment, smaller values of acceptable delay are set for Mixed and Sine1 than for Circles. Specifically, the *acceptable delay* is set to 250 on the Mixed and Sine1 datasets, and 1,000 on the Circles dataset.

Table 4.7 shows the average and standard deviation of classification results for the proposed detector, EDDM, ECCD, SeqDrift2, and RDDM running on 100 samples of datasets. We can observe that on the Mixed and Sine1 datasets, ECCD has the shortest detection delay, with high true positive (TP) rates with both Natïve Bayes (NB) and Hoeffding Tree (HT) classifiers. This is because ECCD uses a window containing a small number of instances. However, the false positive (FP) rates are also very high,

Table 4.7: Results with Natïve Bayes(NB) and Hoeffding Tree(HT) classifiers

		Algorithms	Delay	TP	FP	FN	Accuracy	Rank
Mixed dataset	NB	HUDD-TDS	81.75 ± 18.71	3.97 ± 0.17	1.61 ± 1.34	0.03 ± 0.17	**83.26 ± 0.09**	**1**
		RDDM	104.97 ± 12.12	3.99 ± 0.1	1.86 ± 1.66	0.01 ± 0.1	83.24 ± 0.09	2
		SeqDrift2	200.0 ± 0.0	4.0 ± 0.0	4.39 ± 0.79	0.0 ± 0.0	82.91 ± 0.08	3
		ECCD	38.87 ± 24.65	3.81 ± 0.42	142.29 ± 7.90	0.19 ± 0.42	81.00 ± 0.15	4
		EDDM	247.47 ± 8.65	0.11 ± 0.31	20.22 ± 7.70	3.89 ± 0.31	80.30 ± 2.33	5
	HT	HUDD-TDS	68.20 ± 15.44	4.0 ± 0.0	3.41 ± 2.09	0.0 ± 0.0	**83.25 ± 0.14**	**1**
		RDDM	106.68 ± 11.32	3.99 ± 0.1	3.49 ± 2.48	0.01 ± 0.1	83.17 ± 0.12	2
		SeqDrift2	200.0 ± 0.0	4.0 ± 0.0	4.98 ± 1.21	0.0 ± 0.0	82.91 ± 0.11	3
		ECCD	39.76 ± 26.08	3.79 ± 0.46	138.34 ± 7.95	0.21 ± 0.46	80.95 ± 0.15	4
		EDDM	248.46 ± 7.73	0.05 ± 0.22	21.51 ± 7.74	3.95 ± 0.22	80.65 ± 0.82	5
Sine1 Dataset	NB	HUDD-TDS	85.68 ± 24.47	3.96 ± 0.20	1.04 ± 1.06	0.04 ± 0.20	85.94 ± 0.25	2
		RDDM	89.73 ± 16.54	3.99 ± 0.10	3.93 ± 2.92	0.01 ± 0.1	**85.98 ± 0.27**	**1**
		SeqDrift2	200.0 ± 0.0	4.0 ± 0.0	4.26 ± 0.0	0.0 ± 0.58	85.60 ± 0.25	3
		ECCD	33.30 ± 23.22	3.85 ± 0.39	153 ± 8.34	0.15 ± 0.39	84.38 ± 0.14	4
		EDDM	234.28 ± 22.33	0.57 ± 0.64	33.53 ± 11.56	3.43 ± 0.64	83.44 ± 2.88	5
	HT	HUDD-TDS	58.96 ± 13.04	4.0 ± 0.0	2.25 ± 1.56	0.0 ± 0.0	**86.93 ± 0.17**	**1**
		RDDM	93.54 ± 7.82	4.0 ± 0.0	4.72 ± 3.59	0.0 ± 0.0	86.79 ± 0.19	2
		SeqDrift2	200.0 ± 0.0	4.0 ± 0.0	4.83 ± 1.16	0.0	86.53 ± 0.15	3
		ECCD	36.58 ± 25.54	3.8 ± 0.43	153.78 ± 7.67	0.2 ± 0.43	84.28 ± 0.14	5
		EDDM	243.83 ± 22.33	0.22 ± 0.64	33.77 ± 11.56	3.78 ± 0.64	84.71 ± 2.88	4
Circles Dataset	NB	HUDD-TDS	311.18 ± 109.16	2.9 ± 0.3	1.01 ± 0.93	0.1 ± 0.3	84.09 ± 0.12	2
		RDDM	406.50 ± 69.75	2.99 ± 0.1	2.15 ± 1.95	0.01 ± 0.1	84.05 ± 0.12	3
		SeqDrift2	276.67 ± 91.56	2.92 ± 0.27	2.49 ± 0.98	0.08 ± 0.27	**84.13 ± 0.14**	**1**
		ECCD	194.64 ± 158.13	2.84 ± 0.37	174.53 ± 7.62	0.16 ± 0.37	83.18 ± 0.11	4
		EDDM	938.27 ± 107.14	0.35 ± 0.5	31.09 ± 18.23	2.65 ± 0.5	83.12 ± 0.4	5
	HT	HUDD-TDS	242.60 ± 147.48	2.56 ± 0.52	4.71 ± 1.67	0.44 ± 0.52	85.97 ± 0.23	3
		RDDM	293.80 ± 38.72	2.98 ± 0.14	0.79 ± 1.26	0.02 ± 0.14	86.46 ± 0.16	2
		SeqDrift2	202.67 ± 16.19	3.0 ± 0.0	3.08 ± 0.91	0.0 ± 0.0	**86.47 ± 0.14**	**1**
		ECCD	186.40 ± 151.67	2.86 ± 0.35	175.16 ± 8.39	0.14 ± 0.35	83.21 ± 0.12	5
		EDDM	987.75 ± 54.64	0.06 ± 0.24	24.45 ± 14.57	2.94 ± 0.24	84.89 ± 0.29	4
		Average Ranking	HUDD-TDS: **1.67**	RDDM: 2.0	SeqDrift2: 2.33	ECCD: 4.33	EDDM: 4.67	

resulting in a low accuracy. For the Sine1 dataset with NB learner, RDDM discards old instances from the stream and is the most accurate. However, the difference between RDDM and HUDD-TDS is small, and the FP rate of RDDM is higher than the FP rate of HUDD-TDS. In almost all other cases, the proposed detector has the best accuracy and very good flow rates of detection delay, TP, FP, and false negative (FN). The reason for this is that for the Mixed and Sine1 datasets, changes are abrupt, and the proposed detector is an online and non-parametric detector. It estimates precisely the distribution of the stream. Therefore, it can quickly detect points lying out of the distribution boundary. For gradual concept drifts such as in the Circles dataset, SeqDrift2 has the best performance, either using NB or HT learners. The reason is that SeqDrift2 uses a block of 200 instances for reservoir sampling. The block is helpful for the Circles dataset since it contains gradual concepts drifts with a transition length of 500. Nevertheless, FP rate of SeqDrift2 is still high. In most cases, ECCD and EDDM are the worst detectors.

To summaraze, the average ranks of the algorithms are shown in Table 4.7. This shows that the proposed detector is ranked first among the five compared detectors.

4.5.8 Evaluation of Runtime Performance

This section presents the experimental results of the proposed method in terms of runtime consumption on both real-world datasets and synthetic datasets. Figures 4.8 - 4.10 show the runtime of the proposed approach on the real-world datasets including Chainstore, Accidents, and Kosarak. The average runtime is very small, and it is just around a second for performing drift detection and processing windows of 50k transactions at each checkpoint. The average runtimes are respectively around 156(ms), 2.5(s), and 494(ms) on the Chainstore, Accidents, and Kosarak datasets.

On the four random data streams, the proposed approach is quite fast. As presented in Figures 4.11a - 4.11b, it takes around three seconds in average for performing detection in a window of 50k transactions. Figure 4.12 shows average runtime performing detection on random streams while varying window size. At first, window size was set to 50,000 and then was increased 10 times. Each time we increased the size of window by 10,000 transactions. We record the average runtime consumption while utility threshold values are set to 1,500,000 and 1,700,000 respectively, along with and without using the D_{mo} distance. The result shows that the average runtime consumption is small, and it is linear when the size of window increases.

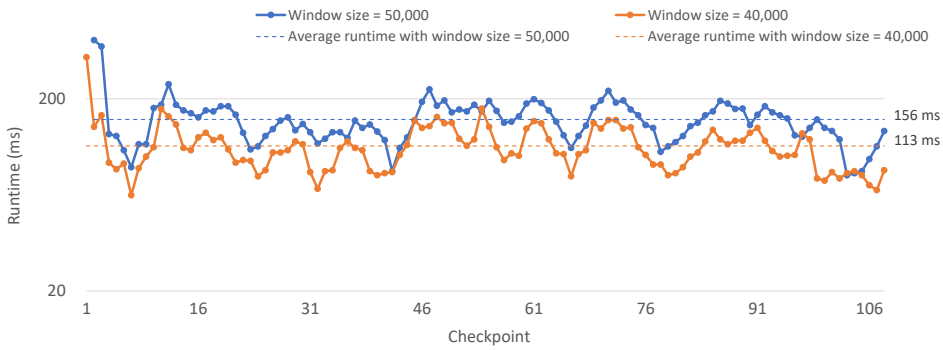


Figure 4.8: Runtime on Chainstore.

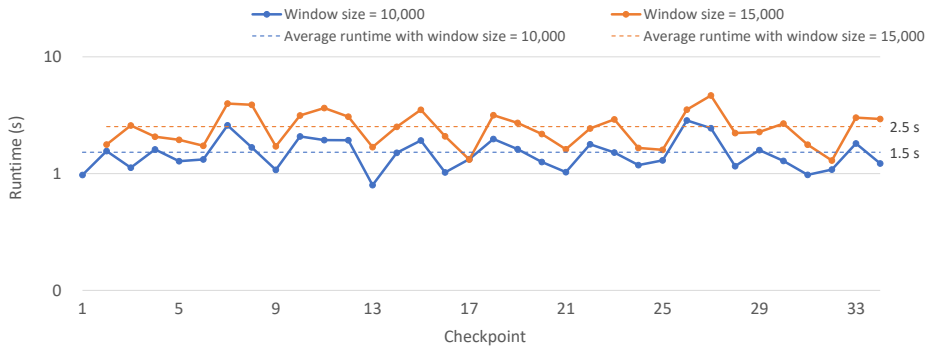


Figure 4.9: Runtime on Accidents.

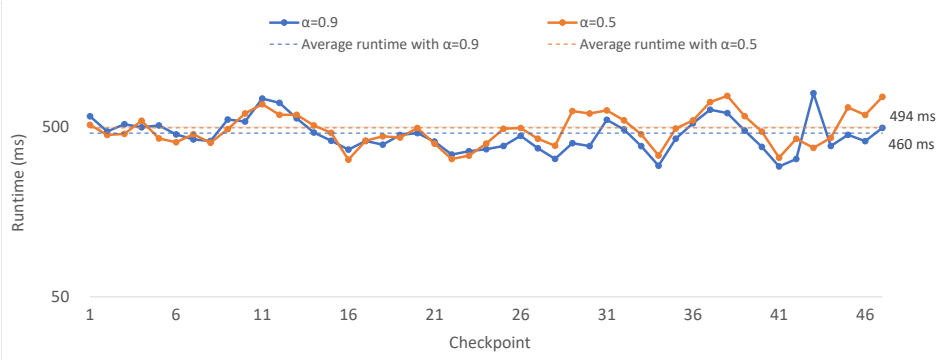
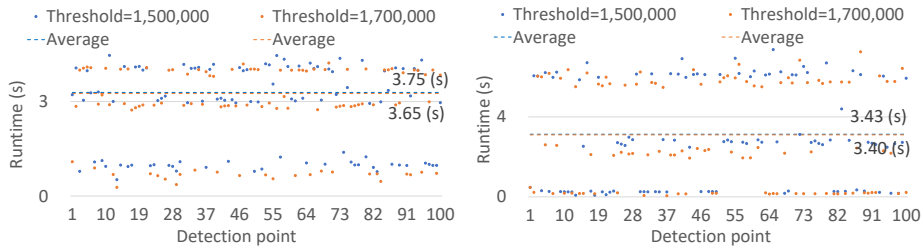


Figure 4.10: Runtime on Kosarak.

In general, our empirical experiments have shown that running time of the proposed algorithm is very small on both real-world datasets and



(a) $\alpha = 0.8$ **(b)** $\alpha = 0.9$

Figure 4.11: Runtime on Random Streams.

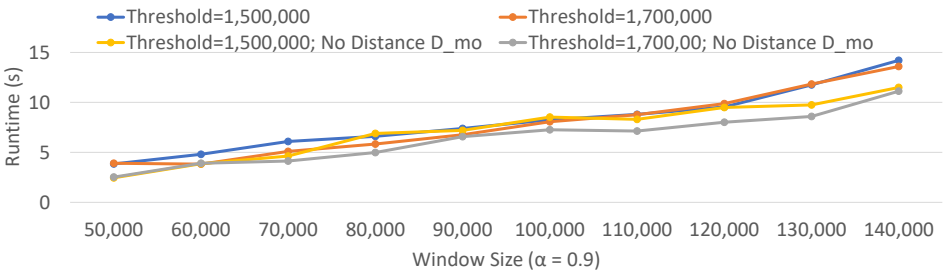


Figure 4.12: Average Runtime on Random Streams.

synthetic datasets under various parameter settings. Furthermore, our detector is online and non-parametric, which makes it suitable for online drift detection from a stream, where concept drifts are short and the method must quickly detect changes with high true positive, high accuracy, and low detection delay.

4.6 Conclusion

In this chapter we presented an algorithm named High Utility Drift Detection in Transactional Data Stream (HUDD-TDS) to detect changes in the utility distributions of itemsets in a stream of quantitative customer transactions. The algorithm utilizes a fading function to quickly adapt to changes in a data stream by placing less importance on older transactions than the recent ones. To ensure that only significant changes are reported to the user, we proposed an approach that uses statistical testing based on the Hoeffding's inequality with Bonferroni correction. The main advantage with our algorithm is that it can detect both local and global utility drifts, i.e., drifts in the utility distribution of single patterns and of multiple high

utility patterns, respectively, in addition to discovering both increasing and decreasing trends. To detect global utility drifts, we proposed a new distance measure, which generalizes the cosine similarity, by also taking the distance between pairs of high utility itemsets into account. To evaluate our approach, we carried out extensive experiments both on real and synthetic datasets. The results from the experiments showed the efficiency of the proposed method. In particular, it can identify both types of changes in the utility distributions of high utility itemsets in real-time, yielding a high true positive rate, while keeping both false positive and false negative at a lowest possible rate.

Chapter 5

Applying Temporal Dependence to Change Detection

The detection of changes in streaming data is an important mining task with a wide range of real-life applications. Numerous algorithms have been proposed for the efficient detection of changes in streaming data. However, the main limitation of existing algorithms is the assumption that data have IID characteristics; these approaches assume that data are generated identically and independently, an assumption rarely holds in real-life streaming environments. In particular, the temporal dependencies of data in a stream have still not been thoroughly studied. This provides the motivation for the work presented in this chapter, in which we address research questions RQ2 to RQ5 [46], i.e., *How can we design efficient models that easily and quickly adapt to the evolution of data? How can we simulate the dependencies of data, and specify the correlations/transitions between data points in a streaming data? What are the mechanisms for the tuning process of these learning models?*

5.1 Motivation

Numerous algorithms have been proposed to detect changes in streaming data [64, 65]. Still, most existing algorithms have been built based on the assumption that streaming data have stable flows, and that they are arriving in the same distribution. In addition, they assume the data to be identically and independently distributed (IID). However, such an assumption hardly holds in real-life streaming environments.

Focusing on non-stationary environments, several learning algorithms have been proposed to overcome the limitation of the IID assumption [104,

92]. The majority of existing approaches have, however, assumed that data in a stream are independently but not identically distributed [21, 178]. Adaptive estimation is among the proposed techniques for handling temporal dependencies of data. For example, in [124, 132, 22, 8], the approaches employed adaptive estimation methodology by considering a so-called forgetting factor. Here, according to a decay function of the forgetting factor, the underlying assumption is that the importance of a data in a stream is inversely proportional to its age. As part of this, cumulative measures of the underlying distribution and estimators are maintained and monitored to detect changes, while new data continuously arrive.

Despite their efficiency with respect to detecting changes in data streams, the underlying assumption of the above methods is that data are generated independent of other data in the same stream. Meanwhile, several empirical experiments, e.g., [178, 17], have shown that there are important temporal dependencies among data points in a stream of data, thus making it crucial for further studies, especially focusing on change detection. Motivated by this, the main goal of this work is to investigate the temporal dependencies of data points in a data stream, and use this to develop a novel method that enables monitoring changes in the stream while continuously estimating the underlying data distribution.

Contributions

In this work, we make the following main contributions:

1. We introduce a new model named *Candidate Change Point (CCP)* that we use to model high-order temporal dependencies and data distribution in streaming data.

2. We develop a new concept called *CCP trail*, which is the path from a given observed data to another specific data in the observation history. Our approach uses the mean value of the CCP trail as a measure of data distribution, and to capture the temporal dependencies among observed data in the stream.

3. We develop a method that is able to handle the fact that data arrives in a high-velocity stream by providing continuously updating estimation factors as part of the CCP model.

4. We propose an efficient real-time algorithm, based on pivotal statistic tests for change detection named Candidate Change Point Detector (CCPD).

5. In order to demonstrate the feasibility, efficiency, and generality of our method, we conduct a thorough evaluation based on several real-life datasets and comparison with related approaches.

Organization

The remainder of this chapter is organized as follows. Section 5.2 briefly reviews the related work. Section 5.3 introduces the problem of change detection. Section 5.4 presents the proposed model and the adaptive estimation method for detecting changes in evolving streaming data. Section 5.5 describes and discusses the results from our experimental evaluation. Finally, Section 5.6 concludes the work.

5.2 Related Work

Hoeffding's inequality [80] is one of the most well-known inequalities. It has been used to design several upper bounds for drift detection [61]. These upper bounds have been used in algorithms such as the Drift Detection Method Based on Hoeffding's Bounds (HDDM) [61], the Fast Hoeffding Drift Detection Method for Evolving Data Streams (FHDDM) [129], Hoeffding Adaptive Tree (HAT) [19], and the HAT + DDM + ADWIN [4] algorithm which extends the ADaptive sliding WINdow (ADWIN) algorithm [18], the Drift Detection Method (DDM) [64], and the Reactive Drift Detection Method (RDDM) [16], which is a recent drift detection algorithm based on DDM. Nevertheless, Hoeffding inequalities have the disadvantage of dropping the dependence on the underlying distribution [161]. Although these algorithms are useful to detect changes in streaming data, most of them assume that data is identically and independently distributed. However, as mentioned earlier, in real-life streaming data, data is inherently dynamic, and is not identically and independently distributed [21, 178]. Also, temporal dependencies are very common in data streams [178]. Thus, to address this issue, temporal dependencies in the streams should be considered.

Regarding temporal dependencies in data streams, adaptive models for detecting changes in the underlying data distribution have been proposed and extensively studied. The main idea of these approaches has been to estimate, maintain some interesting targets in the stream, and then compute the data distributions. The main method used has been based on statistical hypothesis tests, where the hypothesis H_1 means change has been discovered, whereas the null hypothesis H_0 means no change. An example of such an approach has been suggested by Bodenham et al. [22]. They adopted a so-

called forgetting factor, originally suggested by Anagnostopoulos et al. [8], to develop a new approach for continuously monitoring changes in a data stream, using adaptive estimation. They proposed an exponential forgetting factor method to decrease the importance of a data according to a decay function, which is inversely proportional to the age of the observed data.

Although Bodenham's approach enables monitoring changes in data streams by applying so-called Adaptive Forgetting Factor (AFF) [22], it does not address the challenges with temporal dependencies of data. Further, while AFF enables detecting multiple data point changes with adaptive estimation, the approach presented in this chapter focuses on temporal dependencies of data, in addition to supporting online change detection in a data stream.

5.3 Preliminaries

Formally, we define the problem of change detection in streaming data by the following definition. Let there be a data stream S, which is an open-ended sequence of values $\{v_1, v_2, \ldots, v_i, \ldots\}$. Assume that our observation V_i of v_i in the data stream is drawn from a Gaussian model with an unknown distribution, $V_i \sim \mathcal{N}(\mu, \sigma^2 I)$, where μ is the (unknown) mean and is the interest measure in our change detection method, and σ^2 is the error variance. At each observation time i, the observed mean is μ_i. Hence, for a data stream S, we have a sequence of observed mean μ_1, μ_2, ..., μ_i, ... corresponding to observation times 1, 2, ..., i, ... Our change detection method considers the difference between adjacent means μ_i and μ_{i+1}, where not all adjacent means are not necessarily equal. The change detection method is designed to detect all change points i between two observed adjacent means μ_i and μ_{i+1}, for any i and $\mu_i \neq \mu_{i+1}$. Assume that $t_1, t_2, \ldots t_j, \ldots$ are the true change points in a distribution of means. Then, the change detector verifies a change point t_j by testing the following statistical hypotheses:

$$H_0 : \mu_{t_{j-1}} \simeq \cdots \simeq \mu_{t_j-1} \simeq \mu_{t_j} \simeq \mu_{t_j+1} \simeq \cdots \simeq \mu_{t_{j+1}-1}$$

against

$$H_1 : \mu_{t_{j-1}} \simeq \cdots \simeq \mu_{t_j-1} \neq \mu_{t_j} \simeq \mu_{t_j+1} \simeq \cdots \simeq \mu_{t_{j+1}-1}$$

Given a significance confidence ρ, the rule to accept the H_1 hypothesis is if an objective interest measure of mean, i.e., $f(.)$, satisfies:

$$f(\mu_{t_j}) \notin [v_{\frac{\rho}{2}}, v_{1-\frac{\rho}{2}}], \tag{5.1}$$

where $v_{\frac{\rho}{2}}$ and $v_{1-\frac{\rho}{2}}$ are values such that $Pr(f(\mu_{t_j}) < v_{\frac{\rho}{2}}) = \frac{\rho}{2}$ and $Pr(f(\mu_{t_j}) > v_{1-\frac{\rho}{2}}) = 1 - \frac{\rho}{2}$, and $f(.)$ is an objective function. A change point is said to occur at an observation time t_j if H_0 is rejected. This means that the task of detecting change points is to find all the observation times where there are significant differences in the data distribution for a given measure.

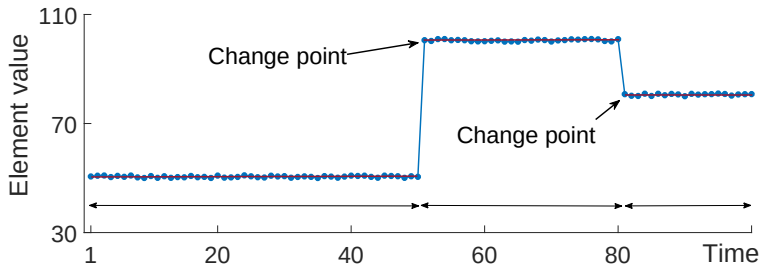


Figure 5.1: An example a stream of data with two change points.

To illustrate, consider a stream of data shown in Figure 5.1, which can be modeled using a piece-wise function with different values depending on the time intervals, $[1, 50]$, $[51, 80]$ and after 81. The estimated mean values are controlled/computed by the function to be 50 in the first interval, 100 in the second and then 80. In conclusion, as also can be observed, significant changes in the stream appear at timestamps 51 and 81.

5.4 Model and Solution

In this section, we propose a new model to represent the temporal dependency of the current observation to its history in the data stream to detect changes.

5.4.1 The Candidate Change Point (Detection) Approach

Most existing work has assumed the streaming data to be identically and independently distributed (i.i.d.). However, focusing on real-life applications, this assumption is too restrictive. In fact, the probability of the current observation is largely depended on previous data and the history of the stream. With this in mind, we consider Markov chain [115] as the natural choice for modeling streaming data, since a Markov process can be used to represent the probability of transitions between states of data in a stream. However, note that the space requirement for maintaining parameters with a higher-order Markov process grows exponentially as functions of the number of pa-

rameters. Thus, approaches employing higher-order Markov processes have a cubic space complexity, and are for this reason inefficient. To cope with this issue, more efficient parameterization approaches, such as the Linear Additive Markov Process (LAMP [96]) and the Retrospective Higher-Order Markov Process (RHOMP [163]), have been proposed. In contrast to the higher-order Markov process, the number of maintained parameters in the LAMP model and the RHOMP model grow linearly, which makes them more suitable for streaming data.

In this work, we build on the ideas of these linear Markov process models to develop an efficient model for temporal dependency in evolving data streams, and use the developed model for detecting concept drifts. We do this based on the fact that the probability of a data to appear in a stream at a specific time is not dependent on the first-order data only. This means that the appearance of data in specific observations can be assumed to be a cause of k previous observations. To be more specific, we propose a novel model, called the candidate change point (CCP) model, for detecting concept drifts. This is done by using the temporal dependency information from previously observed data in the current observation to compute the probability of finding changes in the given observation. What this implies is that with a first-order CCP model, any data that is observed at a specific time has only temporal dependency from the most recent previously observed data in the stream. On the other hand, if the current data point depends on its k previous data points and is a mix of these k data points, then it has a CCP from k-order ancestors. In such a case, the model that we apply is the k-order Candidate Change Point (CCP) model, defined as follows.

Definition 30 (k-order Candidate Change Point)**.** *Given a stream of observations with an open end* $x_1, x_2, \ldots, x_n, \ldots,$ *the* k*-order Candidate Change Point is denoted as* CCP_k*, and at observation time* t*, the CCP model is presented as* $CCP_k(x_t) = (C_1, C_2, \ldots, C_k)$*, where* $0 \leq C_i \leq 1$ *for* $i = 1, 2, \ldots, k$*,* $\sum_{i=1}^{k} C_i = 1$*. We call* C_i *is the probability proportion of CCP in current descendant obtained from* i*-th order in its* k *ancestors.*

Definition 31 (k-order Fading Candidate Change Point)**.** *Given a stream of observations with an open end* $x_1, x_2, \ldots, x_n, \ldots,$ *the* k*-order Fading Candidate Change Point is the* k*-order Candidate Change Point considering forgetting factor* α *of ancestors by order denoted as* $CCP_{k,\alpha}$*. At observation time* t*, the CCP model is presented as* $CCP_{k,\alpha}(x_t) = (\alpha_1 \times C_1, \alpha_2 \times C_2, \ldots, \alpha_k \times C_k)$*, where* $0 \leq \alpha_i \leq 1$ *for* $i = 1, 2, \ldots, k$*,* $\sum_{i=1}^{k} \alpha_i = 1$*,* $0 \leq C_i \leq 1$ *for* $i = 1, 2, \ldots, k$*,* $\sum_{i=1}^{k} C_i = 1$*.*

We call α_i a decaying probability of CCP in current descendant from i-th ancestor in its k ancestors into the history context. Given a scalar x, we denote $\overrightarrow{x_{CCP_k}}$ as a vector of x projected on its k-order ancestors. We have $\overrightarrow{x_{CCP_k}} = (C_1, C_2, \ldots, C_k)$, it is the k-order Candidate Change Point of the scalar. In the rest of the chapter, we use a bold face letters ($\mathbf{x}$) to present the vector for short ($\overrightarrow{x_{CCP_k}}$). Suppose $\mathcal{L}$ be a cost function, the value and option of cost function $\mathcal{L}$ will be discussed in the next section. In this work, our purpose is to solve the minimum of cost function subject to conditions ℓ_1-norm constraint on k-order Candidate Change Point. The problem is described as:

$$\underset{CCP_k}{minimize}(\mathcal{L}(CCP_k)), \tag{5.2}$$

subject to $0 \leq C_i \leq 1, \forall i = 1, 2, \ldots, k, \sum_{i=1}^{k} C_i = 1$.

Projected (sub)gradient methods minimize an objective function $f(x)$ subject to the constraint that x belongs to a convex set $\mathcal{CS}$. The constrained convex optimization problem is $minimize f(x)$ subject to $x \in \mathcal{CS}$.

The projected (sub)gradient method is given by generating the sequence $x^{(t)}$ via:

$$x^{(t+1)} = \prod_{\mathcal{CS}}(x^{(t)} - \gamma_t \times g^{(t)}), \tag{5.3}$$

where $\prod_{\mathcal{CS}}$ is a projection on $\mathcal{CS}$, γ_t is step size, $x^{(t)}$ is the t-th iteration, and $g^{(t)}$ is any (sub)gradient of f at $x^{(t)}$, and will be denoted as $\partial f(x)/\partial x^{(t)}$.

5.4.2 Evolving Data and CCP Parameter Selection

Given a data stream where data evolves over time, i.e., its population distribution or its structure changes over time, the goal is to maximize the probability proportion of CCP for current observation in the set of the k last/previously observed data by solving an optimal problem, using the projected (sub)gradient method with an optimal function to minimize the divergence of the currently observed data when it is projected onto ℓ_1-norm constraints of k previous ancestors. Here, the cost function $f(x)$ on Euclidean projections can be chosen as Euclidean norm (ℓ_2 norm) $\mathcal{L}(x) = \frac{1}{2} \parallel x - v \parallel^2$.

The Euclidean projection in this work is to project a streaming data x^t onto a set of k previously observed data, defined as:

$$\prod_{CCP_k}(x^t) = \underset{\mathbf{x} \in CCP_k}{\arg\min} \frac{1}{2} \parallel \mathbf{x} - \mathbf{x}^t \parallel^2, \tag{5.4}$$

subject to $\| \mathbf{x} \|_1 = \sum_{i=1}^{k} C_i = 1$. The optimal solution for this problem can be solved in linear time as in [43, 109, 32]. The Lagrangian of Eq. 5.4 with Karush Kuhn Tucker (KKT) multiplier ζ and μ is:

$$\mathcal{L}(x, \zeta) = \frac{1}{2} \| \mathbf{x} - \mathbf{x}^t \|^2 + \zeta(\| \mathbf{x} \|_1 - 1) - \mu \mathbf{x} \tag{5.5}$$

Derivation of Lagrangian function at x_i with optimal solution x^* by dimension i is $\frac{\partial \mathcal{L}}{\partial x_i}(x^*) = (x_i^* - x_i^t) + \zeta - \mu_i$. Because x^* is an optimal point then $\frac{\partial \mathcal{L}}{\partial x_i}(x^*) = 0 \rightarrow x_i^* = x_i^t - \zeta + \mu_i$. The KKT inequalities in the problem is that $x_i \geq 0$, then by the complementary slackness, we have $\mu_i = 0$, hence $x_i^* = max\,(x_i^t - \zeta, 0)$. Moreover, we project our vector using a fast and linear method as in [32]. In particular, we consider a vector of the last k data points as the vector of current observation with k elements. This k elements vector will be projected onto ℓ_1 ball constraints by using (sub)gradient method.

Definition 32 (CCP Heritage)**.** *Given a* k*-order Candidate Change Point with forgetting factor α, the CCP model is $CCP_{k,\alpha}(x_t) = (\alpha_1 \times C_1, \alpha_2 \times C_2, \ldots, \alpha_k \times C_k)$, where $0 \leq \alpha_i \leq 1$ for $i = 1, 2, \ldots, k$, $\sum_{i=1}^{k} \alpha_i = 1$, $0 \leq C_i \leq 1$ for $i = 1, 2, \ldots, k$, and $\sum_{i=1}^{k} C_i = 1$. The CCP Heritage of the model at time* t *is denoted as $CH_{k,\alpha}(x_t)$ and computed as $CH_{k,\alpha}(x_t) = \sum_{i=1}^{k} \alpha_i \times C_i$. This value of the scalar can be used to present the temporal dependency proportion of the scalar to history. For the sake of brevity, the notation ch_t will be used to refer to the CCP heritage value at observation time* t *in the rest of this chapter. The sequence of CCP heritage of the stream is denoted $ch_1, ch_2, \ldots, ch_t, \ldots$.*

In this work, the key idea is to investigate how to exploit the temporal dependencies of data for detection of changes in a stream. The method we propose is inspired by the ideas behind linear Markov process models with which the main principle is to estimate the probability of transition between states, i.e., to determine the likelihood of each state transition. Nevertheless, the main difference between our approach and previous Markov-based approaches is that our approach considers the temporal dependencies based on several prior observations. In addition, the proposed model enables estimating the dependency measures in a data stream in an online fashion, thus making it possible to detect changes in the stream in real time.

5.4.3 Adaptive Estimation of Data in Streaming

Adaptive estimation approaches were previously proposed to handle issues with the uncertainty of data. In streaming data, the importance of historical

data is weighted using a forgetting function with a decay factor [8, 22, 142]. This means that, in the stream, the most recent data is more important than the older ones. A decay function is also useful to flush any noise when detecting concept drifts.

There are several approaches to select the decay factor α. Once the value can be set to constant, then we can conduct trial experiments and obtain the optimal value for individual application. The factor can be any function of exponential family of distribution. In [22], the authors proposed adaptive forgetting factor to weigh the importance of historical data by solving an optimal problem in movement of mean. The estimator is continuously monitored when new data arrives to detect changes in the data stream. The principle behind building forgetting factor is to build an exponential decay function of observation time such that the importance of a data in a stream is inversely proportional to its age, and the temporal dependencies can be seen is just 1-order dependency, which is a special case, with k=1 and $\alpha = 1$. In the proposed method, we introduce an adaptive estimation for detecting changes, we use CCP heritage of the CCP model to estimate the CCP mean distribution of streaming data. This method can be compared to the linear high order of states based on Markov chain [163, 96].

To adapt the evolving factor in streaming data, we introduce CCP trail to denote the CCP path from a given observation to another previous observation in the streaming data. Hence, a CCP trail can be considered as the probability of finding the heritage from a data stream. This is formally defined as follows.

Definition 33 (CCP Trail)**.** *Given two different observation times t_1 and t_2, $0 < t_1 \leq t_2$, of a streaming data S, which is an open-ended sequence of values $\{v_1, v_2, \ldots, v_i, \ldots\}$. The CCP trail between two given observations is the probability of finding the heritage of the data at observation time t_1 in the data at observation time t_2. A CCP trail is denoted as $ct(t_2, t_1)$, and is formally defined as:*

$$ct(t_2, t_1) = \begin{cases} 1 & \text{if } t_2 = t_1 \\ ch_{t_2} & \text{if } t_2 = t_1 + 1 \\ ct(t_2, t_2 - 1) \times ct(t_2 - 1, t_1) & \text{otherwise.} \end{cases} \tag{5.6}$$

Property 9. *The CCP trail can be computed by as the product of CCP heritages for all $t_1 \leq t_2$ as follows:*

$$ct(t_2, t_1) = \prod_{t=t_1}^{t_2} (1_{\{t \neq t_1\}} \times ch_t + 1_{\{t=t_1\}}), \tag{5.7}$$

where $1_{\{x\}}$ is binary indicator function. In other words, $1_{\{x\}}$ is equal to 1 if x is TRUE, otherwise $1_{\{x\}}$ is equal to 0.

$$1_{\{x\}} = \begin{cases} 1 & \text{if x is TRUE} \\ 0 & \text{otherwise.} \end{cases} \tag{5.8}$$

Proof. This property can be easily proved by induction. When $t_2 = t_1$, we have:

$$\begin{aligned} \prod_{t=t_1}^{t_2} (1_{\{t \neq t_1\}} \times ch_t + 1_{\{t=t_1\}}) &= 1_{\{t_1 \neq t_1\}} \times ch_{t_1} + 1_{\{t_1=t_1\}} \\ &= 0 \times ch_{t_1} + 1 = 1 \\ &= ct(t_1, t_1) = ct(t_2, t_1). \end{aligned}$$

When $t_2 = t_1 + 1$, we have:

$$\begin{aligned} \prod_{t=t_1}^{t_2} 1_{\{t \neq t_1\}} \times ch_t + 1_{\{t=t_1\}} &= \prod_{t=t_1}^{t_1+1} 1_{\{t \neq t_1\}} \times ch_t + 1_{\{t=t_1\}} \\ &= ch_{t_1+1} = ct(t_1 + 1, t_1) \\ &= ct(t_2, t_1). \end{aligned}$$

Assume that Eq. 5.7 is satisfied when $t_2 = t_1 + m$, with $m \in \mathbb{N}, m > 0$. We prove that Eq. 5.7 is also satisfied with $t_2 = t_1 + m + 1$. We have:

$$\begin{aligned} ct(t_2, t_1) &= ct(t_1 + m + 1, t_1) \\ &= ct(t_1 + m + 1, t_1 + m) \times ct(t_1 + m, t_1) \\ &= ch_{t_1+m+1} \times \prod_{t=t_1}^{t_1+m} (1_{\{t \neq t_1\}} \times ch_t + 1_{\{t=t_1\}}) \\ &= \prod_{t=t_1}^{t_1+m+1} (1_{\{t \neq t_1\}} \times ch_t + 1_{\{t=t_1\}}). \end{aligned}$$

□

CCP trail mean at observation time t in the data stream is then defined by:

$$\overline{CCP(t)} = \frac{1}{\sum_{i=1}^{t} ct(t, i)} \sum_{i=1}^{t} v_i \times ct(t, i) = \frac{cp(t)}{ctsum(t)}, \tag{5.9}$$

where $cp(t) = \sum_{i=1}^{t} v_i \times ct(t, i)$, $ctsum(t) = \sum_{i=1}^{t} ct(t, i)$.

$cp(t)$ is the CCP propagation at observation time t looking back into its history in the data stream. $ctsum(t)$ is the coefficient presenting sum of the CCP trail at time t.

Proposition 4. *Given a data stream, $ctsum(t)$ can be computed by the following equation:*

$$ctsum(t) = ctsum(t-1) \times ch_t + ct(t,t). \tag{5.10}$$

Proof. We have:

$$\begin{aligned} & ctsum(t-1) \times ch_t + ct(t,t) \\ = \; & ct(t,t) + ch_t \times \sum_{i=1}^{t-1} ct(t-1,i) \\ = \; & ct(t,t) + \sum_{i=1}^{t-1} ch_t \times ct(t-1,i) \\ = \; & ct(t,t) + \sum_{i=1}^{t-1} ct(t,t-1) \times ct(t-1,i) \\ = \; & ct(t,t) + \sum_{i=1}^{t-1} ct(t,i) = \sum_{i=1}^{t} ct(t,i) = ctsum(t). \end{aligned}$$

□

Proposition 5. *Similar to the coefficient estimation, the CCP propagation can be estimated by the following sequential updating:*

$$cp(t) = cp(t-1) \times ch_t + v_t. \tag{5.11}$$

Proof. We have:

$$\begin{aligned} cp(t) \; = \; & \sum_{i=1}^{t} v_i \times ct(t,i) = \sum_{i=1}^{t} v_i \prod_{j=i}^{t} (1_{\{j \neq i\}} \times ch_j + 1_{\{j=i\}}) \\ = \; & \sum_{i=1}^{t-1} v_i \prod_{j=i}^{t} (1_{\{j \neq i\}} \times ch_j + 1_{\{j=i\}}) + v_t \prod_{j=t}^{t} (1_{\{j \neq t\}} \times ch_j + 1_{\{j=t\}}) \\ = \; & v_t + \sum_{i=1}^{t-1} v_i (1_{\{t \neq i\}} \times ch_t + 1_{\{t=i\}}) \times \prod_{j=i}^{t-1} (1_{\{j \neq i\}} \times ch_j + 1_{\{j=i\}}) \\ = \; & v_t + ch_t \sum_{i=1}^{t-1} v_i \prod_{j=i}^{t-1} (1_{\{j \neq i\}} \times ch_j + 1_{\{j=i\}}) \\ = \; & v_t + ch_t \times cp(t-1). \end{aligned}$$

□

Eqs. 5.10-5.11 show that we can sequentially update the CCP model, CCP coefficient, and CCP propagation in the stream at each observation time, while new data arrives. Hence, the *CCP trail mean* at the next observation, $\overline{CCP(t+1)}$, is easily estimated in linear time by Eq. 5.9.

5.4.4 Change Detection

Change point detection relies on the null hypothesis H_0 and the alternative hypothesis H_1. The null hypothesis H_0 is a hypothesis that assumes the population means are drawn from the same distribution while the alternative hypothesis H_1 supposes the observations are from the different distribution. The change detector defines a rule to accept H_1 and reject H_0. When H_0 is rejected, it means that there is a significant movement in underlying distribution of data, and change point occurs. In our approach, the detector monitors the movement in CCP trail mean in an online manner.

Given a random variable v, we consider the Gaussian model with mean θ, variance σ^2, and assume that $v \sim \mathcal{N}(\theta, \tau^2 = \sigma^2 I)$. The cumulative distribution function [156] for a linear contrast $p^T\theta$ of mean θ is as follows:

$$\mathcal{D}^{[v_1,v_2]}_{p^T\theta,\tau^2}(p^T v)|Gaussian \sim Uniform(0,1), \tag{5.12}$$

where $[v_1, v_2]$ is the boundary interval of the Gaussian model and $\mathcal{D}$ is pivotal statistic function. The pivotal statistic function $\mathcal{D}$ is defined and computed as:

$$\mathcal{D}^{[v_1,v_2]}_{\mu,\tau^2}(x) = \frac{CDF(\frac{(x-\mu)}{\tau}) - CDF(\frac{(v_1-\mu)}{\tau})}{CDF(\frac{(v_2-\mu)}{\tau}) - CDF(\frac{(v_1-\mu)}{\tau})}, \tag{5.13}$$

where *CDF(.)* is a standard normal cumulative distribution function, and in our proposed method we use the standard normal right tail probabilities as in [26] due to its simple form, and it has a very small error. We use the truncated Gaussian pivot [156] to test hypothesis H_0 with an assumption that the population distribution is equal to zero, $H_0 : p^T\theta = 0$. The alternative positive hypothesis is $H_{1+} : p^T\theta > 0$. The truncated Gaussian statistic then is computed by: $\mathcal{T} = 1 - \mathcal{D}^{[v_1,v_2]}_{0,\tau^2}(p^T v)$. Given a confidence value $0 \leq \rho \leq 1$, in our change detector, we find the v_ρ satisfying

$$1 - \mathcal{D}^{[v_1,v_2]}_{v_\rho,\tau^2}(p^T v) = \rho \rightarrow \mathbb{P}(p^T\theta \geq v_\rho) = 1 - \rho. \tag{5.14}$$

Similar to the two-sided test, we compute the confidence interval $[v_{\frac{\rho}{2}}, v_{1-\frac{\rho}{2}}]$. $v_{\frac{\rho}{2}}$ and $v_{1-\frac{\rho}{2}}$ are computed based on conditions such that:

$$1 - \mathcal{D}^{[v_1,v_2]}_{v_{\frac{\rho}{2}},\tau^2}(p^T v) = \frac{\rho}{2}, \tag{5.15}$$

$$1 - \mathcal{D}^{[v_1,v_2]}_{v_{1-\frac{\rho}{2}},\tau^2}(p^T v) = 1 - \frac{\rho}{2}. \tag{5.16}$$

Then we have $\mathbb{P}(v_{\frac{\rho}{2}} \leq p^T\theta \leq v_{1-\frac{\rho}{2}}) = 1 - \rho$. Change is identified with a confidence ρ if the pivotal population mean does not lie in the interval $[v_{\frac{\rho}{2}}, v_{1-\frac{\rho}{2}}]$.

5.4.5 Choosing Decay Factor

In our method, we consider forgetting factor α of k previous ancestors by order to the current model. The meaning of α is similar to the decaying probability of looking back into the history in [163]. To select the parameter α, we use truncated form of the geometric distribution [28]. The truncated form of the geometric distribution is presented by parameter η, subject to $0 < \eta \leq 1$, and k terms (ancestors). The probability density function at term $i, 1 \leq i \leq k$, is defined by $P(i) = \frac{\eta(1-\eta)^{i-1}}{1-(1-\eta)^k}$. We set this probability density at term i as our forgetting factor α_i of the i-th ancestor,

$$\alpha_1 = \tfrac{\eta}{1-(1-\eta)^k},\ \alpha_2 = \tfrac{\eta(1-\eta)}{1-(1-\eta)^k},\ \ldots,\ \alpha_k = \tfrac{\eta(1-\eta)^{k-1}}{1-(1-\eta)^k}.$$

Observe that the truncated form of the geometric distribution subjects to the condition that sum of probability of k terms equals to 1. In other words, it subjects to $\sum_{i=1}^{k} \alpha_i = 1$.

$$\begin{aligned}
\sum_{i=1}^{k} \alpha_i &= \frac{\eta}{1-(1-\eta)^k} + \frac{\eta(1-\eta)}{1-(1-\eta)^k} + \cdots + \frac{\eta(1-\eta)^{k-1}}{1-(1-\eta)^k} \\
&= \frac{\eta}{1-(1-\eta)^k}(1 + (1-\eta) + \cdots + (1-\eta)^{k-1}) \\
&= \frac{(1-(1-\eta))(1 + (1-\eta) + \cdots + (1-\eta)^{k-1})}{1-(1-\eta)^k} \\
&= \frac{1-(1-\eta)^k}{1-(1-\eta)^k} = 1.
\end{aligned}$$

Furthermore, the condition $0 \leq \alpha_i \leq 1$ is also satisfied. The value of η is application-specific and can be chosen either by a procedure of polynomial interpolation or using a heuristic optimal as in [163].

5.4.6 The Online Change Detection Algorithm

Figure 5.2 gives an overview of the CCPD algorithm, and our method for change detection is presented in Algorithm 11. The input of the algorithm is a sequence of open end values $v_1, v_2, \ldots, v_i, \ldots$, and a confidence value ρ. When a new value v_i is read from the stream, a linear projection onto ℓ_1 ball constraints is performed to project a vector of the k last data values to get CCP_k of the current observation in line 3 (see [32] for more detailed information about the projection method). Line 4 computes ch_i using the truncated form of the geometric distribution. Line 5 is executed to estimate $ctsum(i)$ and $cp(i)$. Then CCP trail mean $\overline{CCP(i)}$ is computed by line 6. In line 7, the boundary interval with confidence ρ is calculated using the pivotal statistic function. The pivotal truncated Gaussian value of the current data is specified in line 8. A check is performed in line 9 to determine there is a change or not. If a change occurs, the estimators are reset and the algorithm outputs that change.

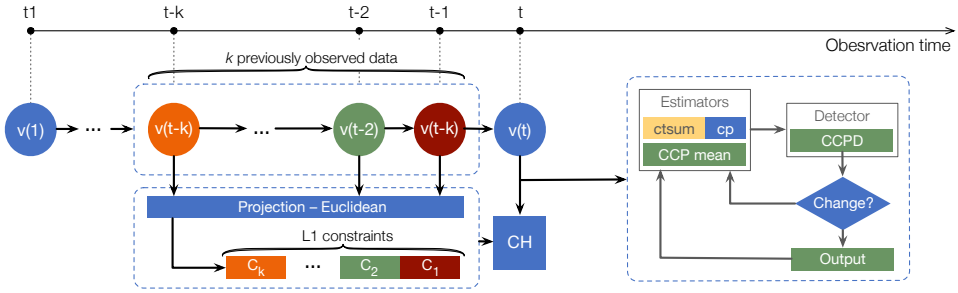


Figure 5.2: Flow diagram of the CCPD algorithm

An illustrative example. Let us consider a sample synthetic data stream presented in Figure 5.1. The stream contains 100 elements with significant changes appearing at timestamps 51 and 81. Assume our parameter $k = 3$. In this example, we include the last data in the projection. Considering the observation at timestamp 50, the values of estimated parameters at timestamp 50 are as follows: ch=0.487, $ct(ctsum)$=1.493, cp=75.372, and $\overline{CCP}$ is computed as $\frac{cp}{ct} = \frac{75.372}{1.493} = 50.484$. Assume that the incoming element in the stream at timestamp 50 is 50.45 and the three latest elements in the stream are sequentially 50.45, 50.66, and 50.08. The processing steps of the proposed method are the following:

Step 1: Project a vector of the last 3 elements (50.45, 50.66, 50.08) on ℓ_1 constraints.

Algorithm 11 CCPD: Online change detector

Input: Streaming data and a confidence ρ.
Output: Changes in the stream.

1: *Init*() Initialize all variables
2: **for** (each data point v_i arriving on a stream $v_1, v_2, \ldots, v_i, \ldots$) **do**
3: Project the k last data points on ℓ_1 constraints $Projection(v_{i-1}, \ldots, v_{i-k}) \rightarrow CCP_k(v_i)$ as in [32]
4: Compute $ch_i = CH_{k,\alpha}(v_i) = \sum_{i=1}^{k} \alpha_i \times C_i$
5: Update estimators $ctsum(i)$ and $cp(i)$ by Eqs. 5.10 and 5.11
6: Update CCP trail mean $\overline{CCP(i)}$ by Eq. 5.9
7: Compute the confidence interval $tailarea = [v_{\frac{\rho}{2}}, v_{1-\frac{\rho}{2}}]$ such that 5.15 - 5.16 satisfy
8: Compute truncated Gaussian pivot of the current observation $pval(i)$
9: **if** $(pval(i) \notin tailarea))$ **then**
10: Output change
11: Reset Estimator

The result of this projection is (0.39, 0.59, 0.02).

Step 2: Calculate the value of CCP Heritage, ch.
Assume the parameter η is set to 0.98, the decay factors $\alpha_i, i = 1, \ldots, 3$, are computed to be (0.98, 0.0196, 0.0004), and the new value of ch is $0.39 * 0.98 + 0.59 * 0.0196 + 0.02 * 0.0004 = 0.394$.

Step 3: Calculate values of estimators ct and cp.
$cp = 75.372 * 0.394 + 50.45 = 80.147$,
$ct = 1.493 * 0.394 + 1 = 1.588$.

Step 4: Update mean $\overline{CCP} = \frac{cp}{ct} = \frac{80.147}{1.588} = 50.470$.

Step 5: Compute the pivotal truncated Gaussian of the new mean, and the confidence interval of the old mean with a confidence $\rho = 0.01$.
We have $pval(\overline{CCP}) = 1.0$, and $tailarea = [v_{\frac{\rho}{2}}, v_{1-\frac{\rho}{2}}] = [0.0, 1.0]$.

Step 6: Check condition $pval(i) \notin tailarea$.
Because $1.0 \in [0.0, 1.0]$, the detector determines that there is no change at timestamp 50.

Step 7: A new data in the stream at timestamp 51 can now be processed. At the timestamp the data value is 100.56, so the last 3 elements in the stream are (100.56, 50.45, 50.66). Repeat Step 1 to Step 6 to process the rest of the stream.

When new data comes in the stream, if a change is detected, the values of the estimators are reset to default. In our example, with the sample synthetic dataset, the CCPD detects two change points at timestamps 52 and 82 (1 delay in comparison with two true change points at timestamps 51 and 81).

Figure 5.3 and Figure 5.4 depict the visualization of the real values of the stream and the estimated mean values by the CCPD method, running on the sample synthetic streaming data shown in Figure 5.1. In particular, Figure 5.3 plots the real values and the estimated mean values computed using the CCPD method with two true change points at timestamps 51 and 81, and two detected change points at timestamps 52 and 82. Figure 5.4 shows the estimated mean values while we vary the value of η in the decay factor. The importance degree of the current data is the combination of the projection vector (on k latest data) and the decay factors, which are, in turn, affected by the choice of η. Because we chose decay factors in a form of geometric distribution, the most effect on the importance of current data can be derived from the most recent data in the stream. We observe that when the stream is stable, the value of the estimator is stable with different values of the decay factor. However, the obtained estimator significantly changes around the change point.

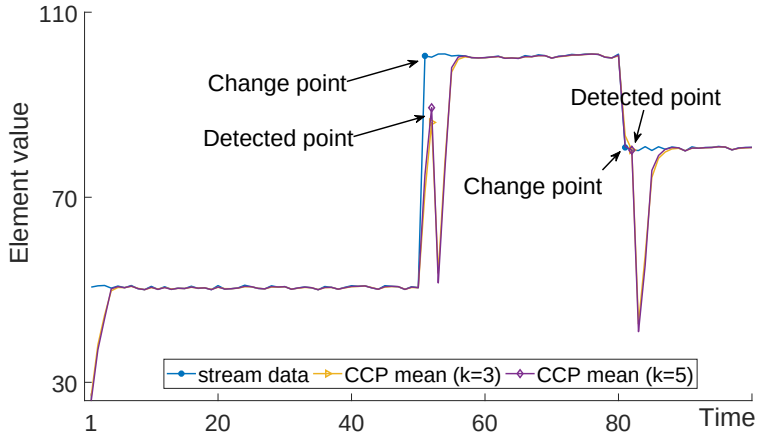


Figure 5.3: A stream of data with two detected change points.

Complexity. The procedure of processing each arrival stream data point has three main parts. The first part is the Euclidean projection of the k last data points on a ℓ_1 ball constraints. This process is a fast projection [32] and has complexity $\mathcal{O}(k)$. Normally, k is small, thus the process can be

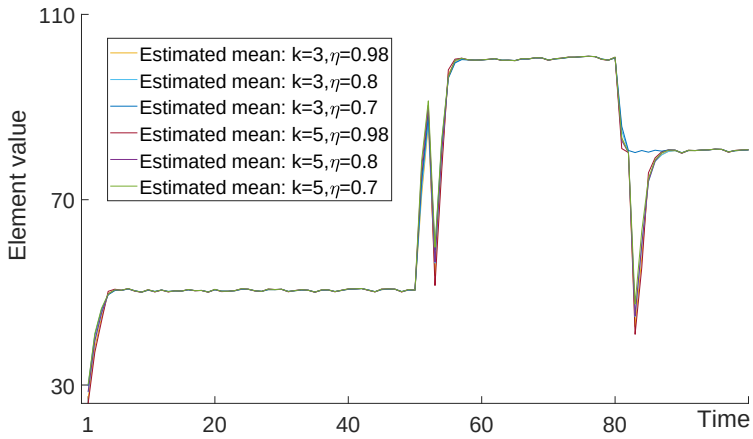


Figure 5.4: Estimated mean while varying η

performed in constant time. The second part is the process of calculating values of CCP, CCP mean, and updating estimators. The complexity of this process is constant $\mathcal{O}(1)$. The last part is a process which we calculate tail area and pivotal Gaussian value. We use a fixed number (100) of steps to search for the boundary of the tail. Hence, this process also has constant complexity $\mathcal{O}(1)$. In summary, the complexity of our method is $\mathcal{O}(k) + \mathcal{O}(1) + \mathcal{O}(1) = \mathcal{O}(k)$, thus the proposed algorithm CCPD has a constant complexity with constant k, $\mathcal{O}(k)$.

5.5 Evaluation

We have performed thorough experiments to evaluate the performance of our method and compare it to the state-of-the-art algorithms. To make our experiments as real and generic as possible, we performed our evaluation on several different real-world datasets, with various characteristics. Besides that, an extensive experiment was conducted to evaluate the flow rates of the stream including detection delay, true positive, true negative, false negative, and accuracy on several artificial datasets with known ground-truth. We carried out the experiments on a computer running the Windows 10 operating system, having a 64 bit Intel i7 2.6 GHz processor, and 16 GB of RAM. The proposed algorithm was implemented in Java. All evaluations are performed using the MOA framework [20][1], with our algorithm integrated.

[1]Version 4.0.0, June 2017.

5.5.1 Datasets

For our experiments, we used both real-world datasets and synthetic datasets to evaluate our proposed method and compare with the state-of-the-art algorithms.

Real Datasets

Eight real-world datasets are used, that are widely-used as benchmark datasets for change-detection methods [18, 129, 61, 178], consisting of Electricity, Poker (Hand), Forest Covertype, Spam, Usenet1, Usenet2, Nursery, and EEG Eye State.

Electricity is an electricity consumption dataset collected from the Australian New South Wales Electricity Market. The dataset has 45,312 instances which contains electricity prices from 7 May 1996 to 5 December 1998. The instances were recorded by an interval every 30 minutes.

Poker-Hand is data of a hand consisting of five playing cards drawn from a standard deck of 52. The dataset contains 829,201 instances.

Forest Covertype is forest cover type for a given observation of 30 x 30 meter cells. The dataset was obtained from US Forest Service (USFS) Region 2 Resource Information System (RIS) data. The dataset is recorded in the Roosevelt National Forest of northern Colorado, US. *Forest Covertype* contains 581,012 instances.

Spam is a dataset based on the Spam Assassin collection and contains both spam and legitimate messages. The *Spam* dataset has 9,324 instances.

Usenet1 and *Usenet2* are based on the twenty newsgroups data set, which consists of 20,000 messages taken from twenty newsgroups. Each of *Usenet1* and *Usenet2* contains 1,500 instances obtained from twenty newsgroups data set to present a stream of messages of a user.

Nursery was derived from a hierarchical decision model originally developed to rank applications for nursery schools in Ljubljana, Slovenia. The dataset contains 12,960 instances.

EEG Eye State was originally used to predict eye states by measuring brain waves with an electroencephalographic (EEG), i.e., finding the correlation between eye states and brain activities [133]. It consists of 14,980 instances with 14 EEG values, where each value indicates the eye state.

Electricity, *Forest Covertype* and *Poker-Hand* were obtained from the most popular open source framework for data stream mining MOA[2], while the remaining datasets were obtained from the UCI Machine Learning Repository[3] (University of California, Irvine)

Synthetic Datasets

There is no ground-truth in real-world datasets. Therefore, in our evaluation in performance evaluation of our drift detection method, four widely-used synthetic data streams, namely Mixed, Sine1, Circles and LED [129, 61], are used with ground-truth of drift point. Each dataset contains 100,000 instances, and 10% noise in class of instances. In brief, the characteristics of these datasets are as follows:

- *Mixed*: This dataset contains four attributes, including two Boolean attributes and two numeric attributes in the [0, 1] interval. The Mixed dataset contains abrupt concept drifts. The drifts occur at every 20,000 instances with a transition length $\xi = 50$.

- *Sine1*: In this dataset there are two attributes that are uniformly distributed in the [0, 1] interval. The Sine1 dataset contains abrupt concept drifts. The drifts occur at every 20,000 instances with a transition length $\xi = 50$.

- *Circles*: This dataset uses four circles to simulate concept drifts. Each instance has two numeric attributes on the [0, 1] interval. The Circles dataset contains gradual concept drifts. The drifts occur at every 25,000 instances with a transition length $\xi = 500$.

- *LED*: This dataset is a seven-segment display of digit dataset. The LED dataset contains gradual concept drifts. The drifts occur at every 25,000 instances with a transition length $\xi = 500$.

[2]https://moa.cms.waikato.ac.nz/datasets/
[3]http://archive.ics.uci.edu/ml/datasets.html

5.5.2 Performance

This section presents the experimental results of the proposed method compared to the state-of-the-art algorithms. We performed empirical experiments to evaluate our proposed algorithm and compared it against HDDM_W, HDDM_A [61], DDM [64], EDDM [13], SeqDrift2 [127], FHDDM [129][4], and RDDM[5] [16]. These algorithms were chosen because they are the state-of-the art algorithms in drift detection. Implementation of the algorithms are provided by the MOA framework. In all our experiments we used Natïve Bayes (NB) and Hoeffding Tree (HT) classifiers as the base learners, which are frequently used in the literature [18, 129, 61, 127]. In addition, we performed comparison with a baseline method, called "No Change Detection", that detects changes with base learners only. The confidence value is set to 0.001, η is set to 0.99. The λ in HDDM_W is set to 0.05 as recommendation by the authors. The sliding window size is set to 25 in FHDDM. According to the authors, it is the optimal value providing the best classification accuracy. With the other algorithms, in all the tests we use the default parameters and configuration values as recommended by the authors of the original work. Accuracy evaluator is computed base on a window with size 100. Frequency of sampling process is every 100 samples. Furthermore, we prepare two versions, named $CCPD_{k=5}$ and $CCPD_{k=3}$. $CCPD_{k=5}$ is our proposed method running with a projection on the last five data points in the stream, while $CCPD_{k=3}$ runs with a projection on the last three data points in the stream.

Table 5.1-5.2 show the average ($\overline{a}$) and standard deviation (std) accuracy of the change detection algorithms with NB and HT classifiers as base learner, respectively. In case of same accuracy result, we consider the algorithm with the lowest standard deviation to be the best. The results show that CCPD wins 5 times on total 8 datasets with NB classifiers and 4 times on total 8 datasets with HT classifiers. Further, Table 5.3 shows that our proposed method obtains the best accuracy scores with the majority of the datasets (five out of eight datasets), including Electricity, Spam, Usenet1, Corvertype, and Poker datasets; while on EEG Eye, Nursery, and Usenet2, the accuracy ranks are 2, 4, and 6.5 (the same rank with FHDDM), respectively. Another observation from the experiments is that, while varying the order of temporal dependency k, the accuracy scores on Nursery and Usernet2 change abruptly. This can be explained as follows. On the Usernet2 dataset, the reason is that the concept drift is moderate and the topic shift

[4]Source code of the FHDDM is provided by the authors of the algorithm

[5]Source code of the RDDM is obtained from the authors personal website

Table 5.1: Accuracy results (%) with Natïve Bayes (NB) classifier

Algorithm	Factor	Electricity (%)	Spam (%)	Usenet1 (%)	Usenet2 (%)	Covertype (%)	Nursery (%)	Poker (%)	EEG Eye (%)
$CCPD_{k=5}$	$\overline{a}$	85.19	92.65	75.67	61.20	88.98	83.46	79.96	99.45
	std	5.93	8.60	10.30	19.87	9.39	14.69	9.33	1.62
$CCPD_{k=3}$	$\overline{a}$	**86.20**	**92.88**	**78.27**	63.00	**89.18**	91.85	**80.42**	99.43
	std	**5.45**	**7.98**	**11.67**	21.81	**8.46**	9.20	**8.36**	1.68
$HDDM_{W-test(0.05)}$	$\overline{a}$	84.47	91.51	75.07	70.93	86.23	91.71	77.11	97.60
	std	6.57	7.39	11.91	13.79	8.07	7.61	8.80	5.13
$HDDM_{A-test}$	$\overline{a}$	85.09	90.67	75.20	71.00	87.44	92.51	76.48	98.33
	std	6.33	9.31	11.59	13.29	7.97	6.50	9.82	3.73
DDM	$\overline{a}$	82.70	89.50	73.73	72.93	88.03	91.72	61.97	**99.57**
	std	8.70	13.89	12.69	12.09	8.35	7.11	21.36	**1.19**
EDDM	$\overline{a}$	84.93	90.66	75.13	**73.27**	86.09	92.02	77.48	96.85
	std	6.23	8.60	12.32	**12.76**	8.66	6.77	8.40	7.26
SeqDrift2	$\overline{a}$	79.83	89.87	65.80	72.13	82.45	87.18	72.25	93.04
	std	11.50	13.38	23.51	11.54	12.31	8.84	14.29	17.05
RDDM	$\overline{a}$	84.88	91.45	74.73	73.07	86.87	**92.70**	76.67	99.06
	std	6.36	7.44	11.87	12.10	8.32	**6.10**	9.11	2.88
FHDDM	$\overline{a}$	83.32	91.36	74.73	71.20	85.10	89.09	76.68	97.28
	std	7.29	6.99	11.86	12.43	9.07	10.82	9.12	5.96
No Change	$\overline{a}$	74.17	90.63	63.33	72.13	60.53	83.35	59.55	47.35
Detection	std	14.69	10.93	23.64	11.54	21.76	14.94	21.96	46.67

Table 5.2: Accuracy results (%) with Hoeffding Tree (HT) classifier

Algorithm	Factor	Electricity (%)	Spam (%)	Usenet1 (%)	Usenet2 (%)	Covertype (%)	Nursery (%)	Poker (%)	EEG Eye (%)
$CCPD_{k=5}$	$\overline{a}$	83.49	92.67	74.47	57.20	89.44	72.22	79.15	99.62
	std	6.56	8.45	11.22	23.14	9.49	13.26	9.75	1.04
$CCPD_{k=3}$	$\overline{a}$	84.26	**92.98**	**75.53**	71.20	**89.51**	86.35	**79.53**	99.62
	std	6.21	**7.78**	**6.80**	12.31	**8.67**	14.82	**8.98**	1.04
$HDDM_{W-test(0.05)}$	$\overline{a}$	85.46	91.48	74.73	70.00	85.98	90.48	77.12	97.49
	std	6.91	7.85	8.41	9.00	8.23	6.66	8.93	4.94
$HDDM_{A-test}$	$\overline{a}$	**86.11**	91.66	74.60	68.87	87.27	90.93	76.40	98.89
	std	**6.45**	7.22	8.14	8.24	8.02	6.02	9.93	2.62
DDM	$\overline{a}$	85.83	91.94	73.00	69.80	82.58	91.31	72.74	**99.64**
	std	7.17	7.22	10.03	9.34	12.95	6.41	15.14	**1.04**
EDDM	$\overline{a}$	85.26	91.68	73.93	**72.00**	86.02	90.17	77.31	96.64
	std	6.66	7.88	8.96	**10.44**	8.46	6.79	8.68	7.33
SeqDrift2	$\overline{a}$	82.30	91.55	69.67	72.00	82.86	89.01	72.51	93.63
	std	11.12	9.23	20.38	11.01	12.09	6.83	13.89	16.42
RDDM	$\overline{a}$	85.83	92.09	74.27	69.60	86.43	**91.75**	76.70	99.06
	std	6.64	7.45	8.39	8.57	8.53	**5.68**	9.21	4.34
FHDDM	$\overline{a}$	84.93	92.38	74.40	69.93	85.07	89.85	76.72	97.02
	std	7.74	7.11	8.31	9.12	9.14	7.82	9.15	6.15
No Change	$\overline{a}$	78.87	90.17	63.13	72.00	80.58	89.85	76.07	75.35
Detection	std	12.40	12.84	21.78	11.01	13.28	7.82	16.19	28.62

on the dataset is blunt and blurred. Moreover, the number of samples in the dataset is small for the training. On Nursery, on the other hand, the change behavior may be due to the large number of distinct values of attributes in the dataset. Thus, the temporal dependency is loose, which has in turn an impact on detecting the changes. Nevertheless, the difference from the best accuracy is not significant. Overall, the empirical results show that CCPD has the best performance in all cases of combinations with NB and HT classifier.

To further evaluate the performance of the proposed method, and in order to get a fair comparison among the algorithms, we performed statistical significance tests based on the average rank of accuracy of the algorithms [41]. Firstly, we select the best accuracy of the algorithms with NB classifier and HT classifier and then report rank of accuracy of the algorithms. Secondly, we use Friedman test because it is a nonparametric analogue of the parametric two-way analysis of variance by ranks. Table 5.3 shows statistical test results using the methodology proposed in [41]. The number in parentheses at each cell is the rank of accuracy. The bold values indicate the best results per dataset. Based on the methodology, we reject the null-hypothesis because $F_F > \textit{Critical F-value}$. This means that there are significant differences between the algorithms. Finally, we use the Nemenyi post-hoc test to present these differences. We set the significance level at 5%. The critical value for 9 algorithms is 3.10. The critical difference at level 5% is $CD = 4.24$. Figure 5.5 shows the results of the Nemenyi test of the data from Table 5.3. On the figure, the methods on the right side have lower average rank of accuracy and are better than the methods on the left.

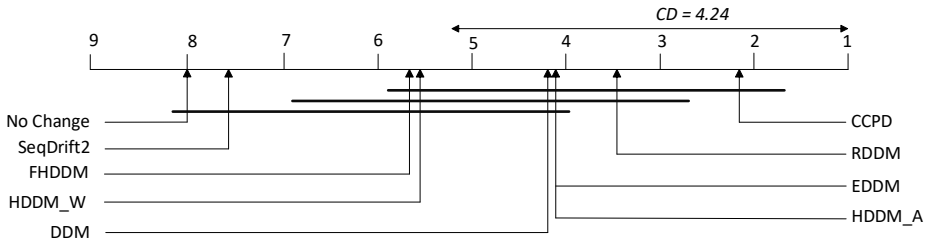


Figure 5.5: Nemenyi test with confidence level $\alpha = 0.05$

Table 5.3: Rank of accuracy of the algorithms and significance tests

Datasets	*FHDDM*	*HDDM$_W$*	*HDDM$_A$*	*DDM*	*EDDM*	*SeqDrift2*	*RDDM*	*NoChange*	*CCPD$_{k=3}$*
Electricity	84.93(7)	85.46(5)	86.11(2)	85.83(3.5)	85.26(6)	82.23(8)	85.83(3.5)	78.87(9)	**86.20(1)**
Spam	92.38(2)	91.51(8)	91.66(6)	91.94(4)	91.68(5)	91.55(7)	92.09(3)	90.63(9)	**92.98(1)**
Usenet1	74.73 (5.5)	75.07(4)	75.20(2)	73.73(7)	75.13(3)	69.67(8)	74.73(5.5)	63.33(9)	**78.27(1)**
Usenet2	71.20(6.5)	70.93(9)	71.00(8)	72.93(3)	**73.27(1)**	72.13(4.5)	73.07(2)	72.13(4.5)	71.20(6.5)
Covertype	85.10(7)	86.23(5)	87.44(3)	88.03(2)	86.09(6)	82.86(8)	86.87(4)	80.58(9)	**89.51(1)**
Nursery	89.85(7.5)	91.71(6)	92.51(2)	91.72(5)	92.02(3)	89.01(9)	**92.70(1)**	89.85(7.5)	91.85(4)
Poker	76.72(4)	77.11(3)	76.48(6)	72.74(8)	77.48(2)	72.51(9)	76.70(5)	76.07(7)	**80.42(1)**
EEG Eye	97.28(6)	97.60(5)	98.89(4)	99.64(1)	96.85(7)	93.63(8)	99.06 (3)	75.35(9)	99.62(2)
average rank	5.69	5.63	4.13	4.19	4.13	7.69	3.38	8.0	**2.19**
mean rank			*Value of χ^2*			F_F			*Critical F-value*
5.0			31.8167			6.9202			2.1782

Table 5.4: Accuracy results (%) of CCPD with Natïve Bayes classifier

CCPD$_k$	Electricity (%)	Spam (%)	Usenet1 (%)	Usenet2 (%)	Covertype (%)	Nursery (%)	Poker (%)	EEG Eye (%)
k=2	**86.29±5.35**	92.84±7.79	76.07±11.40	**70.40±10.63**	**89.29±8.02**	85.35±14.23	80.29±8.12	99.43±1.68
k=3	86.20±5.45	**92.88±7.98**	**78.27±11.67**	63.00±21.81	89.18±8.46	**91.85±9.20**	**80.42±8.36**	99.43±1.68
k=4	85.78±5.62	92.80±8.18	77.13±10.53	65.07±20.62	89.06±8.94	83.45±14.72	80.29±8.80	99.45±1.68
k=5	85.19±5.93	92.65±8.60	75.67±10.30	61.20±19.87	88.98±9.39	83.46±14.69	79.96±9.33	99.45±1.62
k=6	84.83±6.08	92.74±8.40	71.47±9.89	61.27±19.88	88.95±9.55	52.02±12.51	79.61±9.83	99.46±1.57
k=7	84.39±6.39	92.66±8.57	61.87±10.91	53.00±14.85	88.95±9.65	43.02±5.15	79.29±10.23	**99.48±1.45**

5.5.3 Impact of the k-Order

In this subsection, we run tests to evaluate the dependencies of data points in the streams in the Electricity, Poker, Forest Covertype, Spam, Usenet1, Usenet2, Nursery, and EEG Eye State datasets. We record average accuracy and standard deviation of the CCPD on the datasets with NB classifier while varying the parameter k from 2 to 7.

Table 5.4 shows the average accuracy and standard deviation of the CCPD method while varying k from 2 to 7. From the results, we can observe that the different values of k will affect to the accuracy of the algorithm. The best accuracy is almost obtained at order k=2 or k=3. On Electricity, Usenet2, and Covertype datasets, the best performance is obtained when we project on k=2 data, while on the remaining datasets, the best performance is at k=3. On the EEG Eye State dataset, when the value of k increases from 2 to 7, the accuracy also slightly increases. For the sake of comparison, the table only shows $k \in [2, 7]$. Since we observed that the values were still increasing, we decided to vary k, until $k = 20$ to find the optimal value. The optimal accuracy of 99.55% was found at $k = 13$. The best accuracy at different order k also depends on characteristic of the input dataset. Furthermore, the results show that change in accuracy, abrupt or slight, is dataset-dependent. On Electricity, Spam, Covertype, and Poker datasets, when we vary value of order k, the accuracy changes slightly. In contrast, the accuracy varies abruptly on Usenet1, Usenet2, and Nursery datasets.

As shown in the results in Table 5.4, the best results are usually in 2 or 3 orders of dependency. The reason can be explained as that the proposed method uses the fixed length for projection of temporal dependencies. In real life datasets, the order of dependencies may not be fixed length, which means that each data point in a stream may have temporal dependencies from different number of its previous data points. For this reason, our further research will include studying higher-order temporal dependencies.

5.5.4 True Change Point and Delay

One of the disadvantages of some real-world datasets in change detection problem is that the ground truth of the datasets is unknown. On the other hand, the performance of a detection method is evaluated base on the accuracy and delay of detected change points to the real change points in the data. In the above section, we have presented the efficiency of our proposed method with high accuracy detection. In this section, for a further evaluation of the algorithm, we describe the tests we performed with CCPD on a real-world dataset for which its ground truth has been known. This is

the Electricity dataset, which is also widely-used in many methods [21, 178] for change detection. In this dataset, data are heavily autocorrelated with frequency peaks in every 48 instances [178].

In this experiment, we performed tests on the first 1,000 instances of the dataset. We then record change points detected by CCPD and other state-of-the-art algorithms with respect to true change points that are known as ground truth. Specifically, we did experiments employing CCPD, HDDM$_W$, HDDM$_A$, CUSUM, and PAGE-HINKLEY algorithms. CCPD, HDDM$_W$, and HDDM$_A$ are here online detection algorithms. PAGE-HINKLEY is a concept drift detection based on the Page Hinkley Test, while CUSUM is a drift detection method based on cumulative sum. For the best competitive comparison, CUSUM and PAGE-HINKLEY are executed in an online manner at every data point in the stream. Here, we set sample frequency to 1. We adjust the minimal number of instance to 1, and set all the other parameters to default values as in MOA framework according to prior works.

Table 5.6 shows the change points detected by the algorithms on Electricity dataset. The CCPD detects online and at exact change points. CUSUM and PAGE-HINKLEY have the same result with a short delay detection of change, which is 1 data point. HDDM$_{A-test}$ produces the largest delay with 15 data points in delay. And the last algorithm HDDM$_{W-test}$ detects change with 4 data points in delay.

5.5.5 Experiments on Synthetic Datasets

This subsection presents experiments to evaluate detection delay, true positive (TP), true negative (TN), false negative (FN), and accuracy of our proposed method. We compared the results with several state-of-the-art drift detectors, including EDDM [13], ECDD [134], SeqDrift2 [127], and RDDM [16], on four widely-used synthetic data streams in the literature [61, 129]: Mixed, Sine, Circles, and LED. All experiments on the synthetic datasets were performed using the MOA framework with parameters set to the optimal values for all the compared algorithms as recommended in the original papers. We adopted the *acceptable delay* length metric [129] to evaluate the performance of the algorithms. Given a threshold, if a detector can detect a change within a threshold delay from the true change point, it is considered as a true positive. In the experiments on the Mixed and Sine1 datasets, the level of temporal dependency k is set to 3 and the acceptable delay is set to 250. While on the Circles and LED datasets, k is empirically set to 5 and the acceptable delay is set to 1,000.

Table 5.5: Results with Natïve Bayes(NB) and Hoeffding Tree(HT) classifiers on synthetic datasets

		Algorithms	Delay	TP	FP	FN	Accuracy	Rank
Mixed dataset	NB	CCPD	62.60 ± 7.74	4.0 ± 0.0	0.01 ± 0.1	0.0 ± 0.0	**83.34 ± 0.08**	1
		RDDM	104.97 ± 12.12	3.99 ± 0.1	1.86 ± 1.66	0.01 ± 0.1	83.24 ± 0.09	2
		SeqDrift2	200.0 ± 0.0	4.0 ± 0.0	4.39 ± 0.79	0.0 ± 0.0	82.91 ± 0.08	3
		ECDD	38.87 ± 24.65	3.81 ± 0.42	142.29 ± 7.90	0.19 ± 0.42	81.00 ± 0.15	4
		EDDM	247.47 ± 8.65	0.11 ± 0.31	20.22 ± 7.70	3.89 ± 0.31	80.30 ± 2.33	5
	HT	CCPD	63.22 ± 9.23	4.0 ± 0.0	0.04 ± 0.20	0.0 ± 0.0	**83.37 ± 0.10**	1
		RDDM	106.68 ± 11.32	3.99 ± 0.1	3.49 ± 2.48	0.01 ± 0.1	83.17 ± 0.12	2
		SeqDrift2	200.0 ± 0.0	4.0 ± 0.0	4.98 ± 1.21	0.0 ± 0.0	82.91 ± 0.11	3
		ECDD	39.76 ± 26.08	3.79 ± 0.46	138.34 ± 7.95	0.21 ± 0.46	80.95 ± 0.15	4
		EDDM	248.46 ± 7.73	0.05 ± 0.22	21.51 ± 7.74	3.95 ± 0.22	80.65 ± 0.82	5
Sinel Dataset	NB	CCPD	59.54 ± 6.89	4.0 ± 0.0	0.01 ± 0.1	0.0 ± 0.0	**86.03 ± 0.25**	1
		RDDM	89.73 ± 16.54	3.99 ± 0.10	3.93 ± 2.92	0.01 ± 0.1	85.98 ± 0.27	2
		SeqDrift2	200.0 ± 0.0	4.0 ± 0.0	4.26 ± 0.0	0.0 ± 0.58	85.60 ± 0.25	3
		ECDD	33.30 ± 23.22	3.85 ± 0.39	153 ± 8.34	0.15 ± 0.39	84.38 ± 0.14	4
		EDDM	234.28 ± 22.33	0.57 ± 0.64	33.53 ± 11.56	3.43 ± 0.64	83.44 ± 2.88	5
	HT	CCPD	58.45 ± 6.55	4.0 ± 0.0	0.03 ± 0.17	0.0 ± 0.0	**87.02 ± 0.15**	1
		RDDM	93.54 ± 7.82	4.0 ± 0.0	4.72 ± 3.59	0.0 ± 0.0	86.79 ± 0.19	2
		SeqDrift2	200.0 ± 0.0	4.0 ± 0.0	4.83 ± 1.16	0.0 ± 0.0	86.53 ± 0.15	3
		ECDD	36.58 ± 25.54	3.8 ± 0.43	153.78 ± 7.67	0.2 ± 0.43	84.28 ± 0.14	5
		EDDM	243.83 ± 22.33	0.22 ± 0.64	33.77 ± 11.56	3.78 ± 0.64	84.71 ± 2.88	4

(*table continues*)

Table 5.5: Results with Natïve Bayes(NB) and Hoeffding Tree(HT) classifiers on synthetic datasets. (continued)

		Algorithms	Delay	TP	FP	FN	Accuracy	Rank
Circles Dataset	NB	CCPD	621.09 ± 139.12	1.59 ± 0.57	0.50 ± 0.64	1.41 ± 0.57	83.49 ± 0.52	3
		RDDM	406.50 ± 69.75	2.99 ± 0.1	2.15 ± 1.95	0.01 ± 0.1	84.05 ± 0.12	2
		SeqDrift2	276.67 ± 91.56	2.92 ± 0.27	2.49 ± 0.98	0.08 ± 0.27	**84.13 ± 0.14**	1
		ECDD	194.64 ± 158.13	2.84 ± 0.37	174.53 ± 7.62	0.16 ± 0.37	83.18 ± 0.11	4
		EDDM	938.27 ± 107.14	0.35 ± 0.5	31.09 ± 18.23	2.65 ± 0.5	83.12 ± 0.4	5
	HT	CCPD	524.62 ± 144.31	2.02 ± 0.57	0.67 ± 0.78	0.98 ± 0.57	85.94 ± 0.42	3
		RDDM	293.80 ± 38.72	2.98 ± 0.14	0.79 ± 1.26	0.02 ± 0.14	86.46 ± 0.16	2
		SeqDrift2	202.67 ± 16.19	3.0 ± 0.0	3.08 ± 0.91	0.0 ± 0.0	**86.47 ± 0.14**	1
		ECDD	186.40 ± 151.67	2.86 ± 0.35	175.16 ± 8.39	0.14 ± 0.35	83.21 ± 0.12	5
		EDDM	987.75 ± 54.64	0.06 ± 0.24	24.45 ± 14.57	2.94 ± 0.24	84.89 ± 0.29	4
LED Dataset	NB	CCPD	481.75 ± 129.42	2.77 ± 0.55	13.44 ± 7.75	0.23 ± 0.55	89.15 ± 0.57	2
		RDDM	321.80 ± 51.19	2.98 ± 0.14	0.61 ± 0.96	0.02 ± 0.14	**89.63 ± 0.04**	1
		SeqDrift2	445.33 ± 193.24	2.75 ± 0.46	278.82 ± 47.74	0.25 ± 0.46	76.54 ± 2.27	5
		ECDD	194.53 ± 139.51	2.86 ± 0.40	60.51 ± 3.58	0.14 ± 0.40	86.41 ± 0.19	4
		EDDM	949.61 ± 69.29	0.70 ± 0.73	6.33 ± 1.97	2.3 ± 0.73	88.32 ± 0.53	3
	HT	CCPD	479.84 ± 124.70	2.77 ± 0.55	13.9 ± 7.07	0.23 ± 0.55	89.21 ± 0.31	2
		RDDM	321.88 ± 51.20	2.98 ± 0.14	0.61 ± 0.96	0.02 ± 0.14	**89.63** ± 0.04	1
		SeqDrift2	426.0 ± 174.18	2.78 ± 0.44	277.06 ± 47.71	0.22 ± 0.44	76.51 ± 2.29	5
		ECDD	197.07 ± 140.93	2.85 ± 0.41	60.19 ± 3.68	0.15 ± 0.41	86.39 ± 0.19	4
		EDDM	954.97 ± 63.30	0.66 ± 0.71	5.97 ± 1.70	2.34 ± 0.71	88.33 ± 0.50	3
		Algorithms	CCPD	RDDM	SeqDrift2	ECDD	EDDM	
		Average Rank	**1.75**	**1.75**	3.0	4.25	4.25	

Table 5.6: Change points detected.

Algorithm	Detected Points (where i =1, 2, ...)
$CCPD_5$	$(48 \times i + 1) \pm 0$
$HDDM_{W-test(0.05)}$	$(48 \times i + 1) \pm 4$
$HDDM_{A-test}$	$(48 \times i + 1) \pm 15$
$CUSUM$	$(48 \times i + 1) \pm 1$
$PAGE - HINKLEY$	$(48 \times i + 1) \pm 1$

Table 5.5 shows the average and standard deviation of classification results for the CCPD, EDDM, ECDD, SeqDrift2, and RDDM running on 100 samples of datasets. The results show that, in most cases, ECDD and EDDM are the worst detectors. On the Circles dataset, SeqDrift2 has the best performance, while SeqDrift2 has the worst performance on the LED dataset. This can be explained as follows. SeqDrift2 maintains a fixed size reservoir sampling for concept drift detection. The reservoir sampling contains 200 instances, and this size is suitable for the Circles dataset since it contains gradual concept drifts with a transition length of 500. The low accuracy of SeqDrift2 on LED is a result of its very high false positive. On the Mixed and Sine1 datasets, we observe that, the shortest delay is obtained by ECDD. The reason is that the number of instances in the estimated window in ECDD is small. However, with ECDD, both the TP rate and the FP rate are high, thus resulting in a low accuracy. In almost all cases, CCPD and RDDM have the best accuracy and very good flow rates of detection delay, TP, FP, and FN. The CCPD has the most accurate and very low FP, FN rates on the Mixed and Sine1 datasets; while RDDM has good performance on the LED dataset. This is because RDDM discards old instances from the stream, while in CCPD, we weigh the current instance based on a projection on k latest instances in the stream. Therefore, any concept drifts can be quickly detected on abrupt concept drift datasets like Mixed and Sine1. Overall, the CCPD and RDDM have the same rank (1.75).

5.5.6 Runtime Performance

In terms of runtime performance, we performed experiments to evaluate the classification time and the streaming processing speed of our proposed method. Table 5.7 presents the evaluations of the method in terms of running time in streaming on Electricity, Poker, Forest Covertype, Spam, Usenet1, Usenet2, Nursery, and EEG Eye datasets. It shows number of learning evaluations, average total running time in second, average time using for each learning process and number of learning can be processed

per second. From this table, we can observe that our detector is capable of processing a streaming at high velocity, up to 204.5 process per second. Hence, it is feasible to detect changes in an online setting as in streaming manner.

Table 5.7: Evaluations on running time of the CCPD

Metrics	No. of learning evaluations	Average total running time	Average time/learning	Number of processing
Electricity	454	3.135(s)	6.90(ms)	144.8/(s)
Spam	94	5.401(s)	57.46(ms)	17.4/(s)
Usenet1	15	0.190(s)	12.67(ms)	78.9/(s)
Usenet2	15	0.323(s)	21.53(ms)	46.5/(s)
Covertype	5,812	59.513(s)	10.24(ms)	97.7/(s)
Nursery	130	23.719(s)	182.45(ms)	5.5/(s)
Poker	8,293	40.557(s)	4.89(ms)	204.5/(s)
EEG Eye	150	11.220(s)	74.8(ms)	13.4/(s)

5.6 Conclusion

In this chapter, we presented a new approach for detecting changes in an open-ended data stream. We proposed a k-order Candidate Change Point (CCP) Model that builds on linear higher order Markov processes, in order to exploit the temporal dependency among data in a stream. The main idea with the model is to compute the probability of finding change points in a given observation time window, using the temporal dependency information or factors between different observed data points in a stream. To cope with the dynamic nature of the stream, we proposed an approach that can continuously optimize the temporal dependency factors by using a Euclidean projection on ℓ_1 ball constraints. In addition, we introduced a concept called CCP trail, which refers to the probabilistic path from a specific observed data point to another previously observed data point. Our approach adapts the probability of finding change points to continuously estimate the CCP trail means in streaming data. Using CCP trail mean values, we applied statistical tests to detect the change points. To evaluate our approach, we performed extensive experiments using several datasets and compared our algorithm to the state-of-the-art algorithms. Our evaluation showed that our k-order Candidate Change Point Model is effective, and that the Candidate Change Point Detector (CCPD) algorithm outperforms the state-

of-the-art algorithms on most of the datasets. In addition, our method has a linear time performance, which enables it to be deployed online in real-world stream applications.

Part III

Feature Correlations with Concept Drift

In this part, we focus on investigating the correlations and transitions of data in an evolving and changing context using multiple factors to model user behaviors. We propose a solution to the problem of sketching a stream of data in which the data are arriving at high volume and with high velocity. A method is introduced that sketches the stream using a compact representation, quickly adapts to concept drift, and preserves the similarity of data with high accuracy. An evolving, auto-tuning coefficient model is presented that uses multiple weighted factors to simulate the correlations between the features of data in order to estimate a data stream with concept drift.

Chapter 6

Sketching Using Multiple Weighted Factors

To address the issues related to concept drift, the concept of a forgetting factor has been normally used to determine the significance of data based on the time at which a data item occurs in a stream. However, most current studies assume that this forgetting factor is constant, both at each point of observation and over the whole process. This is too restrictive, since in many real-world applications, we cannot apply the same factor to each previous data item at a particular single observation point. The contributions presented in this chapter represent solutions to research questions RQ1 and RQ3 to RQ5 [47], i.e., *How can we maintain high volume and high velocity streaming data considering the limitations on memory and computation infrastructure? Are there any correlations/transitions between data points or the features of data in the data space? How can we design an efficient algorithm to adapt quickly to concept drift in an evolving set of data?*

6.1 Motivation

Normally, data in a stream can be accessed only once, thus making efficient processing of streams a challenging but crucial task. A possible solution is sketching of the streaming data. Sketching is an effective approximation method that maps a stream into a maintainable form, while still retaining the characteristics of the stream with high accuracy. Development of efficient sketching techniques has attracted much research during the past decades [165, 167, 164]. In addition to being used to visualize the statistical information of the data, *histogram* estimation is among commonly used techniques to sketch the underlying distribution of data [142]. Still,

an important limitation of existing histogram-based methods is that most of these methods were developed with the assumption that histograms are drawn from a non-changing or static data distribution [72, 153]. In practice, streaming data are inherently dynamic and evolve over time, which in turn result in dynamic changes in the underlying data distribution, i.e., concept drift [63].

To cope with the issues related to concept drift, the concept of forgetting factor has been introduced to determine the significance of data according to the time the data occurs in a stream [142, 168, 50, 140]. However, most of the current work, e.g., [142] and [168], assume that a forgetting factor is constant at every point of observation and in the whole process, which is too restrictive. This is because in many real-world applications, we cannot consider the same factor for every past data at a particular single observation point. This motivates the need for a model that is capable of dynamically adapting to changes in a data stream. We refer such a model to as an *evolving model*, i.e., a model that can automatically tune its coefficients to follow any occurring changes.

Contributions

The major contributions of this chapter are as follows.

- We propose an evolving model with adaptive, auto-tuning coefficients to investigate the correlations of data, and simulate the transition between features, group of features in dynamic changing data.
- We develop a novel algorithm that can sketch the histograms from a data stream using multiple weighted factors, while taking into account the time-sensitive changes in the stream using different adaptive weighted factors for different groups of data, at every observation point.
- We fully evaluate our approach with extensive experiments on both synthetic and real-life datasets. Our proposed method quickly adapts with concept drifts. Our algorithm is able to achieve up to 3.99% higher overall classification accuracy than the baseline algorithms, and the error rate is 12 times better than the state-of-the-art methods with concept drift.

Organization

The remainder of this chapter is organized as follows. Section 6.2 reviews the related work. Section 6.3 briefly introduces the background and formulation

of sketching a stream of data. Section 6.4 presents the proposed model and the adaptive estimation method. Section 6.5 describes and discusses the results from the experimental evaluations. Finally, Section 6.6 concludes the chapter.

6.2 Related Work

Sketching is a hashing technique [30] that has been extensively studied and has been widely applied to maintain a dataset in an approximating compact representation. Numerous algorithms for sketches have been proposed, including Count-Min sketch (CM) [33], Count-Min sketch with conservative update (CM-CU) [71], and Augmented sketch [135]. However, due to skewness and dynamic property of data in a stream, these sketches are inefficient when applied on real data streams [34, 172], since they often lead to a need to estimate the frequency while considering the weight of data in a stream.

In order to preserve the characteristics of data, a sketch needs to simulate the data distributions of a given dataset into a form that is closest possible to the real distributions. Several algorithms have been proposed to sketch a given dataset [25, 83, 139, 84]. One of the most well-known algorithms is locality-sensitive hashing (LSH) [83], which is a data-independent method that can guarantee the collision probability between similar data points. Examples of LSH-based methods are SimHash [139] and MinHash [25] which differ on the ways they compute similarities. SimHash [139] uses cosine distance to measure the similarity between data files, while in MinHash (or minwise hashing) [25], Jaccard similarity is used to estimate the similarity between two sets. MinHash-based algorithms are generally efficient and are more efficient than SimHash when using cosine similarity, but one of the drawbacks is that they consume much space. Further, minwise hashing uses large number of permutations on the data, which, in turn, degrades the performance of the algorithms since minwise hashing is normally adopted in the context of binary or high-dimensional data. To cope with the costly processing of data in minwise hashing, Weighted Minwise Hashing (WMH) [84], inspired the idea of using densification and one permutation [106]. WMH uses a "rotation" scheme for densifying the estimated sparse sketches of one permutation hashing that assigns new values to all the empty buckets. As a result, WMH greatly reduces computation cost compared to MinHash-based hashing. The WMH is much simpler, significantly faster, and more memory efficient than the original ones, especially in very sparse datasets [30].

Histograms has long been seen as one of the most important data statistical tools for providing information about a data distribution [131]. There ex-

ist various algorithms for estimation of histograms generated from streaming data. A few of these algorithms consider dynamic forgetting factor, including the most recent one, HistoSketch [168]. HistoSketch adopts Weighted Sampling method [84, 105] and Hamming distance, which is comparable with our approach. However, every time a new data arrives in the stream, to cope with concept drift, HistoSketch uses a constant decay factor to decrease the importance of outdated data. Several adaptive estimation approaches have been proposed to efficiently detect concept drifts [22, 142]. An important difference with our approach is that all of these algorithms compute a decay factor at every observation. This means that they decrease the importance of all outdated data by that factor and use the same factor on all data. As mentioned above, the mechanism of applying the same forgetting factor to all data at different timestamps in the stream is too restrictive and could be impractical. To be effective, it is necessary that a sketching approach considers a model that can automatically adapt its decay factor following the changes in the stream. Moreover, sketching the histogram of a stream is an approximate solution and is based on randomization processes and randomization hashing functions. Single randomization is usually a weak process, which in general yields poor performance [140]. Our hypothesis is that combining several weak single processes to form an ensemble component can obtain better performance. Particularly, each single component in the ensemble model estimates a sample of the histogram of the data in a stream, in a sampled sketch. Then, the final sketch is determined using a scoring function on these estimated sketches in a final component.

6.3 Preliminaries

In the following, we present fundamental definitions about consistent weighted sampling [113, 84, 105] that we use in our solution on sketching of a streaming data containing concept drifts. Given two data vectors (weighted sets) of k elements $X, Y \in \mathbb{R}^k$,

Definition 34 (Uniformity [113, 84, 105])**.** *Let (i, S_i) be a sample of X, where $1 \leq i \leq k$ and $0 \leq S_i \leq X_i$. A process is known as uniformity sampling if the sample is distributed uniformly over the pairs.*

Definition 35 (Consistency [113, 84, 105])**.** *Assume that X dominates Y, i.e., $Y_i \leq X_i$, for all i and $1 \leq i \leq k$. If a sample (i, S_i) is sampled from X and satisfies $S_i \leq Y_i$, then (i, S_i) is also sampled from Y.*

Definition 36 (Consistent weighted sampling [113, 84, 105])**.** *Consistent weighted sampling is a sampling process that satisfies the properties of both uniformity and consistency.*

The min-max similarity between X, Y is defined by [84, 105]:

$$SIM_{min-max}(X,Y) = \frac{\sum \min(X_i,Y_i)}{\sum \max(X_i,Y_i)}$$

Let $SE(X)$ and $SE(Y)$ denote the samples of data vectors X and Y, respectively, using a consistent weighted sampling schema. The collision probability between the two vectors X, Y is exactly its min-max similarity [84, 105]:

$$Pr[SE(X) = SE(Y)] = SIM_{min-max}(X,Y).$$

Assume $SE(X) = \{i_x^*, s_x^*\}$, and $SE(Y) = \{i_y^*, s_y^*\}$ are two CWS samples of X and Y, respectively. Theoretical results for consistent weighted sampling show that:

$$Pr(i_x^* = i_y^*) \simeq Pr(\{i_x^*, s_x^*\} = \{i_y^*, s_y^*\}) \tag{6.1}$$

Given a data stream S, let t the current time point, v_t the current incoming item at an instant in time (order) t with timestamp $\mathcal{T}_t$, and $V_t = (v_1, v_2, \ldots, v_t)$ the current sub-stream. The task of sketching a stream S at a point of time t involves maintaining a parameter sketch SK (s elements) such that SK is a compact representation of the current sub-stream V_t, and SK preserves the interest (similarity) measure of V_t. In particular, in this work, the data stream is a stream of histogram elements of many POIs (point of interest). We estimate an adaptive weighted count-min histogram for each POI in the stream. Then, a sample is generated by utilizing a consistent weighted sampling schema [113, 84, 105] on each weighted histogram. Specifically, we utilize 0-bit consistent weighted sampling [104] because of its simplicity, having the same performance (shown in Eq. 6.1), and i^* is bounded by the data. To get s samples, we repeat the process s times using an independent set of random numbers, and the similarity of two samples is preserved based on the theoretical guarantee of consistent weighted sampling.

6.4 Model and Solution

This section presents the proposed model and the adaptive estimation method to sketch the streaming data.

6.4.1 Histograms

For a data stream, a histogram is a data summarization method, in which data are hashed into a set of buckets. Each bucket maintains the frequency of the hashed data. In order to estimate the histogram of a data stream, we adopt the count-min sketching method originally proposed by [33] because of its efficiency and simplicity. A histogram of a data stream using a count-min sketch is created using a matrix $\Theta \in \mathbb{R}^{n \times m}$, where n is the number of random hash functions h_i, $i = 1, 2, \ldots, n$, and m is the number of ranges, such that each hash function h_k maps an incoming data item v in the stream to a range from 1 to m. When a new data item v arrives in the stream, n hash functions h_i, $i = 1, 2, \ldots, n$, are used to hash the value v to a corresponding range (a set of m columns of matrix Θ). The value of the corresponding cell of matrix Θ is then incremented, i.e., $\Theta(i, h_i(v)) = \Theta(i, h_i(v)) + 1$.

6.4.2 Weighted Sampling

This subsection presents how the proposed method weighs the importance of histogram element frequencies in a stream. The weighted cumulative frequencies of histogram elements are used to estimate a compact sketch of that stream.

Evolving Data and Weight of Histogram

Fading factor, which is used to weigh the elements within a streaming context, was proposed to reduce the impact of noise in streaming data and weigh the importance of data in the stream [168, 142, 140]. In these works, at each single observation time, the weight of all the old data is divided by the same constant factor. In a data stream, however, data evolves over time, and the evolvement of data might be different with respect to different group of elements. Some previously generated elements or groups of elements might have more importance than other elements. Therefore, it is crucial not to treat the importance of all past data equally by the dividing with the same constant factor. On top of this, a data point could be correlated with other data points, which makes this even more important.

Moreover, there might be strong correlation between features of data during the time they are evolving, such that a feature evolves, this might affect or impact the evolvement of other features, as well. Intuitively, the idea of the modeling procedure is as follows. At each observation time, groups of elements might have different weights and there might exist correlations between some features. We need to model the evolvement data

features by estimating the variables that determine the weights of the histogram elements. These variables are unknown in advance, and they are continuously updated and estimated from observed data. The model then automatically performs the variable selection to minimize a loss function, while considering the variance and sparsity factors of the variables.

Here, to address the correlation of data, we utilize the idea of the elastic net [179] to simulate the evolvement of data in our proposed method. In our model, we define an objective function at an observation time t, having a form of an elastic net, as follows:

$$\mathcal{L}(\Theta_i) = \Omega_1(\Theta_i^t) + \Omega_2(\Theta_i^t) + \Delta(\Theta_i^t, \Theta_i^{t-1}), \tag{6.2}$$

where Θ_i are our predicting estimators. The meaning of penalties in the objective function are the same as in elastic net regularization. The first term, a penalty term, $\Omega_1(\Theta_i^t) = \alpha\beta\Lambda(\Theta_i^t)$ is to control sparsity in the single solution. To estimate the sparsity, we choose $\Lambda(\Theta_i^t)$ as ℓ_1 column norm of vector at row i of the matrix Θ at time t, $\Lambda(\Theta_i^t) =\| \Theta_i^t \|_1$, because it minimizes the less squares loss function [155]. However, using only ℓ_1 norm tends to select one variable from a group of highly correlated variables and set less important coefficients to zero. To overcome this drawback, the second term $\Omega_2(\Theta_i^t)$ is used to enforce the hierarchy constraint, and it is set to a form of 2-norm to force the coefficients close to the average value rather than zero, $\Omega_2(\Theta_i^t) = (1-\alpha)\beta \| \Theta_i^t \|_2^2$. The parameter α, β in the first two penalties are tuning parameters, $\beta \in [0, \infty)$, and $\alpha \in [0, 1]$. Here, α controls the relative weight on the first two penalties and is a relaxation parameter; while parameter β controls the selection of parameters.

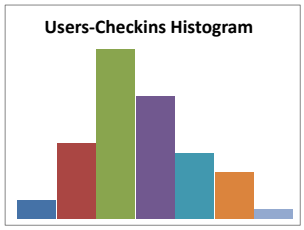
The last term $\Delta(\Theta_i^t, \Theta_i^{t-1})$ is a penalty to minimize the deviation across different timestamps. The last term in Eq. 6.2 controls this deviation, on which we target the similarity of Θ at different observation times t and (t-1). The deviation is estimated by a robust ℓ_2 penalty that is useful in concept drift detection. Δ has a form of $\Delta(\cdot) =\| \cdot \|_2$. The objective function in Eq. 6.2 now can be rewritten as:

$$\mathcal{L}(\Theta_i) = \alpha\beta \| \Theta_i^t \|_1 +(1-\alpha)\beta \| \Theta_i^t \|_2^2 + \| (\Theta_i^t, \Theta_i^{t-1}) \|_2^2 \tag{6.3}$$

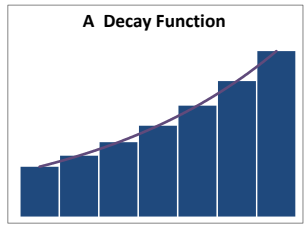
Our evolving model minimizes objective function $\mathcal{L}(\Theta_i)$ to obtain its coefficients at observation time t. Figure 6.1 shows the weighting process of histogram elements of the current technique and our proposed method.

Time Constraints

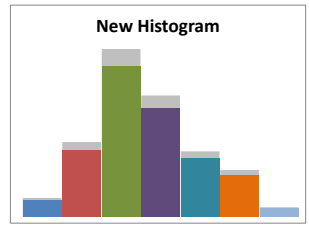
Studying current approaches, it seems that most of them assume that the observations are captured sequentially in time. They only consider the



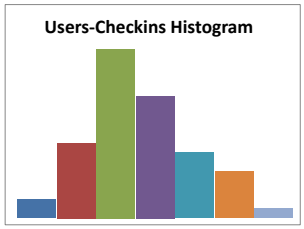
(a) Current histogram of elements



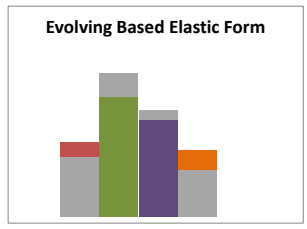
(b) An exponential decay function



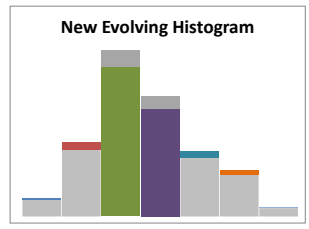
(c) Decay histogram of elements



(d) Current histogram of elements



(e) An evolving process bases on an elastic net model



(f) Evolving histogram of elements

From the original histogram (a), the current method applies an exponential decay function (b), a constant decay, on all the data equally to get a new weighted histogram of elements (c). Our proposed method utilizes the idea of elastic net (e) in order to allow us to study the correlation of different groups of elements in the original histogram (d), and finally get an evolving histogram (f).

Figure 6.1: How a histogram of elements evolves over time.

order of observations, and do not include the time interval in the study but consider the timespan between observations to be constant. This means that $\Delta(t) = \mathcal{T}_t - \mathcal{T}_{t-1}$ is constant, i.e., intervals between sampling are discretely framed. However, such an assumption usually does not hold in practice. To cope with this, we re-define the third term of $\mathcal{L}(\Theta_i)$ when considering the real time of observations of data sampling. We let $\mathcal{T}^t$ be the time between two consecutive observations, i.e., the timespan between current and previous time within which the data are sampled. The intuition is that we split the frame into $\mathcal{T}^t$ equal segments. Further, we assume that there are virtual data sampling observations at each splitting point. Therefore, the deviation penalty is cumulative deviation via a solution path including

$\mathcal{T}^t$ penalties:

$$\Omega_3(\Theta_i^t, \Theta_i^{t-1}) = f(\mathcal{T}^t) \parallel (\frac{\Theta_i^t - \Theta_i^{t-1}}{\mathcal{T}^t}) \parallel_2^2,$$

where $f(.)$ is an interest function of time. The objective function corresponding to each row of histogram matrix Θ now becomes:

$$\mathcal{L}(\Theta_i) = \alpha\beta \parallel \Theta_i^t \parallel_1 + (1-\alpha)\beta \parallel \Theta_i^t \parallel_2^2 + f(\mathcal{T}^t) \parallel (\frac{\Theta_i^t - \Theta_i^{t-1}}{\mathcal{T}^t}) \parallel_2^2$$

Given a histogram matrix Θ of a data stream, the value of Θ at observation time (t-1) is denoted as Θ^{t-1}. Given that Θ evolves over time, at time point t, Θ becomes Θ^t such that the objective function $\mathcal{L}(\Theta_i)$ is minimum. Because Θ^{t-1} is known, in the following we use notation A regrading Θ^{t-1} to indicate that it is constant. Our problem is an optimization problem to minimize the objective function as follows:

$$\begin{aligned} \mathcal{L}(Z,t) = \alpha\beta \parallel Z^t \parallel_1 + (1-\alpha)\beta \parallel Z^t \parallel_2^2 \\ + f(\mathcal{T}^t) \parallel (\tfrac{Z^t - A_i}{\mathcal{T}^t}) \parallel_2^2, \end{aligned} \tag{6.4}$$

where Z is used to denote Θ_i. For the sake of brevity, in the rest of the chapter, we remove subscript of A and drop the argument t. The proposed objective function in (6.4) has a form of an elastic net regularization [179]. Numerous works have been extensively studied elastic net regularization in many applications, such as machine learning and neural networks, but to the best of our knowledge, using elastic net like in the model in Eq. 6.4 for sketching a data stream with concept drift is new.

Optimization Solver

To minimize the objective function in Eq. 6.4, we employ the Alternating Direction Method of Multipliers (ADMM) [24] by controlling each penalty separately. Firstly, the ADMM breaks the objective function in Eq. 6.4 into three smaller parts, such that each part is easier to handle. Thereafter, it solves each part separately while considering the rest as constant. Finally, the global convex optimization problem is solved when all parts are handled.

We use a consensus variable $U = \{U_1, U_2\}$ and scaled dual form variable $V = \{V_1, V_2\}$. We form the augmented Lagrangian with respect to the augmented penalty, $\rho > 0$, which is a penalty parameter of the ADMM. The augmented Lagrangian is as follows:

$$\begin{aligned} \mathcal{L}^\rho(Z,U,V) = (1-\alpha)\beta \parallel Z \parallel_2^2 + f(\mathcal{T}^t) \parallel (\tfrac{U_2 - A}{\mathcal{T}^t}) \parallel_2^2 + \\ \alpha\beta \parallel U_1 \parallel_1 + \tfrac{\rho}{2}\big(\parallel Z - U_1 + V_1 \parallel_2^2 + \parallel Z - U_2 + V_2 \parallel_2^2 \big), \end{aligned}$$

where $U_1 = Z, U_2 = Z$, and $Z_j \geq 0, j \in [1, \ldots, m]$.

The objective has a form of mixed norm regularization with a non-negative constraint on element values of vector Z (weighted histogram sampling). We apply ADMM to our optimization problem. The ADMM alternately estimates (Z, U, V) which results in iteration steps. The iteration of individual penalty solving steps in scaled form is as follows:

$$Z^{(k+1)} = \underset{Z}{\operatorname{argmin}}\ \mathcal{L}^{\rho}(Z,\ U^{(k)},\ V^{(k)}) \tag{6.5}$$

$$U^{(k+1)} = \underset{U_1,U_2}{\operatorname{argmin}}\ \mathcal{L}^{\rho}(Z^{(k+1)},\ U,\ V^{(k)}) \tag{6.6}$$

$$V^{(k+1)} = V^{(k)} + Z^{(k+1)} - U^{(k+1)} \tag{6.7}$$

Let C be the constraint set of non-negative, $C = \{Z : Z_j \geq 0, \text{for } j = 1, \ldots, m\}$, and $\prod_C$ be a projection onto C space. The update U step in Eq. 6.6 involves a projection $\prod_C$ onto set C, rewritten as:

$$U^{(k+1)} = \prod_{C}(Z^{(k+1)} + V^{(k)})$$

A non-negative projection on C is chosen as: $\prod_C(x) = \max(x, 0)$. The primal and dual residuals which control the convergence of the process are determined at iteration step k as $p^{(k)}$ and $d^{(k)}$ and are computed by:

$$\begin{cases} p^{(k+1)} &= Z^{(k+1)} - U^{(k+1)} \\ d^{(k+1)} &= \rho_i^{(k)} \times (U^{(k)} - U^{(k+1)}) \end{cases} \tag{6.8}$$

If the dual residual $d^{(k)}$ approaches zero, then the obtained value of the objective is nearly close to the optimal solution. Meanwhile, the $p^{(k)}$ approaches zero when conditional constraints in the objective function are enforced accurately. We use a maximum number of the iteration steps, l, and a tolerance value, ϵ, on the primal and dual residuals as stopping criteria, given by:

$$\begin{cases} \| p^{(k)} \|^2 &\leq \epsilon \times \max(\| Z^{(k)} \|_2^2, \| U^{(k)} \|_2^2) \\ \| d^{(k)} \|^2 &\leq \epsilon \times \| V^{(k)} \|_2^2 \end{cases} \tag{6.9}$$

An improvement of ADMM in ρ penalty is a relaxed ADMM approach. Relaxed ADMM algorithms have a relaxation parameter γ. After Z-update step in Eq. 6.5, they continue to update Z by: $\gamma Z^{(k+1)} + (1 - \gamma)U^{(k)}$. Like ADMM, the convergence rate of the relaxed ADMM algorithms depends on the choice of the relaxation parameter.

To address the drawback of ADMM and the relaxed variants, an adaptive method for automatically tuning ρ and γ, called Adaptive Relaxed ADMM (ARADMM) [166], was proposed. The penalty and relaxation parameters of ARADMM are continuously updated at each iteration k based on its safeguarding values. (For details on how to update ρ and γ at every iteration step k, please refer to [166]). In addition to the ability to automatically tune its parameters, ARADMM converges fast and is robust. Since it is also suitable for streaming data, we utilize ARADMM in the solver for our optimization problem. The optimal values of the histogram Θ are obtained when the optimization reaches convergence. Thereafter, the value of the corresponding cell of Θ is increased by 1, i.e., $\Theta_{i,h_i(v)} = \Theta_{i,h_i(v)} + 1$, where 1 is used to indicate that it is the weight of the current data in the stream. Θ is continuously estimated and updated following the changes in the stream.

6.4.3 Ensemble Sampling

The histogram of a stream is estimated by a matrix $\Theta \in \mathbb{R}^{n \times m}$. The stream is sketched by a vector SK of s values (s elements) based on the weighted values of the histogram elements in Θ. The value of sketch element SK_i ($1 \leq i \leq s$) is computed using a process with three random variables as in [105, 84]. The process of choosing the sketch values of the stream is done as follows:

$$SK_i = \underset{j \in (1;m)}{\operatorname{argmin}} \, a_{i,j}, \tag{6.10}$$

where

$$a_{i,j} = \frac{c_j}{y_{i,j} \exp(r_j)},$$
$$y_{i,j} = \exp(r_j(\lfloor \frac{\ln(H_j)}{r_j} + \beta_j \rfloor - \beta_j))$$

r_j, c_j, and β_j are the three random variables that are generated as:

$$r_j \sim Gamma\ (2,1),$$
$$c_j \sim Gamma\ (2,1),$$
$$\beta_j \sim Uniform\ (0,1).$$

H_j is the weighted histogram of element range j, which is estimated as a count-min sketch-modeled histogram Θ. It is given by $H_j = min_i(\Theta_{i,h_i(j)})$. In order to get s samples of SK, we perform this process s times with different values of the three random variables r_j, c_j, and β_j. This process corresponds to a single component and can be seen as a weak process [140]. We

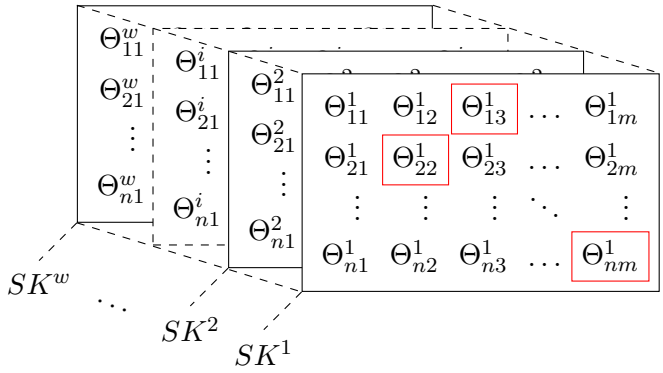


Figure 6.2: Ensemble w components of histograms.

investigate combining several single processes to improve the performance. The ensemble-centric method is used by w components such that each single component C^j sketches Θ^j with a corresponding sketch SK^j. Hence, the estimated sketch will be a $w \times s$ matrix: $SK = [SK^1 \quad SK^2 \quad \dots \quad SK^w]^T$. The ensemble-centric SC of SK is computed based on a scoring function: $SC = Scoring(SK)$, where $Scoring$ is a metric summarizing a sketch from the matrix SK, which can be set as the average, min, max, or a random selection. Figure 6.2 shows an ensemble of w components of histograms of a stream, where each red square in each row i of the matrix is the corresponding histogram of the incoming stream value v_t under hash function h_i. At every observation, a sketch SC of the stream is estimated. Given two sketches SC_1 and SC_2, then the similarity of two sketches is computed using a min-max similarity. The similarity value between the two sketches is used to classify the histograms. For the classification, we use kNN classifier.

6.4.4 Implementation Details

Figure 6.3 shows a flow diagram of the proposed method. The flow diagram is processed as follows.

1. Whenever there is a histogram element arriving in the stream, the corresponding count-min histogram of the element is updated (step $S1$).

2. In the eRSS, the coefficients of the histogram matrix evolve to new optimal values such that they satisfy an objective constraint (Eq. 6.4) corresponding to each individual row of the matrix (step $S2$).

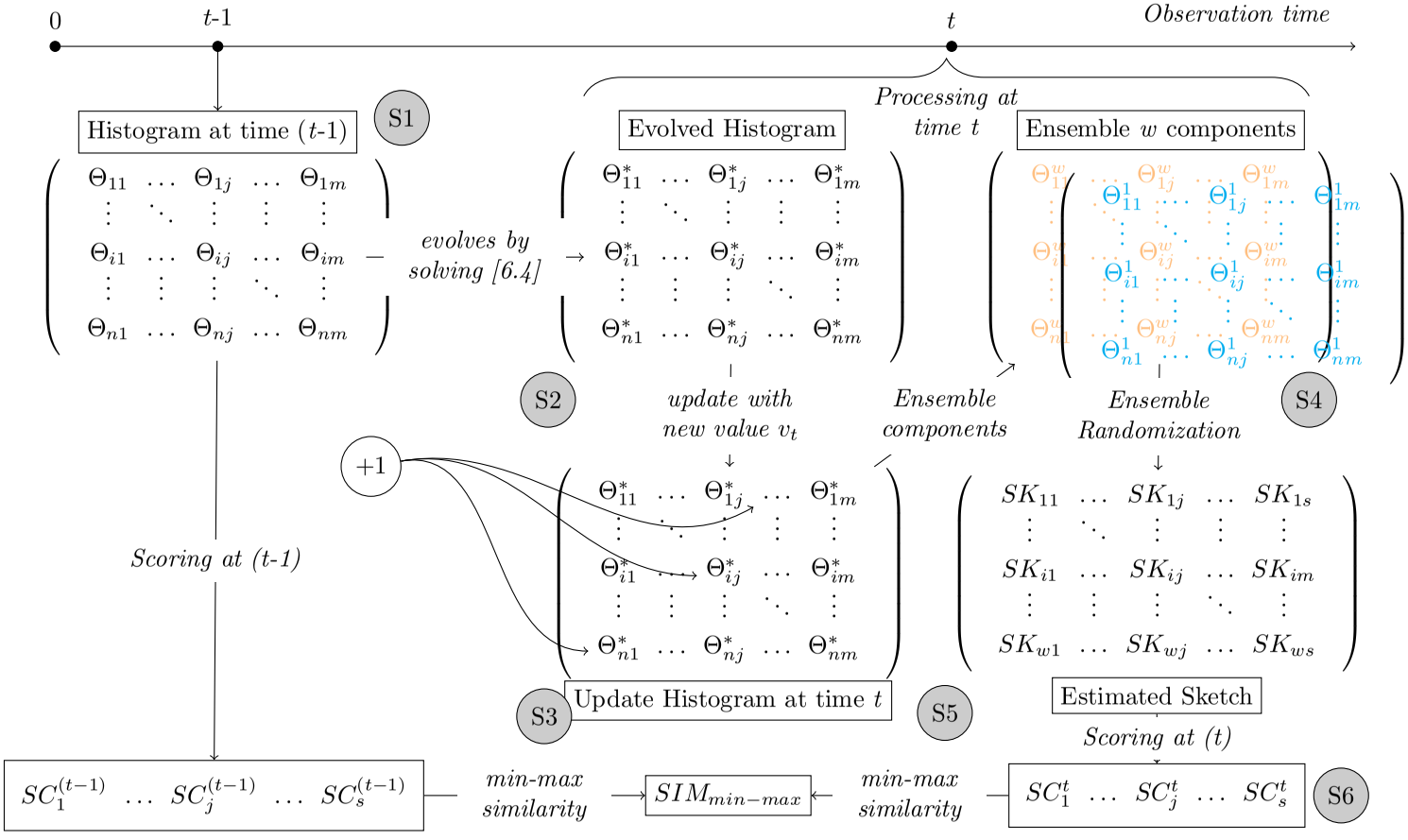


Figure 6.3: Flow diagram of the proposed method.

3. After the evolving step, step $S3$ is processed to update the histogram with the current element.

4. An ensemble w components is used in step $S4$.

5. Each single component uses a weighted minwise hashing method with a different set of random variables to sketch the corresponding histogram to a vector of s elements in step $S5$.

6. The final sketch of the histogram is obtained in step $S6$ by using a random selection.

These individual processes are independent. Thus, they can be implemented in a parallel setting. At the same time, we propose an ensemble method which combines several single processes to obtain a more efficient result. The single processes can also be scheduled into tasks in a parallel configuration. The implementation detail of the proposed algorithm is shown in Algorithm 12.

Complexity Note

For each incoming element in the stream, the major time-consuming task is the *UpdateTheta* step. The execution time of optimization solver for matrix Θ is bounded by $O(nl)$, where l is the maximum number of iterations of ARADMM. In the worst case, the running time to get the count-min sketch of Θ (n rows) and to produce a k element sketch is $O(n + kw)$. In summary, the runtime complexity of the proposed framework for processing each incoming element is $O(nl + n + wk)$.

In terms of space, a count-min sketch Θ of size $n \times m$ takes $O(nm)$ space, while using k elements for sketching streaming histogram in an ensemble w components takes $O(wk)$ space. Overall, the space complexity is $O(nm + wk)$.

6.5 Evaluation

We evaluated the performance of our eRSS algorithm and compared it with the current state-of-the-art methods, HistoSketch[168] and the POISketch[170] algorithms. All the experiments were carried out on a personal computer running the Windows 10, having a 64 bit Intel i7 2.6 GHz processor and 16 GB of RAM, and the algorithms were implemented in Matlab[1] version R2017b.

[1] `http://mathworks.com/products/matlab/`

Algorithm 12 eRSS Algorithm

Input: A stream data S

Output: A sketch SK of S, and change alarm

Initialization (α, β, Θ, SK)
for each data point v_t arriving on a stream S **do**
    UpdateTheta(Θ)
    for i from 1 to n **do**
        Increase cell value: $\Theta_{i,h_i(v_t)} \leftarrow \Theta_{i,h_i(v_t)} + 1$
    for i from 1 to w **do**
        $SK^i \leftarrow SingleRandomization$
    $SC \leftarrow Scoring(SK)$
    if change detected (labeling using kNN) **then**
        Report alarm
procedure UpdateTheta(Θ)
    while not converge by Eq. 6.9 And $++k \leq maxiter$ l **do**
        Update Θ, U, V by Eqs. 6.5-6.7
        Update ρ, and γ (refer to [166])
        Update residuals p and d.
function SingleRandomization
    for k from 1 to s **do**
        for l from 1 to m **do**
            Random sampling three variables r, c, β
            $r \sim Gamma\ (2, 1)$,
            $c \sim Gamma\ (2, 1)$,
            $\beta \sim Uniform\ (0, 1)$
            $y_l \leftarrow \exp(r(\lfloor \frac{\ln(H_l)}{r} + \beta \rfloor - \beta))$
            $a_l \leftarrow \frac{c}{y_l \exp(r)}$
        $SK_k \leftarrow \underset{l}{\operatorname{argmin}}\ a_l$
    return SK

6.5.1 Datasets

We used both synthetic and real-world datasets. The datasets are user check-in activity on locations and are simulated as stream of histogram elements, with the following characteristics:

Synthetic Dataset

We created a synthetic data set that contains 1,000,000 check-ins. We simulated the dataset as a stream of histogram elements of 1,000 histograms. Each histogram has 1,000 elements. The histograms are classified into two distribution classes, and each class has 500 histograms. The classes are generated using Gaussian distributions such that the mean and the variance of the distribution are (10,2) and (11,2), respectively.

Real-world Dataset

We used *POI datasets* consisting of different user check-in data from Foursquare[2]. Specifically, we perform our experiments on user check-ins in America, Japan, and Turkey, because these places are the most checked places by Foursquare users. The datasets are based on 18 months of user check-ins from April 2012 to September 2013, and were provided by [169]. Each POI in the datasets is classified into hierarchical categories by Foursquare[3]. We pre-processed the datasets to remove all POIs that have a small number of check-ins to prevent skewness and sparsity of data. Particularly, we kept POIs that have number of check-ins greater than 100, 200 and 400 times with America, Japan and Turkey dataset, respectively. Considering the visiting time behaviors for POIs and POI types, previous works [170, 168] have shown that the visiting time behaviors for different types of POIs are different. Each kind of POIs might have a specific check-in time. For example, the number of check-ins for a bar is highest during the night, while the check-ins for a park is highest during daytime. Hence, for a fair comparison, we use fine-grained element as a pair of user and check-in time in a week in the HistoSketch method [170] as a histogram element to make it get the best results. In particular, each week is first split into 168 hours (each day has 24 hours, 7 days a week $\times$ 24 = 168 hours). Second, each check-in time is mapped into a range of 168 hours. A pair of user and time range will form an element of a histogram. Since the model used in the eRSS algorithm evolves with respect to time, we use coarse-grained elements of user check-ins. The characteristics of the datasets are summarized in Table 6.1.

[2] `https://developer.foursquare.com/`

[3] `https://developer.foursquare.com/docs/resources/categories`

Table 6.1: Dataset characteristics.

Datasets	Check-ins	No of POIs	No of Users
America	303,647	2,555	16,531
Japan	871,646	1,340	14,052
Turkey	780,657	1,783	15,059
Synthetic	1,000,000	1,000	1,000

6.5.2 Baseline Algorithms

We evaluated the performance of the proposed method, eRSS, and compared the classification accuracy and running time against current state-of-the-art algorithms, POISketch [170] and HistoSketch [168], using the source code provided by the authors. The POISketch algorithm maintains the frequency of histogram elements using Count-Min sketch. It does not consider the weight of the elements in streams. Instead, the POISketch treats the importance of elements in the stream equally. The HistoSketch algorithm considers the weight of the elements by using a constant fading factor. The HistoSketch uses fine-grained elements for similarity preservation. The eRSS uses an automatically tuning coefficient model, combining a random ensemble process to adapt to changes in a stream. We prepared two variants of the eRSS, namely *eRSS-deviation* and *eRSS-evolving*. *eRSS-deviation* is a version where selection parameter β is set to 0, so that it only considers the optimization objective function in the deviation of Θ, and thus fading is utilized on Θ. *eRSS-evolving*, on the other hand, considers both tuning variables α and β.

6.5.3 Configuration Setting of Parameters

We conducted the experiments by varying the number of items in the sketch, $s = 20$, 50, 100, 150, and 200. The number of components w was 4, and we used a random selection in the ensemble components. We used the same configuration setting of the histogram as in [168], $n = 10$ hash functions of range $m = 50$. The decay factor in the HistoSketch was alternately set to 0.02, and 0.01, as done in the original paper. To classify histograms, we used a k-nearest neighbors (kNN) algorithm [119] because of its implementation simplicity, robustness to noise, and immediate adaptation with new training data. We empirically set $k = 5$ (i.e., five nearest neighbors) to test the labels for the streaming histograms, where a weight of a hit is given on its position

in the k-nearest neighbors having the testing label. Specifically, if a nearest neighbor at p-th position order in the k-nearest neighbors has a label equal to the testing label, then the weight of the hit is set to $1/p$. Otherwise, the hit is set to 0.

We split the datasets into 2 subsets, testing set and training set. Then, we recorded the accuracy of the algorithms when labeling the POIs. The testing-training sets are randomly selected as 50%-50% and 10%-90% of POIs, with the synthetic and real-world datasets, respectively. In the experiments with the synthetic dataset, a drift was simulated at point 300K of the stream, and we evolved 25% of 1000 histograms of the stream randomly during the evaluation. Further, we varied the parameters k, α, and β to study the impacts of variables α and β on the sparsity and hierarchy constraint on the optimization solution of the objective function. On real-world datasets, we performed the classification at the last check-in time every month, and the time function in the objective was chosen as square root of timespan in hour as a unit.

6.5.4 Experimental Results

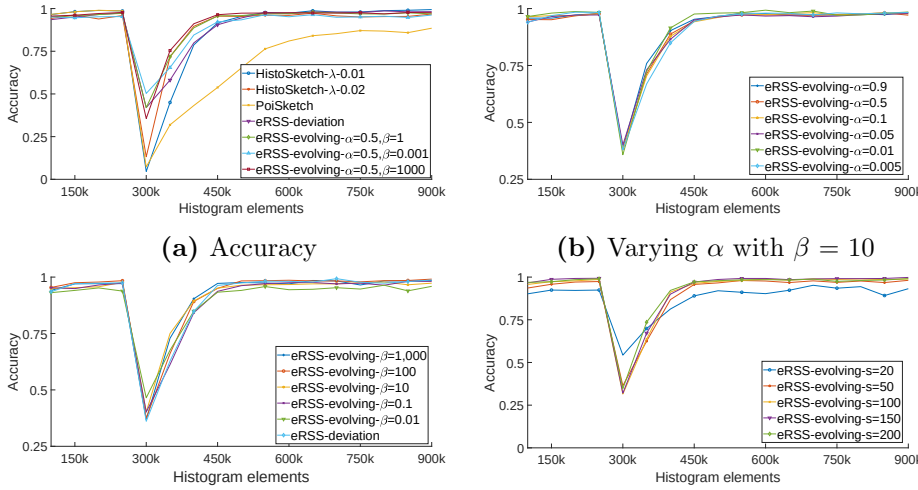
This subsection presents evaluation results of our proposed model, and we compare with the state-of-the-art algorithms.

Synthetic Dataset

Figure 6.4 shows the classification accuracy results on the synthetic dataset when we set the sketch length to 50. As shown in Figure 6.4a, POISketch adapts to a concept drift slowly. It has the worst performance and very low accuracy at the drift point. The reason is that POISketch treats all data in the past equally. Conversely, HistoSketch and eRSS quickly adapt to a concept drift. The classification accuracy of both HistoSketch and eRSS are high, with being the best. We observe that the accuracy values of the algorithms are slightly different at stable points, while the speed of concept drift adaptations are similar. In summary, the average accuracy of eRSS is 91.87%, while HistoSketch and POISketch obtain 89.04%, and 74.64%, respectively.

Furthermore, the results show a large difference in concept drift detection. At a drift point, HistoSketch classifies histogram with a very high error rate. The error rates are 96% and 87% with the values of the forgetting factor $\lambda = 0.01$ and 0.02, respectively. In contrast, with eRSS the error rate is much lower. Specifically, with the *eRSS-evolving* variant with β=0.001, the rate is 50%, and 58% with *eRSS-deviation*. In general, the

accuracy obtained with eRSS is much better than the previous methods. eRSS is an order of magnitude accurate higher than the baseline algorithms with concept drift. Specifically, our experimental results show that at drift points, eRSS has 12.5x times higher accuracy value than the baseline algorithms (50% versus 4%). This can be explained as follows. In streams, when using the same factor for the whole data as HistoSketch does, it might be efficient with stable data, and it can quickly adapt to changes after changes have occurred. However, around drift points, the data distribution changes dramatically, in terms of both internal characteristics and the relationship with other features. This is exactly why considering evolving models with different factors is useful for detecting concept drifts in streams.



(a) Accuracy

(b) Varying α with $\beta = 10$

(c) Varying β with $\alpha = 0.5$

(d) Varying sketch length

Figure 6.4: Classification accuracy on synthetic dataset.

Figures 6.4b-6.4d show the classification accuracy of the *eRSS* when varying the different parameters, and the timespan was set to a unit. The experimental results when varying α, β are shown in Figs. 6.4b-6.4c. In the first test, β was set to 10, and we varied the value of α within [0.9, 0.5, 0.1, 0.05, 0.01, 0.005]. In the second test, α was set to 0.5, and then we varied the value of β within [1,000, 100, 10, 0.1, 0.01, 0]. We observe that when testing on synthetic dataset, with time being generated sequentially and equally, the impact of relaxation parameter is low, and that selecting a small value of β has significant effect on concept drifts. Further, Figure 6.4d plots the classification accuracy when we vary the sketch elements s within [20, 50, 100, 150, 200] while α, β are set to 0.5, 1.0. We observe that the adaptive

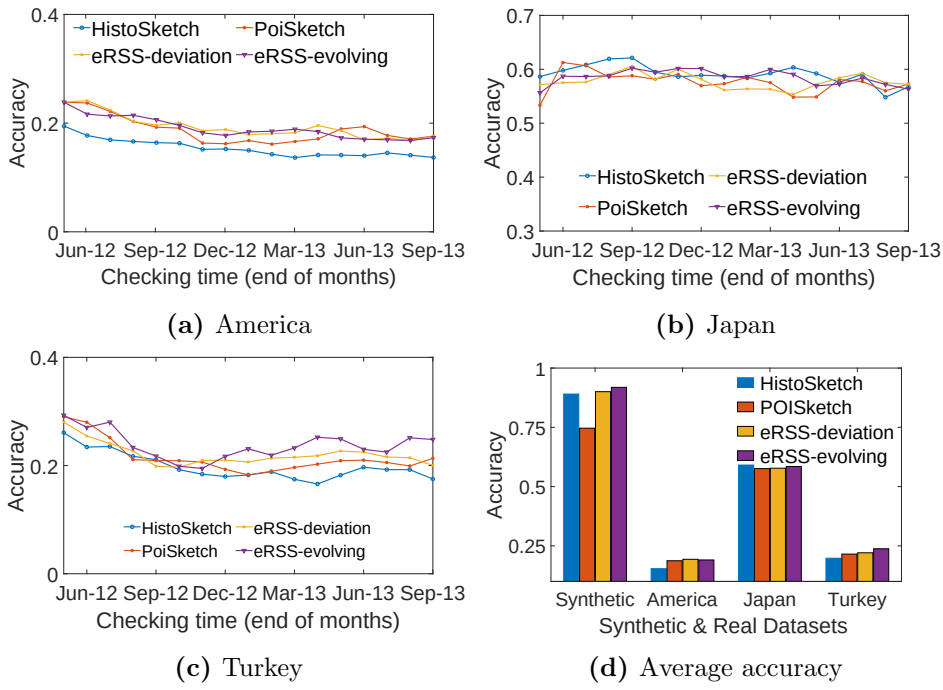
speed of eRSS with concept drift is fast. The result shows that, when the value of s increases, the accuracy at stable points also increases. The lowest value is obtained when $s = 20$, and the highest accuracy is obtained when $s = 200$. However, we need to make a trade-off between accuracy and space complexity, i.e., the larger s is, the larger space is consumed.

Table 6.2: Average runtime (second) and velocity (number of elements per ms).

Datasets	**America**	**Japan**	**Turkey**	**Synthetic**
Histo	24.513	59.512	60.228	60.459
Sketch	12.38	14.64	12.96	16.54
POI	71.329	205.240	168.807	62.711
Sketch	4.26	4.24	4.62	15.95
eRSS	71.411	196.167	169.635	46.815
Deviation	4.25	4.44	4.60	21.36
eRSS	72.816	188.441	175.084	49.055
Evolving	4.17	4.63	4.46	20.39

Real-world POI Datasets

Figure 7.3 shows the classification accuracy of the algorithms on real-world POI datasets. We set the sketch length to 50 and the forgetting factor in HistoSketch to 0.02 as suggested in the original paper, which give the best performance for the HistoSketch algorithm. We empirically set the value of (α, β) in evolving variant of eRSS to (0.5, 100). We observe that on the Japan dataset, POISketch performs the worst, while HistoSketch has the best performance, and both variants of the proposed method have similar results as HistoSketch. Nevertheless, the gap among all the results is very small, around 0.6% between HistoSketch and eRSS. On America and Turkey datasets, on the other hand, the worst accuracy is obtained by HistoSketch, and the eRSS is the most accurate algorithm. These results can be analyzed as follows. The Japan dataset is the most dense dataset with largest number of check-ins and smallest number of users and POIs. This increases collaboration and feature selectivity between check-ins, users and



(a) America **(b)** Japan

(c) Turkey **(d)** Average accuracy

Figure 6.5: Classification accuracy on real datasets.

POIs. Therefore, it might be sufficient to apply the same factor for all data. However, the America and Turkey datasets are very sparse. Hence, the correlation between check-ins, users and POIs is loose or even independent, which, in turn, forms very different check-in behaviors. In the HistoSketch paper, the authors did not seem to have investigated the correlation of different characteristics of data.

Although HistoSketch used fine-grained elements, it applied a constant factor on all the data at different observations. This calls for an evolving model that takes such correlations in a stream into account. The eRSS utilizes an auto-tuning model with respect to different kinds/groups of data to explore the relations between individual check-in behaviors. This's why the eRSS is able to obtain good performance on sparse datasets. Figure 6.5d plots the average accuracy of the algorithms on the experimental datasets. We learn that the eRSS has a good performance and has the best average accuracy on three of four test datasets. Importantly, the eRSS quickly adapts to concept drifts but also preserves the high classification accuracy when drift occurs.

In terms of running time, Table 6.2 displays the execution time and

the number of elements can be processed in the streams per millisecond (ms). Interestingly, it reveals that in [168], HistoSketch traded off accuracy (3.5%) for speed (7500x speedup as compared to Histogram-Fine-Forgetting, a variant of POISketch). Note, however, that the efficiency of HistoSketch in term of running time benefits from kNN classification. The result shows that eRSS is a bit slower than HistoSketch, around 2.9 times, which is as we expected. Nevertheless, it is a trade-off between running time and accuracy and concept drift adaptation speed. In addition, the proposed method is not only efficient with high accuracy classification (3.99% higher than HistoSketch for the Turkey and America datasets) and quickly adapts to changes, it also has a fast execution time (2500x speed up, compared to Histogram-Fine-Forgetting according to [168]). With our experimental setup, eRSS is capable of processing streams at high velocity, up to 4000 check-ins per second.

6.6 Conclusion

In this chapter, we proposed a novel robust method for sketching streaming histograms based on an ensemble randomization method. To obtain the histogram elements, we developed an algorithm called eRSS, which uses an evolving model with adaptive coefficients. To obtain the values of the coefficients, the eRSS algorithm considers the timestamps of different observations in each coefficient and solves an optimization problem. Here, we studied applying Adaptive Relaxed Alternating Direction Method of Multipliers (ARADMM) as a solver for the optimization problem. To evaluate our approach, we performed extensive experiments on both real-world datasets and synthetic dataset. The results from this evaluation showed the effectiveness and the efficiency of the proposed method. More specifically, our algorithm was able to achieve up to 3.99% higher overall classification accuracy than the baseline algorithms. Overall, our evaluation demonstrated our method's ability to preserve the similarity of generated sketches, with the capability to adapt to concept drifts in data streams, in an online fashion.

Part IV

Dense Subregion Detection

This part addresses the problem of event detection from a complex data structure such as a tensor or graph. The study presented in this part focuses on an analysis of structured data, that are used to model user activity behaviors, to identify suspicious behaviors such as fraud, anomalies, and network attacks. In this part, we introduce both theoretical foundations and practical work in order to generalize and guarantee solutions to the dense detection problem; we first address the problem of dense subtensor detection, and then expand our method on graph data. In particular, we introduce a better theoretical density guarantee for both dense subtensor and dense subgraph detection for greedy approximation-based algorithms. We provide proofs for a higher lower bound density for the estimated subtensors and subgraphs, and also introduce a novel theoretical foundation to generalize the detection of multiple dense subtensors with guarantee, in an attempt to answer the following questions: (**1**) *Can the lower bound density be guaranteed higher*? (**2**) *Are there many subtensors or subgraphs with a density greater than the lower bound*? (**3**) *Can we estimate these subtensors and subgraphs*?

Chapter 7

Density Guarantee For Finding Multiple Subgraphs and SubTensors

Extensive studies have shown that an unexpected dense subregion (such as a subgraph or subtensor) is a strong indicator of anomalous behaviors. Dense subregion detection is an efficient approach, and is widely used in the detection of fraudulent behavior. The detection of dense subregions such as subtensors and subgraphs is a well-studied area with a wide range of applications, and numerous efficient algorithms have been proposed that generally perform well in many applications. However, the main drawback of most of these algorithms is that they can estimate only one instance at a time, with a low guarantee of the density. Although some methods can estimate multiple instances, they can give a guarantee of the density with respect to the input data for the first estimated instance only. We address these drawbacks by providing both theoretical and practical solutions for estimating multiple dense subregions in tensor and graph data. We generalize the problem by maintaining multiple dense subregions (i.e., subtensors), provide a concrete proof to guarantee a higher lower bound density, and show that they have a higher density guarantee than solutions in prior works. The contributions presented in this chapter are solutions to research questions RQ1, RQ3, RQ6 and RQ7 [48, 49], i.e., *(i) What are the disadvantages of the state-of-the-art methods? Is it possible to generalize the problem? (ii) How can we provide a more theoretical foundation for generalizing the dense subregion detection problem? (iii) How can we provide both theoretical and practical solutions to the problem?*

7.1 Motivation

In many real-world applications, generated data are commonly represented in complex structures such as graph data or multidimensional array data, that can be referred to as *tensor* [31]. Tensors and graphs have been used in several important domains, including geometry, physics and biology as well as computer science [122, 176, 81, 145]. As a result of the growth in the number of applications involving tensors, graphs, combined with the increase of researchers' interests, numerous tensor, graph-related approaches have been proposed, including tensor decomposition [150, 107], tensor factorization [168, 123, 126], and dense subgraph detection [68, 99, 82].

Dense subregion (subtensor and subgraph) detection have been extremely studied and have attracted much interest due to a wide-range of real-life applications [159, 136, 174, 143]. Finding the densest subtensor or the densest subgraph is generally an NP-complete, or an NP-Hard problem [70, 11, 87], and the hardness of the densest detection problem varies with the choice of constraint requirements, e.g, the size and the dimension of the data, and the chosen density measure. Due to the complexity of the exact algorithm, it is infeasible for large data or in dynamic environments such as streaming. Therefore, the approximation methods are commonly used for detecting the densest subregions [12, 27, 14]. GREEDY is an efficient approximation algorithm that proposed to find the optimal solution in a weighted graph [12]. Charikar [27] introduced a further analysis of the GREEDY, and the analysis showed that the GREEDY method can be solved by using linear programming technique. The authors proposed a greedy 2-approximation for this optimization problem with a density guarantee of the dense subgraph greater than a half of the maximum density in the graph. Tremendous algorithms have adopted the greedy method with a guarantee on the density of dense subgraphs in specific applications such as fraud detection, event detection, and genetics applications [159, 136, 82], among others. Common for these works is that they use the greedy 2-approximation to find a dense subgraph to optimize an objective of a given interest density measure.

Besides graph, tensor has gradually attracted much interest of researchers because the data generated by many sources in real applications can be represented naturally in the form of a tensor. Various algorithms have been proposed by extending the works in dense (sub)graph detection to tensor data for network attack detection, change detection in communication networks, and fraud detection [117, 85, 148, 47], just to name a few. M-Zoom [146] and M-Biz [147] are among the current state-of-the-art dense subtensor detection algorithms. They extend the approaches on dense (sub)graph detec-

tion, such as [27, 62], into tensor detection by considering more dimensions for a specific problem to obtain highly accurate algorithms. Further, they utilize a greedy approach to provide local guarantee for the density of the estimated subtensors. However, the adopted density guarantee is the same as in the original work without any improvement in the density guarantee. M-Zoom and M-Biz are able of maintaining k subtensors at a time. Each time a search is performed, a snapshot of the original tensor is created, and the density of the estimated subtensor in each single search is guaranteed locally on the snapshot. Hence, M-Zoom and M-Biz only provide a density guarantee with respect to the current intermediate tensor rather than the original input tensor. A newer approach, called DenseAlert [149], was developed to detect an incremental dense subtensor for streaming data. Despite its efficiency, however, DenseAlert can estimate only one subtensor at a time, and it can only provide a low density guarantee for the estimated subtensor. Hence, it might miss a huge number of other interesting subtensors in the stream.

Extensive studies have shown that DenseAlert, M-Zoom, and M-Biz generally outperform most other tensor decomposition methods, such as [91, 177], in terms of efficiency and accuracy. Nevertheless, an important drawback of these methods is that they can only provide a loose theoretical guarantee for density detection, and that the results and the efficiency are mostly based on heuristics and empirical observations. More importantly, these methods do not provide any analysis of the properties of multiple estimated subtensors.

Contributions

To give an overview of the differences between our proposed method, namely MUST, and the existing approaches, Table 7.1 compares the characteristics of MUST against current state-of-the-art algorithms (Approx stands for Approximation). In summary, the main contributions of this work are:

1. We introduce a foundation to theoretically guarantee a better density of both estimated subgraph and subtensor in dense subgraph and dense subtensor detection. We provide a new method that is capable of estimating subtensors with a density guarantee that is higher than those provided by existing methods. Specifically,
   - The new density bound for the dense subtensor is $\frac{1}{N}(1 + \frac{N-1}{min(a,\sqrt{n})})$, while the current widely-used bound is $1/N$. Here, n and a denote the size of the tensor and the densest subtensor, respectively, and N is the number of ways of the tensor.

Table 7.1: A brief comparison of between existing algorithms and MUST

	Approx	**Multiple estimation support**	**Single density guarantee**	**Multiple density guarantee**	**Number of guaranteed estimations**	**Bound guarantee***
Goldberg's [70]			✓		1	1
GREEDY [27]	✓		✓		1	$\frac{1}{N}$
GreedyAP [136]	✓		✓		1	$\frac{1}{N}$
M-Zoom [146]	✓	✓	✓		1	$\frac{1}{N}$
DenseAlert [149]	✓		✓		1	$\frac{1}{N}$
M-Biz [147]	✓	✓	✓		1	$\frac{1}{N}$
FrauDar [82]	✓		✓		1	$\frac{1}{N}$
ISG+D-Spot [174]	✓	✓				
MUST	✓	✓	✓	✓	$\boldsymbol{min(1 + \frac{n}{2N}, 1 + N(N\text{-}1))}$	$\frac{1}{N}(1 + \frac{N\text{-}1}{min(a, \sqrt{n})})$

* N is the number of ways of tensor (with graph, we consider its number of ways is 2 because we can represent a graph in a form of matrix). a is the size of the densest subregion.

Data	Goldberg's	GREEDY	GreedyAP	M-Zoom	DenseAlert	M-Biz	FrauDar	ISG+D-Spot	Ours
Tensor				✓	✓	✓			✓
Graph	✓	✓	✓					✓	✓

- For the dense subgraph detection, the new density bound is $\frac{1}{2}(1 + \frac{1}{min(y,\sqrt{n})})$, where n and y denote the size of the graph and the densest subgraph, respectively.

2. We present a novel theoretical foundation, along with proofs showing that it is possible to maintain multiple subtensors with a density guarantee.
3. We prove that there exist at least $min(1 + \frac{n}{2N}, 1 + N(N-1))$ subtensors that have a density greater than a lower bound in the tensor.
4. We perform an extensive experimental evaluation on real-world datasets to demonstrate the efficiency of our solution. The proposed method is up to 6.9 times faster and the resulting subtensors have up to two million times higher density than state-of-the-art methods.

Organization

The rest of this chapter is organized as follows. Section 7.3 describes the preliminaries for the method and the related work. Sections 7.4-7.5 elaborate on the theoretical foundation for providing a new density guarantee of dense subtensors and a better density guarantee of dense subgraphs. Section 7.6 presents the solution for detecting multiple dense subtensors with a density guarantee. Section 7.7 discusses the evaluation of our method and explains its applicability. Finally, Section 7.8 concludes the chapter.

Reproducibility: The source code and data used in the chapter are publicly available at `https://bitbucket.org/duonghuy/mtensor`.

7.2 Related Work

The problem of finding the densest subgraphs is generally NP-complete or NP-hard [70, 11]. Due to the complexity of the exact algorithm with which an exponential number of subgraphs must be considered, it is infeasible for large datasets or data streams. Therefore, approximation methods are commonly used for detecting the densest subregions [12, 27, 14]. Ashiro et al. [12] proposed an efficient greedy approximation algorithm to find the optimal solution for finding the densest subgraph in a weighted graph. Their idea is to find a k-vertex subgraph of an n-vertex weighted graph with the maximum weight by iteratively removing a vertex with the minimum weighted-degree in the currently remaining graph, until there are exactly k vertices left. Charikar [27] studied the greedy approach (GREEDY) further, which showed that the approximation can be solved by using linear programming technique. Specifically, the author proposed a greedy 2-approximation for this optimization problem, with which a density guarantee of the dense

subgraph is greater than a half of the maximum density in the graph. Many algorithms have later adopted the greedy method with a guarantee on the density of dense subgraphs targeting specific applications, such as fraud detection, event detection, and genetics applications [111, 159, 136, 82].

Inspired by the theoretical solutions in graphs, numerous approaches have been proposed to detect dense subtensors by using the same min-cut mechanism [149, 147]. As mentioned earlier, mining the densest subtensor in a tensor is hard, and an exact mining approach has a polynomial time complexity [70], thus making it infeasible for streaming data or very large datasets. To cope with this, approximate methods/algorithms are commonly used. Among the proposed algorithms, DenseAlert [149], M-Zoom [146], and M-Biz [147] are – because of their effectiveness, flexibility, and efficiency – the current state-of-the-art methods. They are far more faster than other existing algorithms, such as CPD [91], MAF [117], and CrossSpot [85]. DenseAlert, M-Zoom, and M-Biz adapt the theoretical results from dense (sub)graph detection, i.e., [10, 9, 159], to tensor data by considering more dimensions than two. The algorithms utilize a greedy approach to guarantee the density of the estimated subtensors, which has also been shown to yield high accuracy in practice [85]. However, the adopted density guarantee is the same as in the original work, which also applies for the more recent algorithm, *ISG+D-Spot* [174]. This means that with an N-way tensor, the density guarantee is a fraction of the highest density with the number of the tensor's way N. *ISG+D-Spot* converts an input tensor to a form of graph to reduce the number of ways, but it drops all edges having weight less than a threshold. As a result, *ISG+D-Spot* only provides a loose density guarantee.

The greedy 2-approximation approach has been utilized in many algorithms with both types of data, graph and tensor [154, 138, 137, 143]. Despite of that, most of current works, including [82, 120, 138, 147, 174] can only roughly provide a guarantee by $\frac{1}{2}$ (with graph), and $\frac{1}{N}$ (with tensor) density of the densest subregion. To the best of our knowledge, none of the existing approximation works provides a better guarantee than the baseline algorithms [12, 27]. They can only provide a loose theoretical density detection guarantee. As discussed in Section 7.1, DenseAlert, M-Zoom, and M-Biz employed the same guarantee as in the original work without any further improvement in the density guarantee. Thus, these methods can only guarantee low density subtensors. To address the limitations of the previous approaches, we generalize the problem by maintaining multiple dense subtensors, with which we provide a concrete proof to guarantee a higher lower bound density and show that they have a higher density guarantee

than the solutions in prior works.

7.3 Preliminaries

In the following, we present the fundamental preliminaries of the dense subtensor, subgraph detection problem, based on [149, 147].

Dense Subgraph Detection

Definition 37 (Graph)**.** *Let G be an undirected graph that is composed of a pair $(V; E)$ of sets of vertices V and edges E. We denote the graph as $G(V; E)$. There is a weight a_i at each vertex v_i, and a weight c_{ij} on each edge e_{ij} between two vertices v_i and v_j in G.*

Definition 38 (Density of Graph)**.** *The density of G is denoted by $\rho(G)$ and is defined by: $\rho(G) = \frac{\sum a_i + \sum c_{ij}}{|V|} = \frac{f(G)}{|V|}$, where $|V|$ is the number of vertices of G, and $f(G) = \sum a_i + \sum c_{ij}$, $f(G)$ is called the mass of graph G.*

Definition 39 (Subgraph)**.** *Let G be an undirected graph that is composed of a pair $(V; E)$ of sets of vertices V and edges E. S is a subgraph of G if S is induced by a subset of the vertices of V and edges in E.*

Definition 40 (Weight of vertex in Graph)**.** *Consider a graph $G(V, E)$ with a weight a_i at vertex v_i, and a weight c_{ij} on edge between 2 vertices v_i, v_j. The weight of vertex v_i in graph G is denoted by $w_i(G)$, and is defined by: $w_i(G) = a_i + \sum_{v_j \in G \wedge e_{ij} \in E} c_{ij}$.*

Definition 41 (Dense Subgraph Detection)**.** *Given an undirected graph $G = (V; E)$ and a density measure df, the problem of dense subgraph detection involves finding subgraphs S induced by a subset of vertices of V and edges in G to maximize the density of S.*

The processing of the greedy approximation algorithm is as follows [12, 27]. The algorithm iteratively removes a vertex with the minimum weighted-degree in the currently remaining graph until all vertices are removed. Finally, it picks the highest density subgraph among the estimated subgraphs. The algorithm gives a 2-approximation with a density guarantee of a half of the highest density. Note that, in the original work, the density measure is average of weighted-degree of the graph. In this chapter, we consider a general density measure of both weights at vertices and on edges.

Dense Subtensor Detection

Several dense subtensor detection methods have been proposed by extending the works in dense (sub)graph detection to tensor data. However, they use the same min-cut mechanism as in dense subgraph detection [148, 147, 174]. These methods employed the same guarantee as in the original work without any improvement in density guarantee. In this chapter, we generalize the problem in both dense subtensor and dense subgraph detection and propose our new theoretically proofs to give a better approximation guarantee of the density. In the rest of this chapter, we use subregion to indicate both subtensor and subgraph.

Definition 42 (Tensor)**.** *A tensor T is a multidimensional array data. The order of T is the number of its ways. Given a N-way tensor, there are multiple spaces on each way, each of which is called a slice.*

Definition 43 (SubTensor)**.** *Given an N-way tensor T, Q is a subtensor of T if it is composed of a subset s of the set of slices S of T, and there is at least one slice on each way of T. Intuitively, Q is the part of T that remains after we remove all slices in S but not in s.*

Definition 44 (Entry of a Tensor)**.** *E is an entry of an N-way (sub)tensor T if it is a subtensor of T and is composed of exactly N slices.*

Definition 45 (Size of a (Sub)Tensor)**.** *Given a (sub)Tensor Q, the size of Q is the number of slices that making up Q.*

Definition 46 (Density)**.** *Given a (sub)tensor Q, the density of Q, denoted by $\rho(Q)$, is computed as: $\rho(Q) = \frac{f(Q)}{size\ of\ Q}$, where $f(Q)$ is the mass of the (sub)tensor Q, and is calculated as the sum of every entry value of Q.*

Definition 47 (Weight of a Slice in a Tensor)**.** *Given a tensor T, the weight of a slice q in T is denoted by $w_q(T)$, and is defined as the sum of the entry values composing the intersection of T and q.*

Definition 48 (D-Ordering)**.** *An ordering π on a (sub)tensor Q is a D-Ordering, if*

$$\forall q \in Q, q = \underset{p \in Q \wedge \pi^{-1}(p) \geq \pi^{-1}(q)}{argmin} w_p(\pi_q), \tag{7.1}$$

where $\pi_q = \{x \in Q | \pi^{-1}(x) \geq \pi^{-1}(q)\}$, $\pi^{-1}(q)$ indicates the index of the slice q in ordering π, and $w_p(\pi_q)$ is the weight of p in π_q. Intuitively, the D-Ordering is the order in which we select and remove the minimum slice sum at each step.

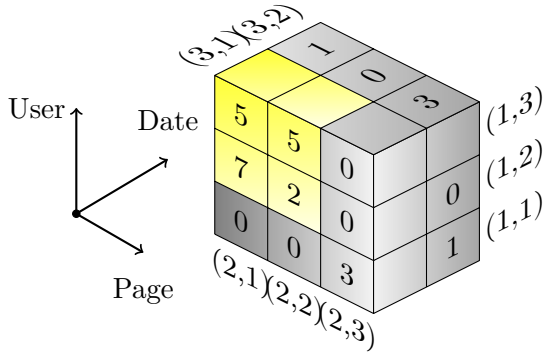


Figure 7.1: An example of 3-way tensor.

The principal of D-Ordering in tensor data is the similar to the min-cut mechanism in dense subgraph detection, like GREEDY [12, 27].

Definition 49 (Mining of Dense Subtensors)**.** *Given a tensor T, the problem of dense subtensor detection involves finding subtensors $Q \in T$ that maximize the density of Q.*

For readability, the notations used in this chapter are summarized in Table 7.2. In the rest of this chapter, when specifying a (sub)tensor, we use its name or set of its slices interchangeably.

Example 6. *Let us consider an example of 3-way tensor T as in Figure 7.1. The value in each cell is the number of visits that a user (mode User) visits a web page (mode Page) on a date (mode Date). The values of hidden cells are all zero. The set of slices of tensor T is {(1,1), (1,2), (1,3), (2,1), (2,2), (2,3), (3,1), (3,2)}. A subtensor Q formed by the following slices {(1,2), (1,3), (2,1), (2,2), (3,1)} is the densest subtensor (the yellow region) and the density of Q is (5+5+7+2)/5 = 3.8.*

The problem of mining dense subtensors [149, 147] can be presented and solved as follows. Given a list of n variables $d_\pi(i)$ $(1 \leq i \leq n)$, where $d_\pi(i)$ is calculated during the construction of D-Ordering. Its value at each time is picked by the minimum slice sum of the input (sub)tensor. Then, a *Find-Slices*() function finds the index $i^* = \underset{1 \leq i \leq n}{\operatorname{argmax}}\, \rho_\pi(i)$, which is the location to guarantee a subtensor with a density greater than the lower bound. *Find-Slices()*, shown in Algorithm 13, is a function that was originally defined in [146, 149, 147], which is a principal function for estimating a subtensor,

Table 7.2: Table of notations

Symbols	Description
T, Q	Tensor data T, Q
I_i	The i-*th* dimension of tensor I
$\|I_i\|$	Number of slices on way I_i of a tensor I
T^*	Densest subtensor T^*
G	(Sub)Graph data G.
G^*	Densest Subgraph.
v_i, a_i	Vertex v_i, and weight a_i at vertex v_i.
c_{ij}	Weight on edge between two vertices v_i and v_j.
Z, z_0	Zero subtensor Z with zero point z_0
B	Backward subtensor
F	Forward subtensor
n, N	Size (with tensor, it is number of slices, with graph, it is number of vertices), and number of ways of data
ρ, ρ^*	Density ρ, highest density ρ^*
$\rho(Q)$	Density of Q
π	An ordering π
$Q(\pi, i)$	A subtensor of Q formed by a set of slices $\{p \in Q, \pi^{-1}(p) \geq i\}$
$\rho_\pi(i)$	Density of subtensor $Q(\pi, i)$
q	A slice of a tensor
a	Size of densest subtensor
b	Number of slices in Zero subtensor such that not in densest subtensor
y	Size of densest subgraph
m	Size of Zero subtensor Z, $m = a + b$
$f(Q)$	Mass of the (sub)tensor Q
$w_q(Q)$	Weight of element q (vertex, or slice) in data Q (graph, or tensor).

such that its density is greater than the lower bound. The density of an estimated subtensor is guaranteed as follows.

Theorem 3 (Density Guarantee) [149, 147])**.** *The density of the subtensor returned by the Algorithm 13 is greater than or equal to* $\frac{1}{N}\rho^*$*, where* ρ^* *is the highest density in the input tensor.*

Algorithm 13 Find-Slices

Input: A D-Ordering π on a set of slices Q
Output: An estimated subtensor S

```
1: S ← ∅, m ← 0
2: ρ_max ← −∞, q_max ← 0
3: for (j ← |Q|..1) do
4:     q ← π(j), S ← S ∪ q
5:     m ← m + d_π(q)
6:     if m/|S| > ρ_max then
7:         ρ_max ← m/|S|
8:         q_max ← q
9: return Q(π, π^{-1}(q_max))
```

Proof. The proof of this theorem was provided in [149, 147]. For convenience, we recall their proof as follows. Let $q^* \in T^*$ be the slice such that $\pi^{-1}(q^*) \leq \pi^{-1}(q), \forall q \in T^*$. This means that q^* is the slice in the densest subtensor having the smallest index in π. Therefore $\rho_\pi(i^*) \geq \rho_\pi(\pi^{-1}(q^*)) \geq \frac{1}{N}\rho^*$. □

7.4 The New Density Guarantee of Subtensor

As can be inferred from the discussion above, the basic principle underlying DenseAlert, M-Zoom, and M-Biz is Theorem 3. It is worth noting that this theorem guarantees the lower bound of the density on only one estimated subtensor from an input tensor. To the best of our knowledge, none of existing approximation approaches provides a better density guarantee than GREEDY. Based on this, we can raise the following questions: (1) Can this lower bound be guaranteed higher? (2) Are there many subtensors having density greater than the lower bound? (3) Can we estimate these subtensors?

In this section, we answer question (1) by providing a proof for a new higher density guarantee. Questions (2) and (3) will be answered in the next section by providing a novel theoretically sound solution to guarantee the estimation of multiple dense subtensors that have higher density than the lower bound.

7.4.1 A New Bound of Density Guarantee

We prove that the estimated subtensors provided by the proposed methods have a higher bound than in the state-of-the-art solutions.

In [149, 147], the authors proved that density of the subtensor $\rho_\pi(\pi^{-1}(q^*)) \geq \frac{1}{N}\rho^*$, hence satisfying Theorem 3. A sensible question is: Can we estimate several subtensors with a higher density guarantee than the state-of-the-art algorithms?

In the following subsections, we introduce our new solution to improve the guarantee in the aforementioned *Find-slices()* function and show how a density with higher lower bound than that in [149, 147] can be provided. We present several theorems and properties to support our solution to estimate multiple dense subtensors.

Definition 50 (Zero Subtensor)**.** *Given a tensor T, T^* is the densest subtensor in T with density ρ^*, π is a D-ordering on T, and $z_0 = \min_{q \in T^*} \pi^{-1}(q)$ is the smallest indices in D-Ordering π of all slices in T^*. A subtensor called Zero Subtensor of T on π, denoted as $Z = T(\pi, z_0)$, and z_0 is called zero point.*

Theorem 4 (Lower Bound Density of the Estimated Subtensor)**.** *Given an N-way tensor T, and a D-ordering π on T. Let Z and z_0 be a Zero Subtensor and a zero point, respectively. Then, there exists a number $b \geq 0$ such that the density of the estimated subtensor Z is not less than $\frac{Na+b}{N(a+b)}\rho^*$, where a and ρ^* are the size and density of the densest subtensor T^*.*

Proof. We denote $w_0 = w_{\pi(z_0)}(Z)$. Further, note that because T^* is the densest subtensor. Then,

$$\forall q \in T^*, w_q(T^*) \geq \rho^* \Rightarrow w_0 \geq \rho^*.$$

Due to the characteristic of D-Ordering, we have

$$w_q(Z) \geq w_{\pi(z_0)}(Z) = w_0, \forall q \in Z.$$

Consider a way I_i among the N ways of the tensor T. Then,

$$f(Z) = \sum_{q \in T^* \wedge q \in I_i} w_q(Z) + \sum_{q \notin T^* \wedge q \in I_i} w_q(Z).$$

Furthermore, regarding the way we choose Z, we have

$$T^* \subseteq Z \Rightarrow \sum_{q \in T^* \wedge q \in I_i} w_q(Z) \geq \sum_{q \in T^* \wedge q \in I_i} w_q(T^*) = f(T^*).$$

Therefore,

$$f(Z) \geq f(T^*) + \sum_{q \notin T^* \wedge q \in I_i} w_q(Z) \geq f(T^*) + b_{I_i} w_0, \tag{7.2}$$

where b_{I_i} is the number of slices in Z on dimension I_i that are not in T^*. Let $b = \sum_{i=1}^{N} b_{I_i}$. Applying Eq. 7.2 on N ways, we get

$$\begin{aligned} & Nf(Z) && \geq Nf(T^*) + w_0 \sum b_{I_i} \\ \Rightarrow & N(a+b)\rho(Z) && \geq Na\rho^* + w_0 b \\ \Rightarrow & N(a+b)\rho(Z) && \geq Na\rho^* + b\rho^* \\ \Rightarrow & \rho(Z) && \geq \frac{Na+b}{N(a+b)}\rho^* \end{aligned}$$ □

The equality happens when $b = 0$ or in the simple case when $N = 1$. However, if these conditions hold, the Zero Subtensor becomes the densest subtensor T^*. In the next paragraphs, we consider the higher order problem of tensor with order $N > 1$.

Property 10. *The lower bound density in Theorem 4 is greater than $\frac{1}{N}$ of the highest density and this bound is within $[\frac{1}{N}(1 + \frac{a(N-1)}{n}), 1]$.*

Proof. Let Z be the fraction of the density of the estimated subtensor, and R denote densest subtensor. We have the following properties about the lower bound fraction:

1. In the simplest case, when $N = 1$, the lower bound rate values both in the previous proof and in this proof are 1. This means that the estimated subtensor Z is the densest subtensor, with the highest density value. Otherwise,

$$R \geq \frac{Na+b}{N(a+b)} = \frac{a+b}{N(a+b)} + \frac{(N-1)a}{N(a+b)} > \frac{1}{N}, \forall N > 1. \tag{7.3}$$

   Moreover, since the size of Z is not greater than n, we have

$$R \geq \frac{1}{N}(1 + \frac{(N-1)a}{(a+b)}) \geq \frac{1}{N}(1 + \frac{a(N-1)}{n}) \tag{7.4}$$

2. In conclusion, we have the following boundary of the density of estimated Zero Subtensor, Z:

$$\rho(Z) = \begin{cases} \rho^*, & \text{if } N = 1 \vee b = 0 \\ \frac{1}{N}(1 + \frac{a(N-1)}{n})\rho^*, & \text{if } a + b = n. \end{cases}$$

In an ideal case, when the value of b goes to zero, the estimated subtensor becomes the densest subtensor, and its density can be guaranteed to be the highest. □

7.4.2 A New Higher Density Guarantee

In this subsection, we provide a new proof to give a new higher density guarantee of dense subtensor.

Theorem 5 (Upper Bound of the Min-Cut Value in Tensor)**.** *Given an N-way tensor T with size n, and a slice q is chosen for the minimum cut, such that the weight of q in T is minimum. Then, the weight of q in T satisfies the following inequality:*

$$w_q(T) \leq N\rho(T) \tag{7.5}$$

Proof. Because q is a slice having the minimum cut, we have $w_q(T) \leq w_p(T), \forall p \in T$. Summing all the slices in the tensor gives

$$\begin{aligned} |T|w_q(T) &\leq \sum_{p \in T} w_p(T) = Nf(T) \\ \Rightarrow w_q(T) &\leq \frac{Nf(T)}{|T|} = N\rho(T) \end{aligned}$$ □

Let $T_i (1 \leq i \leq a)$ be the subtensor right before we remove i-th slice of T^*, and q_i be the slice of T^* having the minimum cut w_i at the step of processing T_i. Since the size of the densest T^* is a, we have a indexes from 1 to a. Note that T_1 is the Zero subtensor Z. Further, let M_{I_i} denote the index of the last slice in way I_i of T^* that will be removed. Then, we have following property:

Property 11 (Upper Bound of the Last Removed Index)**.** *The minimum index of all $M_{I_i}, 1 \leq i \leq N$, denoted by M, is not greater than $(a - N + 1)$, i.e., $M = min(M_{I_i}) \leq a - N + 1$.*

Proof. Let M_{I_i}, M_{I_j} be the indexes of the last removed slices of the two ways I_i and I_j. Further, assume that the difference between M_{I_i}, M_{I_j} is $\Delta(M_{I_i}, M_{I_j}) = |M_{I_i} - M_{I_j}| \geq 1$, and that we have N numbers (N ways) and the maximum (the last index) is a. Then, we get

$$\begin{aligned} max(M_{I_i}) - min(M_{I_i}) &\geq N - 1 \\ \Rightarrow M = min(M_{I_i}) &\leq a - N + 1 \end{aligned}$$ □

Theorem 6. *The sum of min-cut of all slices from index 1 to M is greater than the mass of the densest subtensor T^*:*

$$\sum_{i=1}^{M} w_{q_i}(T_i) \geq f(T^*) \tag{7.6}$$

Proof. Let E be any entry of the densest subtensor T^* and E is composed by the intersection of N slices, $q_{I_x}(1 \leq x \leq N)$, q_{I_x} is on the way I_x.

Assume that the first removed index of all the slices composing E is at index i. Since this index cannot be greater than M, the entry E is in T_i, and its value is counted in $w_{q_x}(T_x)$. Therefore, we have:

$$\sum_{i=1}^{M} w_{q_i}(T_i) \geq f(T^*) \qquad \square$$

Let ρ_{max} be the maximum density among all subtensors $T_i, (i \leq i \leq M)$. According to Theorems 5 and 6, we have

$$f(T^*) \leq \sum_{i=1}^{M} w_{q_i}(T_i) \leq \sum_{i=1}^{M} N\rho(T_i) \leq MN\rho_{max} \tag{7.7}$$

$$\Rightarrow a\rho^* \leq N(a - N + 1)\rho_{max} \tag{7.8}$$

$$\Rightarrow \rho_{max} \geq \frac{\rho^*}{N}\frac{a}{a - N + 1}. \tag{7.9}$$

Theorem 7 (Better Density Guarantee of Dense Subtensor)**.** *The density guarantee of dense subtensor mining by min-cut mechanism is greater than* $\frac{1}{N}(1 + \frac{N-1}{min(a,\sqrt{n})})\rho^*$.

Proof. According to Theorem 4 and Property 10, we have

$$\rho_{max} \geq \rho(T_1) \geq \frac{1}{N}(1 + \frac{a(N-1)}{n})\rho^* \tag{7.10}$$

Furthermore, by Inequation 7.9, we also have

$$\rho_{max} \geq \frac{\rho^*}{N}\frac{a}{a - N + 1} \geq \frac{1}{N}(1 + \frac{N-1}{a})\rho^* \tag{7.11}$$

By combining Eq. 7.10 and Eq. 7.11, we get

$$\rho_{max} \geq \frac{1}{N}(1 + \frac{1}{2}(\frac{a(N-1)}{n} + \frac{N-1}{a}))\rho^*$$
$$\Rightarrow \rho_{max} \geq \frac{1}{N}(1 + \frac{N-1}{\sqrt{n}})\rho^*$$

Note that since $\rho_{max} \geq \frac{1}{N}(1 + \frac{N-1}{a})\rho^*$, we finally have

$$\rho_{max} \geq \frac{1}{N}(1 + \frac{N-1}{min(a,\sqrt{n})})\rho^* \qquad \square$$

An Illustrated Example

Let's consider an example of 3-way tensor T as in Figure 7.2. The value in each cell is the number of requests that a user (probably an attacker, in mode Attacker) sends to a server (mode Server) in a period of time (mode Time). The values in the hidden cells are all zeros. Our task is to analyze the data to detect attackers. The set of slices of tensor T is {(1,1), (1,2), (1,3), (2,1), (2,2), (2,3), (3,1), (3,2)}. Subtensor Q formed by the following slices {(1,2), (1,3), (2,1), (2,2), (3,1)} is the densest subtensor (the red region), and the density of Q is (5+5+7+3)/5 = 4.0. Here the number of ways of T is 3, and its size (number of slices that composes T) is 8.

The existing methods can only give a guarantee of the estimated subtensor as a fraction of the highest density. The guarantee in this case is $\frac{1}{N}\rho^* = \frac{4}{3}$. However, by using our new proof, we proved that the new lower bound of density in this example is guaranteed to be greater than $\frac{1}{N}(1 + \frac{N-1}{min(a,\sqrt{n})})\rho^* \geq \frac{4}{3}(1 + \frac{3-1}{\sqrt{8}}) = \frac{2+\sqrt{2}}{2}\frac{4}{3} \geq \frac{1.7\times4}{3}$. In comparison between two guarantees, our proposed guarantee on the density is 1.7 ($\simeq 1 + \frac{\sqrt{2}}{2}$) times greater than the guarantee by the existing methods. So, our proposed guarantee is more than 70% higher than the current guarantee.

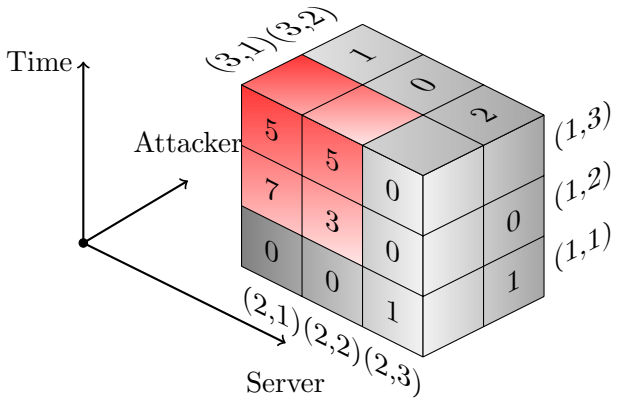


Figure 7.2: An illustrated example of high density guarantee.

7.5 The New Density Guarantee of Subgraph

As aforementioned, tremendous algorithms have adopted the greedy method with a guarantee on the density of dense subgraphs in specific applications

such as fraud detection, event detection, and genetics applications [159, 136, 82, 148, 138], among others. The common of these works is that they use the greedy 2-approximation to find a dense subgraph to optimize an objective of a given interest density measure. Numerous methods have been proposed later using the same min-cut mechanism as in dense subgraph detection [148, 147, 174] for the dense subtensor detection problem. These methods employed the same guarantee as in the original work without any improvement in density guarantee. So these methods can only provide a loose theoretical guarantee for density detection. In this study, we generalize the problem in both dense subtensor and dense subgraph detection. We propose our new theoretical proofs to give a better approximation guarantee of the density. In Section 7.4, we proved and provided a new bound on the density in a tensor data. Now we raise the following questions with the original problem of detecting dense subgraph: (1) Can we provide a higher guarantee on the density of the dense estimated subgraph in a graph? (2) Is this bound constrained to any other information rather than the dimension of the data space? This section answers the question by introducing our proofs to give a better approximation guarantee on the density of the estimated subgraph in a graph, that is the original foundation for both dense subgraph and dense subtensor detection problems. Our novel mathematical proof here is capable of giving a better guarantee for the current state-of-the-art methods, and shows that the bound is also constrained to the size of the densest subgraph.

Theorem 8 (Density Guarantee of Dense Subgraph Detection)**.** *Given an undirected graph $G(V;E)$ with size $n = |V|$. Let G^* be the densest subgraph in G. There exists a number $p \geq 0$ such that the lower bound density of estimated subgraph in the GREEDY [27] is $\frac{2y+p}{2(y+p)}\rho^*$, where ρ^* is the density of the densest subgraph G^*, y is the size of G^*, and $y \leq (y+p) \leq n$.*

Proof. Let G_1 be the subgraph that is right before we pick the first vertex of the densest subgraph G^* to be removed, we denote the vertex is v_{s1}. So definitely we have: $G^* \subseteq G_1$, and the size of G_1 is $n_1 \leq n$. We have:

$$
\begin{aligned}
2f(G_1) &= 2\sum_{v_i \in G_1} a_i + 2\sum_{v_i, v_j \in G_1} c_{ij} \\
&= \sum_{v_i \in G_1} a_i + \sum_{v_i \in G_1} w_i(G_1) \\
&= \sum_{v_i \in G_1} a_i + \sum_{v_i \in G_1 \wedge v_i \in G^*} w_i(G_1) + \sum_{v_i \in G_1 \wedge v_i \notin G^*} w_i(G_1)
\end{aligned}
$$

Let's denote $V(G_1 \backslash G^*) = \{v_i, v_i \in G_1 \wedge v_i \notin G^*\}$ and $p = |V(G_1 \backslash G^*)|$. Because v_{s1} is chosen for the cut, it means that v_{s1} has the minimum cut weight, so we get: $w_j(G_1) \geq w_{s1}(G_1), \forall v_j \in G_1$, and G^* is the densest subgraph then $w_{s1}(G^*) \geq \rho^*$. Therefore:

$$\sum_{v_i \in G_1 \wedge v_i \notin G^*} w_i(G_1) \geq p \times w_{s1}(G_1) \geq p \times w_{s1}(G^*) \geq p \times \rho^*.$$

On the other hand, we have:

$$\begin{aligned} \sum_{v_i \in G_1} a_i + \sum_{v_i \in G_1 \wedge v_i \in G^*} w_i(G_1) \quad &\geq \sum_{v_i \in G^*} a_i + \sum_{v_i \in G_1 \wedge v_i \in G^*} w_i(G^*) \\ &\geq 2\Big(\sum_{v_i \in G^*} a_i + \sum_{v_i, v_j \in G^*} c_{ij}\Big) \\ &\geq 2f(G^*) \end{aligned}$$

Note that, size of G_1 is $n_1 = y + p$, finally we have:

$$\begin{aligned} & 2f(G_1) && \geq 2f(G^*) + p \times \rho^* \\ \Rightarrow\ & 2(y+p)\rho(G_1) && \geq 2y\rho^* + p \times \rho^* \\ \Rightarrow\ & \rho(G_1) && \geq \frac{2y+p}{2y+2p} \times \rho^*, \end{aligned}$$

where ρ^* is the highest density and $y \leq n_1 = (y + p) \leq n$. The theorem is proved. □

Theorem 9 (Density Guarantee Boundary)**.** *The density of the subgraph G_1 as in Theorem 8 is $\rho(G_1)$, and this density is in $[\frac{1}{2}(1 + \frac{y}{n})\rho^*, \rho^*]$, where ρ^* is the highest density in G.*

Proof. Because ρ^* is the highest density so $\rho(G_1) \leq \rho^*$. Moreover, by Theorem 8, we have (because $n_1 = y + p \leq n$):

$$\begin{aligned} \frac{\rho(G_1)}{\rho^*} &\geq \frac{2y+p}{2y+2p} = \frac{1}{2}(1 + \frac{y}{y+p}) \qquad (7.12) \\ \Rightarrow \rho(G_1) &\geq \frac{1}{2}(1 + \frac{y}{n})\rho^* \end{aligned}$$

□

According to Theorem 9, the density of the subgraph G_1 is in the boundary $[\frac{1}{2}(1 + \frac{y}{n})\rho^*, \rho^*]$. We denote $G_1, G_2, \ldots, G_m$ are subgraphs right before we are going to remove vertex $v_1, v_2, \ldots, v_y$ of G^*. Intuitively, G_i is the subgraph right before we remove i-*th* vertex of G^*. The corresponding min-cut at the step of processing G_i is denoted as w_i. We have a following property about the min-cut value.

Property 12 (Upper Bound Of The Min-Cut Value In Graph)**.** *Given an undirected graph $G(V, E)$ with vertex v_i having the minimum cut (its weight is minimum). The weight of vertex v_i in graph G satisfies the following inequality:*

$$w_i(G) \leq 2\rho(G) - \overline{a}(G), \tag{7.13}$$

where $\overline{a}(G) = \frac{\sum_{v_k \in G} a_k}{|V|}$ is the average weight of all vertices in G.

Proof. Because v_i is a vertex having the minimum cut, so we have $w_i(G) \leq w_k(G), \forall v_k \in G$. Sum up of all the vertices in the graph, we get:

$$\begin{aligned}
|V|w_i(G) &\leq \sum_{v_k \in G} w_k(G) = \sum_{v_k \in G} a_k + 2 \sum_{v_k, v_j \in G} c_{kj} \\
\Rightarrow |V|w_i(G) &\leq 2(\sum_{v_k \in G} a_k + \sum_{v_k, v_j \in G} c_{kj}) - \sum_{v_k \in G} a_k \\
\Rightarrow w_i(G) &\leq \frac{2(\sum_{v_k \in G} a_k + \sum_{v_k, v_j \in G} c_{kj}) - \sum_{v_k \in G} a_k}{|V|} \\
\Rightarrow w_i(G) &\leq 2\rho(G) - \overline{a}(G).
\end{aligned}$$

□

Let ρ_{max} be the maximum density among subgraphs G_i,

$$\rho_{max} = max(\rho(G_i)) \tag{7.14}$$

We have:

$$\begin{aligned}
\sum_{i=1}^{y-1} w_i(G_i) + a_y &= w_1(G_1) + w_2(G_2) + \cdots + w_{y-1}(G_{y-1}) + a_y \\
&\geq \sum_{v_i \in G^*} a_i + \sum_{v_i, v_j \in G^*} c_{ij} = f(G^*)
\end{aligned}$$

if we assume that $a_n = 0$ as in the GREEDY algorithm [27], or in many other works in the literature, they assume that weight at vertices are zero [82, 174], so we have:

$$\sum_{i=1}^{y-1} w_i(G_i) \geq f(G^*) \tag{7.15}$$

$$\Rightarrow 2(y-1)\rho_{max} \geq y\rho^* \tag{7.16}$$

$$\Rightarrow \rho_{max} \geq \frac{y}{2(y-1)}\rho^* \tag{7.17}$$

$$\Rightarrow \rho_{max} \geq \frac{1}{2}(1 + \frac{1}{y})\rho^* \tag{7.18}$$

Theorem 10 (Better Density Guarantee of Dense Subgraph)**.** *The density guarantee of dense subgraph mining by the min-cut mechanism is greater than* $\frac{1}{2}(1+\frac{1}{min(y,\sqrt{n})})\rho^*$*, where* ρ^* *is the highest density value in the graph.*

Proof. According to Theorem 8, we have:

$$\rho_{max} \geq \rho(G_1) \geq \frac{1}{2}(1+\frac{y}{n})\rho^*. \tag{7.19}$$

Furthermore, note that we have

$$\rho_{max} \geq \frac{1}{2}(1+\frac{1}{y})\rho^*, \text{ by Inequation [7.18]}$$

We combine together two inequations [7.18-7.19], we get:

$$\rho_{max} \geq \frac{1}{2}(1+\frac{1}{2}(\frac{y}{n}+\frac{1}{y}))\rho^*$$
$$\Rightarrow \rho_{max} \geq \frac{1}{2}(1+\frac{1}{\sqrt{n}})\rho^*$$

Note that $\rho_{max} \geq \frac{1}{2}(1+\frac{1}{y})\rho^*$, so finally we have:

$$\rho_{max} \geq \frac{1}{2}(1+\frac{1}{min(y,\sqrt{n})})\rho^* \qquad \square$$

7.6 The Solution For Multiple Dense Subtensors

As shown in Theorem 4, $\rho(Z) \geq \frac{Na+b}{N(a+b)}\rho^*$, where $Z = T(\pi, z_0)$ is the Zero subtensor. As discussed before, the state-of-the-art algorithm, DenseAlert, can estimate only one subtensor at a time, and a density guarantee is low, i.e., $\frac{1}{N}$ of the highest density. M-Zoom (or M-Biz) is, on the other hand, able of maintaining k subtensors at a time by repeatedly calling the *Find-Slices()* function k times, with the input (sub)tensor being a snapshot of the whole tensor (i.e., the original one). Recall, however, that such processing cannot guarantee any density boundary of the estimated subtensors with respect to the original input tensor. Therefore, the estimated density of the subtensors is very low. With this, an important question is: How many subtensors in n subtensors of D-ordering as in Algorithm 13 having density greater than a lower bound density and what is the guarantee on the lower bound density with respect to highest density? This section answers this question.

7.6.1 Forward Subtensor from Zero Point

Again, given a tensor T, T^* is the densest subtensor in T with density ρ^*. π is a D-ordering on T, and the zero point $z_0 = \min_{q \in T^*} \pi^{-1}(q)$ is the smallest indices in π among all slices in T^* (cf. Definition 50).

Definition 51 (Forward Subtensor)**.** *A subtensor is called i-Forward subtensor in T on π, denoted by F_i, if $F_i = T(\pi, z_0 - i), 0 \leq i < z_0$.*

Let us consider an i-forward subtensor $F_i = T(\pi, i), i < z_0$. Because $i < z_0$, $Z \subseteq F_i$. This means that $f(F_i) \geq f(Z)$. As a result of Theorem 4, we have the following:

$$\begin{aligned} & Nf(Z) && \geq (Na+b)\rho^* \\ \Rightarrow\ & (Na+b)\rho^* && \leq Nf(Z) \leq Nf(F_i) \\ \Rightarrow\ & (Na+b)\rho^* && \leq N(a+b+i)\rho(F_i) \\ \Rightarrow\ & \rho(F_i) && \geq \frac{Na+b}{N(a+b+i)}\rho^*. \end{aligned}$$

From the above inequality, we get the following theorem.

Theorem 11. *The density of every i-Forward subtensor $F_i = T(\pi, i)$, where $i \leq N \times (N-1)$ is greater than or equal to $1/N$ of the highest density in T, ρ^*.*

Proof. From the above inequality, $\rho(F_i) \geq \frac{Na+b}{N(a+b+i)}\rho^*$.
If we have $i \leq N(N-1)$, then

$$\begin{aligned} \Rightarrow\ & a+b+i && \leq a+b+N(N-1) \\ \Rightarrow\ & \frac{Na+b}{N(a+b+i)}\rho^* && \geq \frac{Na+b}{N(a+b+N(N-1))}\rho^* \\ \Rightarrow\ & \frac{Na+b}{N(a+b+i)}\rho^* && \geq \frac{a+b+a(N-1)}{N(a+b+N(N-1))}\rho^* \\ \Rightarrow\ & \frac{Na+b}{N(a+b+i)}\rho^* && \geq \frac{a+b+N(N-1)}{N(a+b+N(N-1))}\rho^* \\ \Rightarrow\ & \rho(F_i) && \geq \frac{Na+b}{N(a+b+i)}\rho^* \geq \frac{1}{N}\rho^* \end{aligned}$$

□

Property 13. *Among n subtensors $T(\pi, i), 1 \leq i \leq n$, there is at least $min(z_0, 1 + N(N-1))$ subtensors having a density greater than $\frac{1}{N}$ of the densest subtensor in T.*

Proof. According to Theorem 11, there is at least $min(z_0, 1 + N(N-1))$ forward subtensors that have density greater than $\frac{1}{N}$ of the highest density. □

7.6.2 Backward Subtensor from Zero Point

We have considered subtensors formed by adding more slices to Z. Next, we continue investigating the density of the subtensors by sequentially removing slices in Z.

Definition 52 (Backward Subtensor)**.** *A subtensor is called i-Backward subtensor in T on π, denoted by B_i, if $B_i = T(\pi, z_0 + i), i \geq 0$.*

Let us consider an i-backward subtensor B_i. We show that its density is also greater than the lower bound density.

Property 14. *The density of the 1-Backward Subtensor, B_1 is greater than or equal to $\frac{1}{N}\rho^*$.*

Proof. Due to the limitation of space, we omit the proof and provide it in an extension supplement upon request. □

Theorem 12. *Let B_k denote the k-Backward subtensor, $B_k = T(\pi, z_0 + k)$. Density of B_k is greater than or equal to $1/N$ of the highest density in $T, \forall k \leq \frac{b}{N}$.*

Proof. Note that $f(B_i) = f(B_{i+1}) + w_{\pi(z_0+i)}(B_i)$. Let $B_0 = Z$, and in the following we let $w_i(B_i) = w_{\pi(z_0+i)}(B_i)$ for short. Then, we have

$$\begin{aligned} Kf(Z) &= K(f(B_1) + w_0(B_0)) \\ &= K(f(B_2) + w_0(B_0) + w_1(B_1)) \\ &= Kf(B_k) + K\sum_{i=0}^{k-1} w_i(B_i). \end{aligned}$$

Because $T^* \subseteq Z$, then:

$$Kf(Z) \geq Kf(T^*) + \sum_{q \in Z \wedge q \notin T^*} w_q(Z), \tag{7.20}$$

By substitution, we get

$$\begin{aligned} Kf(B_k) + K\sum_{i=0}^{k-1} w_i(B_i) \quad &\geq Kf(T^*) + \sum_{q\in Z \wedge q\notin T^*} w_q(Z) \\ \Rightarrow Kf(B_k) \quad &\geq Kf(T^*) + \sum_{q\in Z \wedge q\notin T^*} w_q(Z) - K\sum_{i=0}^{k-1} w_i(B_i). \end{aligned}$$

We denote the set $Q = \{q|\ q \in Z \wedge q \notin T^*\}$ by $\{q_1, q_2, \ldots, q_b\}$. Note that $B_i \subseteq Z$. Thus $\forall j, i, w_{q_j}(Z) \geq w_{q_j}(B_i) \geq w_i(B_i)$, and $w_{\pi(z_0)}(Z) \geq w_{\pi(z_0)}(T^*) \geq \rho^*$.

On the other hand, we have the condition of k: $b - k \times K \geq b - k \times N \geq 0$. In conclusion, this gives the following inequality:

$$\begin{aligned} Kf(B_k) - Kf(T^*) \quad &\geq \sum_{q\in Z \wedge q\notin T^*} w_q(Z) - K\sum_{i=0}^{k-1} w_i(B_i) \\ &\geq \sum_{i=0}^{k-1}\sum_{j=1}^{K} w_{q_{(i\times K+j)}}(Z) - Kw_i(B_i) + \sum_{i=k\times K+1}^{b} w_{q_i}(Z) \\ &\geq (b - k \times K) \times E_{\pi(z_0)}(Z) \\ &\geq (b - k \times K)\rho^* \\ \Rightarrow K\rho(B_k)(a + b - k) \quad &\geq Ka\rho^* + (b - k \times K)\rho^* \\ \Rightarrow \rho(B_k) \quad &\geq \frac{Ka + b - k \times K}{K(a + b - k)}\rho^* \\ \Rightarrow \rho(B_k) \quad &\geq \frac{K(a - k) + b}{K(a + b - k)}\rho^* \\ \Rightarrow \rho(B_k) \quad &\geq \frac{1}{K}\rho^* \geq \frac{1}{N}\rho^*. \end{aligned}$$

□

Theorem 13. *Assume that the size of the Zero subtensor Z, $(a + b)$, is sufficiently big. Let B_k denote the k-Backward subtensor. The density of B_k is greater than or equal to $1/N$ of the highest density in $T, \forall k \leq min(\frac{a}{N}, \frac{(a+b)(N-1)}{N^2})$.*

Proof. Assume I_x is the way that has the smallest number of slices in T^*, with a number of slices s. Then, $s \leq a/N$.

Let $Q = \{q \in Z\} = \{q_1, \ldots, q_s, \ldots, q_a, \ldots, q_{a+b}\}$, denote the set of slices in Z, and $(a + b)$ be the size of the Zero subtensor.

Let B_k be a k-Backward Subtensor of T, with $1 \leq k \leq \frac{(a+b)}{N}$. Then,

$$Nf(Z) = \sum_{i=1}^{s} w_{q_i}(Z) + \sum_{i=s+1}^{a+b} w_{q_i}(Z) \geq f(T^*) + \sum_{i=s+1}^{a+b} w_{q_i}(Z).$$

Because $Nf(Z) = N(f(B_k) + \sum_{i=0}^{k-1} w_i(B_i))$, the above inequality can be rewritten as

$$\Rightarrow N(f(B_k) + \sum_{i=0}^{k-1} w_i(B_i)) \geq f(T^*) + \sum_{i=s+1}^{a+b} w_{q_i}(Z).$$

The subtensor B_i is a backward subtensor of Z by removing i slices in Z, i.e., $B_i \subseteq Z$ and $\forall j, i, E_{q_j}(Z) \geq E_{q_j}(B_i) \geq E_{\pi(z_0+i)}(B_i)$. Hence,

$$\begin{aligned} Nf(B_k) \quad &\geq f(T^*) + \sum_{i=s+1}^{a+b} w_{q_i}(Z) - N\sum_{i=0}^{k-1} w_i(B_i) \\ &\geq f(T^*) + \sum_{i=0}^{k-1}\sum_{j=1}^{N} w_{q_{(s+i\times N+j)}}(Z) - Nw_i(B_i) + \sum_{i=s+k\times N+1}^{a+b} w_{q_i}(Z) \\ &\geq f(T^*) + (a+b-kN-s)w_{\pi(z_0)}(Z). \end{aligned}$$

Because

$$\begin{aligned} a+b-kN-s &\geq a+b-kN-\frac{a}{N} \\ &\geq \frac{(a+b)(N-1)+b}{N} - kN \\ &\geq 0, \forall k \leq \frac{(a+b)(N-1)}{N^2}, \end{aligned}$$

we have

$$\begin{aligned} &Nf(B_k) \geq a\rho^* + (a+b-kN-s)\rho^* \\ &Nf(B_k) \geq (2a+b-kN-s)\rho^* \\ &\Rightarrow \rho(B_k) \geq \frac{(2a+b-kN-s)}{N(a+b-k)}\rho^* \\ &\Rightarrow \rho(B_k) \geq \frac{1}{N}\frac{2a+b-kN-s}{a+b-k}\rho^* \\ &\Rightarrow \rho(B_k) \geq \frac{1}{N}\frac{(a+b-k)+(a-k(N-1)-a/N)}{a+b-k}\rho^* \\ &\Rightarrow \rho(B_k) \geq \frac{1}{N}(1+\frac{(a-kN)(N-1)}{N(a+b-k)})\rho^* \\ &\Rightarrow \rho(B_k) \geq \frac{\rho^*}{N}, \forall k \leq \frac{a}{N}. \end{aligned}$$

□

7.6.3 Multiple Dense Subtensors with High Density Guarantee

In this subsection, we show that there exist multiple subtensors that have density values greater than a lower bound in the tensor.

Theorem 14. *Given an N-way tensor T with size $n >> N$, an order π is a D-Ordering on T, and Algorithm 13 processes $m = (n - N)$ subtensors. Then, there are at least $min(1 + \frac{n}{2N}, 1 + N(N-1))$ subtensors among m subtensors, such that they have density greater than $1/N$ of the highest density subtensor in T.*

Proof. Let Z denote the Zero subtensor of T on π by Algorithm 13, and the zero index is z_0, such that $N \leq n - z_0$. Then, we have the following:

1. By Theorem 11, there are at least $min(N(N-1), z_0)$ forward subtensors $F_1, F_2, \ldots,$ having density higher than $\frac{1}{N}\rho^*$.

2. By Theorems 12-13, there are backward subtensors $B_1, B_2, \ldots,$ having density higher than $\frac{1}{N}\rho^*$. The principle of the number of backward subtensors having density greater than $\frac{1}{N}$ of the highest density is as follows:

$$\begin{cases} \frac{b}{N}, & \text{by Theorem 12.} \\ min(\frac{a}{N}, \frac{(a+b)(N-1)}{N^2}), & \text{by Theorem 13.} \end{cases} \tag{7.21}$$

From Eq. 7.21, there is at least $max(\frac{b}{N}, min(\frac{a}{N}, \frac{(a+b)(N-1)}{N^2}))$ backward subtensors having density greater than the lower bound.

If $\frac{a}{N} \leq \frac{(a+b)(N-1)}{N^2}$, then number of backward subtensors having density greater than the lower bound is at least $max(\frac{a}{N}, \frac{b}{N}) \geq \frac{a+b}{2N}$.

Otherwise, we have

$$min(\frac{a}{N}, \frac{(a+b)(N-1)}{N^2}) = \frac{(a+b)(N-1)}{N^2} \geq \frac{a+b}{2N}.$$

Hence, the number of backward subtensors is at least $\frac{a+b}{2N}$. Further, if we combine this with the number of forward subtensors, then there is at least $min(1 + \frac{n}{2N}, 1 + N(N-1))$ subtensors in the tensor having density greater than a lower bound. This can be proved as follows.

According to Theorem 13, we have the number of backward subtensors having density greater than the lower bound, denoted by bw, and $bw \geq \frac{(a+b)}{2N}$. By Theorem 11, we have the number of subtensors having density

greater than the lower bound, we denote this by fw, and $fw \geq min(N(N-1), z_0)$.

If $z_0 \geq N(N-1)$, then the number of subtensors that have density values greater than a lower bound is $1 + fw + bw \geq 1 + N(N-1)$, where 1 is used to account for the zero subtensor. Otherwise (i.e., $z_0 \leq N(N-1)$), we have $a + b + z_0 = n$, and we get

$$\begin{aligned} 1 + fw + bw &\geq 1 + \frac{(a+b)}{2N} + z_0 \\ \Rightarrow 1 + fw + bw &\geq 1 + \frac{(n-z_0)}{2N} + z_0 \\ \Rightarrow 1 + fw + bw &\geq 1 + \frac{n}{2N} + \frac{z_0(2N-1)}{2N} \\ \Rightarrow 1 + fw + bw &\geq 1 + \frac{n}{2N}. \end{aligned}$$

This gives that the number of subtensors having density values greater than the lower bound is $1 + fw + bw \geq min(1 + \frac{n}{2N}, 1 + N(N-1))$.

If $(a + b) \leq n - N(N-1)$, then we have at least $N(N-1)$ forward subtensors having density greater than $\frac{1}{N}$ of the highest density.

Otherwise, if $n >> N$ such that

$$\begin{aligned} (a+b) &\geq n - N(N-1) \geq 2N^3 \\ \Rightarrow \text{then we get } \frac{(a+b)}{2N} &\geq N(N-1). \end{aligned}$$

In conclusion, we have at least $N(N-1)$ backward subtensors, each having density greater than $\frac{1}{N}$ of the highest density. By adding the zero subtensors, we have at least $(1+N(N-1))$ subtensors having density greater than $\frac{1}{N}$ of the highest density each. □

Our approach described above can be employed to improve the state-of-the-art algorithms on estimating multiple dense subtensors using Algorithm 14.

Complexity Discussion. In order to estimate k dense subtensors, the complexity of M-Zoom and M-Biz are high. The worst-case time complexity of M-Zoom and M-Biz is $O(kNn\log n)$ [147]. Its complexity increases linearly with respect to the number of estimated subtensors, k.

Focusing on the proposed solution, MUST, the complexity includes the cost of D-Ordering, which is $O(Nn\log n)$, and the cost of executing Algorithm 14, which utilizes Google Guava ordering[1], is $O(n\log n)$, in the worst

[1]https://opensource.google.com/projects/guava

Algorithm 14 Multiple Estimated Subtensors

Input: A D-Ordering π on a set of slices Q of tensor T
Output: Multiple estimated subtensors with guarantee on density

```
1: Initialization()
2: TS ← ∅,  S ← ∅
3: Number of estimated subtensors: mul ← 0
4: mul ← min(1 + n/2N, 1 + N(N − 1))
5: for (j ← |Q|..1) do
6:     q ← π(j)
7:     S ← S ∪ q
8:     TS.add (S, ρ(S))
9: Sort TS by descending order of density
10: return top mul subtensors having highest density in TS
```

case. In total, the complexity MUST is $O(Nnlogn)$, which does not depend on the number of estimated subtensors k.

7.7 Evaluation

In this section, we present the results from our experimental evaluation, where we evaluate the performance of our proposed method in terms of both the execution time (i.e., efficiency) and the accuracy of the density of the estimated subtensors (i.e., effectiveness).

7.7.1 Experimental Setup

We used four widely-used density measures in our experiments: arithmetic average mass (ρ_a) [27]; geometric average mass (ρ_g) [27]; entry surplus (ρ_e) [159], with which the surplus parameter α was set to 1 as default; and suspiciousness (ρ_s) [85]. Note that in M-Zoom (M-Biz), Dense-Alert, and in this work, the density measure used for the proof of guarantee is arithmetic average mass. Nevertheless, the only difference among the density measures is the choice of coefficients. Hence, we can utilize the same proof for other mass measures to get similar results. In this work, we specifically provide theoretical proofs for density guarantee of dense subtensors. Here, it is worth noting that we can easily extend and apply our proofs of higher density for dense subgraphs, as well.

We implemented our approach based on the implementation used in the previous approaches [146, 149, 147]. We compared the performance of

the proposed solution with the state-of-the-art algorithms, M-Zoom and M-Biz (where M-Zoom was used as the seed-subtensor). To do this, in our experiments, we run the algorithms using M-Zoom, M-Biz, and MUST to get top 10 subtensors that have the highest density. We carried out all the experiments on a computer running Windows 10 as operating system, having a 64 bit Intel i7 2.6 GHz processor and 16GB of RAM. All the algorithms were implemented in Java, including M-Zoom and M-Biz, the source codes for which were provided by the authors[2].

7.7.2 Datasets

In order to evaluate the performance of the proposed solution and compare it with the state-of-the-art algorithms, we used the following 10 real-world datasets:

- *Air Force*, which contains TCP dump data for a typical U.S. Air Force LAN. The dataset was modified from the KDD Cup 1999 Data and was provided by Shin et al. [147].
- *Android*, which contains product reviews and rating metadata of applications for Android from Amazon [78].
- *Darpa*, which is a dataset collected by MIT Lincoln Lab to evaluate the performance of intrusion detection systems (IDSs) in cooperation with DARPA [108].
- *Enron Emails*, provided by the Federal Energy Regulatory Commission to analyze the social network of employees during its investigation of fraud detection and counter terrorism.
- *Enwiki* and *Kowiki* provided by Wikipedia[3]. Enwiki and Kowiki are metadata representing the number of user revisions on Wikipedia pages at given times (in hour) in English Wikipedia and Korean Wikipedia, respectively.
- *LBNL-Network*, which consists of internal network traffic captured by Lawrence Berkeley National Laboratory and ICSI [125]. Each instance contains the packet size that a source (ip, port) sends to a destination (ip, port) at a time.
- *NIPS Pubs*, which contains papers published in NIPS[4] from 1987 to 2003 [69].
- *StackO*, which represents data of users and posts on the Stack Overflow. Each instance contains the information of a user marked a post as favorite

[2]https://github.com/kijungs/mzoom
[3]https://dumps.wikimedia.org/
[4]https://nips.cc/

at a timestamp [97].
- *YouTube*, which consists of the friendship connections between YouTube users [118].

The *Air Force* dataset was modified from the KDD Cup 1999 Data[5]. We kept fields (features) such that each instance has a structure of (*protocol_type*, *service*, *flag*, *src_bytes*, *dst_bytes*, *count*, *srv_count*) as described in Table 7.3, while other fields were removed. The *Android* dataset was obtained from Stanford Network Analysis Project at this address[6]. The *Darpa* dataset was provided in the prior work, DenseAlert [149], and we downloaded the dataset at this address[7]. The *Enron Emails*, *NIPS Pubs*, and *LBNL-Network* were directly downloaded from an open source project, The Formidable Repository of Open Sparse Tensors and Tools (FROSTT) [151], at this address[8]. The *StackO* and *YouTube* were directly downloaded from The Koblenz Network Collection repository [97], and we got the datasets at this address[9]. The Kowiki and Enwiki datasets were downloaded from Wikipedia. We selected these datasets because of their diversity, and because they are widely used as benchmark datasets in the literature [149, 147]. We selected these datasets because of their diversity, and because they are widely used as benchmark datasets in the literature [149, 147]. A more detailed information about the datasets are listed in Table 7.3.

7.7.3 Density of the Estimated Subtensors

Figure 7.3 shows the density of the estimated subtensors obtained with M-Zoom, M-Biz, and MUST. In the figure, we plot the average (AVG) and the low boundary (BOUND) density of the top-10 estimated subtensors. As shown, although the estimated subtensors found by M-Zoom and M-Biz have guarantee locally on the snapshot, the density of the subtensors drops dramatically with respect to the increasing number of the estimated subtensors, k. On all the datasets, the average and the bound density of the estimated subtensors with MUST are much higher than those obtained with M-Zoom and M-Biz in all density measures. MUST also outperforms M-Zoom and M-Biz on density accuracy of estimated subtensors, focusing on both the average and boundary of density of the top ten estimated subtensors.

[5] http://kdd.ics.uci.edu/databases/kddcup99/kddcup.data.gz

[6] http://snap.stanford.edu/data/amazon/productGraph/categoryFiles/ratings_Apps_for_Android.csv

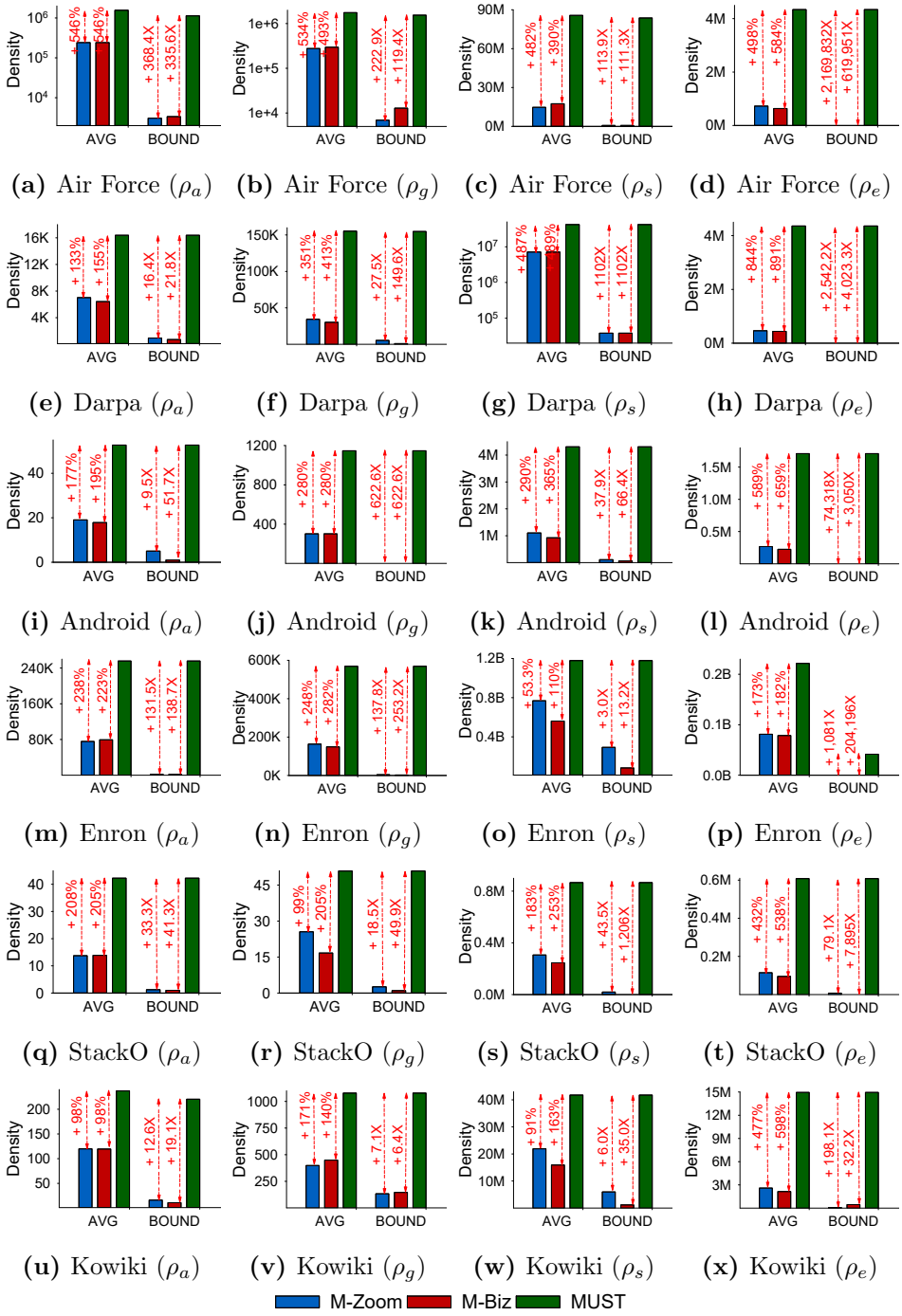
[7] http://www.cs.cmu.edu/ kijungs/codes/alert/data/darpa.zip

[8] http://frostt.io/tensors/

[9] http://konect.uni-koblenz.de/downloads/tsv

Table 7.3: Summary of the real-world datasets used in the experiments

Dataset	Instance Structure	Entry	Size	#Instances	#Ways	Data Type
Air Force	(protocol, service, flag, s-bytes, d-bytes, counts, srv-counts, connects)	connects	3×70×11×7,195× 21,493×512×512	4,898,431	7	TCP Dumps
Android	(user, application, score, date, rate)	rate	1,323,884×61,275× 5×1,282	2,638,173	4	Ratings
Darpa	(s-ip, d-ip, date, connects)	connects	9,484×23,398× 46,574	4,554,344	3	TCP Dumps
Enron Emails	(sender, receive, word, date, count)	count	6,066×5,699× 244,268×1,176	54,202,099	4	Text Social Network
Enwiki	(user, page, time, revisions)	revisions	4,135,167× 14,449,530×132,079	57,713,231	3	Activity Logs
Kowiki	(user, page, time, revisions)	revisions	662,370×1,918,566× 125,557	21,680,118	3	Activity Logs
LBNL Network	(s-ip, s-port, d-ip, d-port,date, packet)	packet	1,605×4,198×1,631× 4,209×868,131	1,698,825	5	Network
NIPS Pubs	(paper, author, word, year, count)	count	2,482×2,862× 14,036×17	3,101,609	4	Text Academic
StackO	(user, post, favourite, time)	favourite	545,195× 96,678×1,154	1,301,942	3	Activity Logs
YouTube	(user, user, connected, date)	connected	1,221,280× 3,220,409×203	9,375,374	3	Social Network



(a) Air Force (ρ_a) **(b)** Air Force (ρ_g) **(c)** Air Force (ρ_s) **(d)** Air Force (ρ_e)

(e) Darpa (ρ_a) **(f)** Darpa (ρ_g) **(g)** Darpa (ρ_s) **(h)** Darpa (ρ_e)

(i) Android (ρ_a) **(j)** Android (ρ_g) **(k)** Android (ρ_s) **(l)** Android (ρ_e)

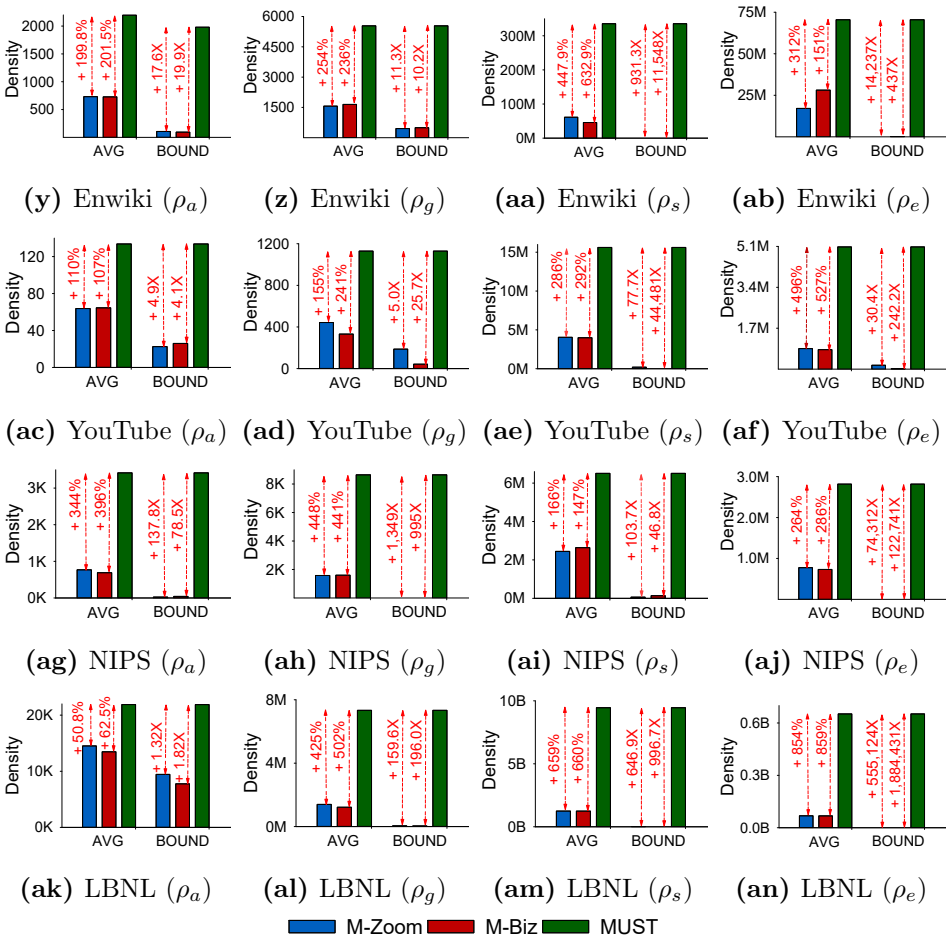
(m) Enron (ρ_a) **(n)** Enron (ρ_g) **(o)** Enron (ρ_s) **(p)** Enron (ρ_e)

(q) StackO (ρ_a) **(r)** StackO (ρ_g) **(s)** StackO (ρ_s) **(t)** StackO (ρ_e)

(u) Kowiki (ρ_a) **(v)** Kowiki (ρ_g) **(w)** Kowiki (ρ_s) **(x)** Kowiki (ρ_e)

K: thousand, M: million, B: billion. Best viewed in color and zoom mode.

Figure 7.3: Average and bound of density on datasets



(y) Enwiki (ρ_a) **(z)** Enwiki (ρ_g) **(aa)** Enwiki (ρ_s) **(ab)** Enwiki (ρ_e)

(ac) YouTube (ρ_a) **(ad)** YouTube (ρ_g) **(ae)** YouTube (ρ_s) **(af)** YouTube (ρ_e)

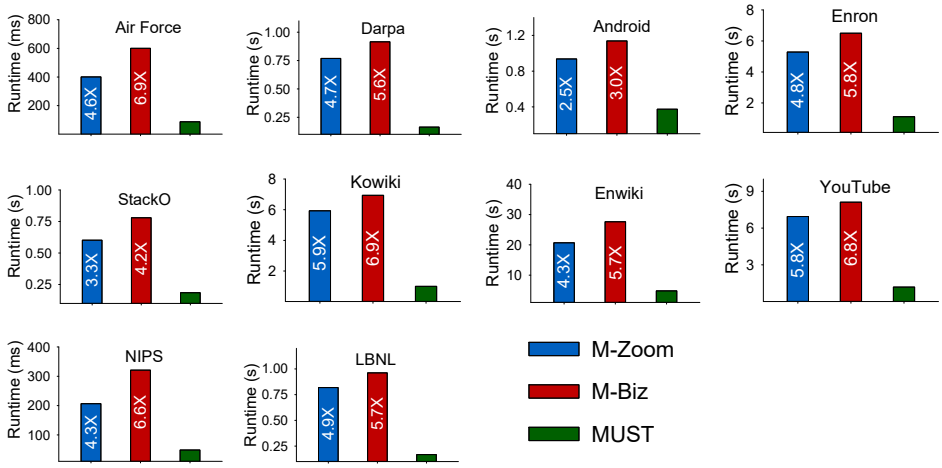
(ag) NIPS (ρ_a) **(ah)** NIPS (ρ_g) **(ai)** NIPS (ρ_s) **(aj)** NIPS (ρ_e)

(ak) LBNL (ρ_a) **(al)** LBNL (ρ_g) **(am)** LBNL (ρ_s) **(an)** LBNL (ρ_e)

K: thousand, M: million, B: billion. Best viewed in color and zoom mode.

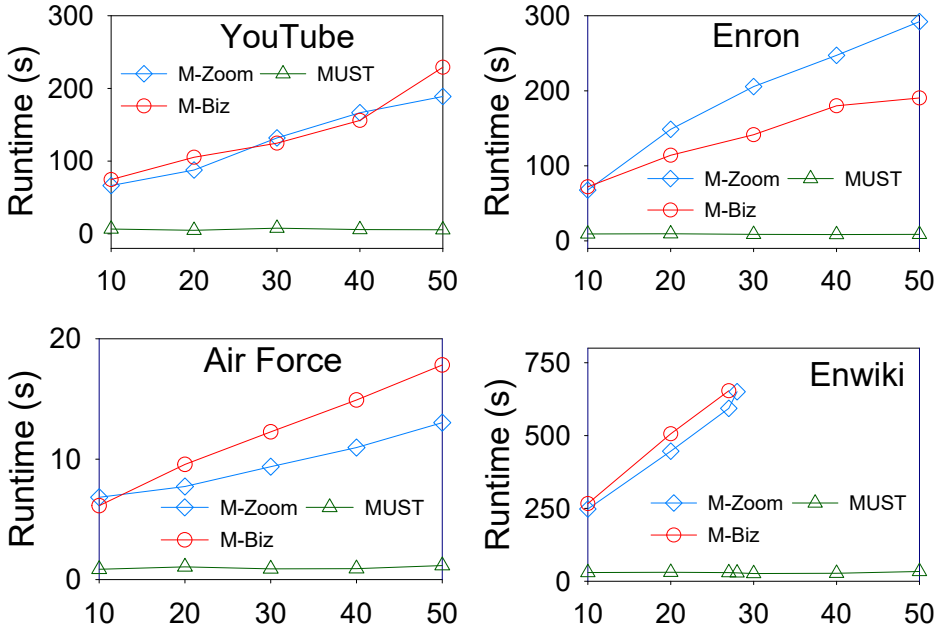
Figure 7.3: Average and bound of density on datasets (continued)

In particular, on the Air Force dataset, the average density with MUST is up to 546% higher than with M-Zoom and M-Biz, using the arithmetic average mass measure, and more than 891% higher on the Darpa dataset using entry surplus measure. In terms of lower bound of density of the estimated subtensors, there is a huge gap between the proposed algorithm and the baseline algorithms. For instance, on the Air Force dataset, the lower bound of density of the estimated subtensors with MUST are more than 360 times and two million times bigger than with both baseline algorithms, when applying arithmetic average mass and entry surplus measure, respec-



Best viewed in color.

Figure 7.4: Average runtime for a (sub)tensor on datasets.



Best viewed in color.

Figure 7.5: Runtime while varying k.

tively. More specifically, in the top three estimated subtensors by MUST, M-Zoom, and M-Biz in evaluation of network attack detection on the Air Force dataset (Section 7.7.5), the density of the second and the third subtensors found by the compared methods drops significantly and are much lower than in our proposed method. The densities of the second and the third estimated subtensors found by MUST are 7 times (∼1,930,307/263,295) times and 29 times (∼1,772,991/60,524) higher than the compared methods. The explanation for this result is that M-Zoom and M-Biz are not capable of providing a guarantee on the density of these estimated subtensors with respect to the original input data.

7.7.4 Diversity and Overlap Analysis

An important difference between MUST and other approaches is its ability to estimate multiple subtensors. Hence, important aspects worth evaluating and discussing are (1) how much difference it is between estimated subtensors, and (2) the fractions of overlap among the detected subtensors. Intuitively, MUST sequentially removes one slice which has a minimum slice weight at a time. Finally, the algorithm picks the top k highest densities among estimated subtensors.

In this subsection, we evaluate the diversity of the top three estimated subtensors by MUST, M-Zoom on the Enwiki, Kowiki, and Air Force datasets to analyze the overlap fractions of subtensors. We use arithmetic average mass (ρ_a) as the density metric and the used diversity measure is the same as in [146]. The diversity of two subtensors is the average dissimilarity between pairs of them. Here, we chose the Enwiki, Kowiki, and Air Force datasets because they contain anomaly and fraud events, and that they are commonly used for this type of benchmark [146, 149]. Table 7.4 shows the diversity (divers for short) of the top three estimated subtensors by MUST and M-Zoom. We observe that the obtained diversities by MUST are 36.2%, 37.2%, and 20.8% on Enwiki, Kowiki, and Air Force, respectively. The overlap between the subtensors are acceptable and considerable in many contexts, e.g., anomaly and fraud detection, because groups of fraudulent users might share some common smaller groups or some fraudsters. Another reason is that fraudulent behaviors of users might happen in just some specific periods of time. Compared to M-Zoom, M-Zoom can find more diverse subtensors, which can be explained as follows. M-Zoom is specifically designed to find different subtensors by creating a snapshot of the data at each detection process, and it mines a block in this intermediate tensor. The results of this is, however, that M-Zoom cannot provide guarantee on the density of the detected subtensors, except on the first sub-

Table 7.4: Diversity of estimated subtensors

	Dataset	#	Volume*	Density	Diversity(%)
MUST	Enwiki	1	4 (1 × 2 × 2)	2397.6	36.2%
		2	20 (1 × 4 × 5)	2375.7	
		3	9 (1 × 3 × 3)	2355.9	
	Kowiki	1	8 (2 × 2 × 2)	273.0	37.2%
		2	80 (4 × 4 × 5)	258.5	
		3	64 (4 × 4 × 4)	240.5	
	Air Force	1	2 (X_1 × 2 × 1 × 1 × 1)	1,980,948	20.8%
		2	1 (X_1 × 1 × 1 × 1 × 1)	1,930,307	
		3	8 (X_1 × 2 × 1 × 2 × 2)	1,772,991	
M-Zoom	Enwiki	1	4 (1 × 2 × 2)	2397.6	96.7%
		2	6 (1 × 2 × 3)	1961.5	
		3	18 (2 × 3 × 3)	908.25	
	Kowiki	1	8 (2 × 2 × 2)	273.0	99.4%
		2	12 (2 × 2 × 3)	246.0	
		3	29,520 (16 × 41 × 45)	181.6	
	Air Force	1	2 (X_1 × 2 × 1 × 1 × 1)	1,980,948	70.8%
		2	1 (X_1 × 1 × 1 × 1 × 1)	263,295	
		3	4,320 (X_2 × 5 × 4 × 3 × 3)	60,524	

* Where $X_1 = 1 \times 1 \times 1$, and $X_2 = 3 \times 4 \times 2$.

tensor. This is one of the drawbacks of M-Zoom, and as discussed below (Section 7.7.5), the effectiveness of M-Zoom on network attack detection greatly drops with multiple subtensors.

7.7.5 Effectiveness on Network Attack Detection

Extensive studies have shown that unexpected dense subregion (subgraph, subtensor) is a high sign of anomaly behaviors [82]. So, dense subregion

detection is one of the efficient approaches and is widely-used in fraudulent behavior detection. In this section, we evaluate the efficiency of dense subregion detection in Network Attack Detection by performing extensive experiment on Air Force dataset. Air Force is specifically suitable for evaluating network attack detection ability. As mentioned earlier, it is a dataset of TCP dump data of a typical U.S. Air Force LAN. It contains the ground truth labels of connections, including both intrusions (or attacks) connections, and normal connections. In detail, there are 972,781 connections as normal, while other connections are attacks. This dataset is widely used for the task of detecting anomaly and network attacks.

Here, we demonstrate the efficiency, the effectiveness of our proposed method on anomaly and network attack detection, and compare it with M-Zoom and M-Biz. We analyze the five highest subtensors returned by M-Zoom, M-Biz, and MUST on Air Force, and then we compute how many connections in the estimated subtensors are normal activities or attack[10]. Table 7.5 shows the connections in the top five subtensors detected by MUST, M-Zoom, and M-Biz using arithmetic average mass (ρ_a) as the density metric. We observe that all connections in the top five subtensors found by MUST are attack connections with no false positive. This is because MUST guarantees the density of all multiple subtensors it finds. With M-Zoom and M-Biz, they have the same result as MUST in the top two subtensors. However, in the three remaining subtensors, there are many normal connections that are wrong estimated by M-Zoom and M-Biz. For example, in the third subtensor estimated by M-Zoom, only 56,433 connections are attack, and 151,080 other connections are normal among 207,513 connections. So, the ratio of attack is only 27,2% with M-Zoom, and 26.03% with M-Biz. In other words, M-Zoom and M-Biz produce a high rate of false positive, which in turn means that MUST outperforms M-Zoom, M-Biz when used in the task of network attack detection, using the Air Force dataset.

[10]We provide the Matlab code to analyze attack connections in the code repository at `https://bitbucket.org/duonghuy/mtensor/src/master/data/`.

Table 7.5: Network attack detection on Air Force in the top five subtensors

	#	Volume	Density (ρ_a)	# Connections	# Attack Connections	# Normal Connections	# Ratio of Attack
MUST	1	$1 \times 1 \times 1 \times 2 \times 1 \times 1 \times 1$	1,980,948	2,263,941	2,263,941	0	100%
	2	$1 \times 1 \times 1 \times 1 \times 1 \times 1 \times 1$	1,930,307	1,930,307	1,930,307	0	100%
	3	$1 \times 1 \times 1 \times 2 \times 1 \times 2 \times 2$	1,772,991	2,532,845	2,532,845	0	100%
	4	$1 \times 1 \times 1 \times 2 \times 1 \times 1 \times 2$	1,764,860	2,269,106	2,269,106	0	100%
	5	$1 \times 1 \times 1 \times 2 \times 1 \times 2 \times 2$	1,612,741	2,534,308	2,534,308	0	100%
M-Zoom	1	$1 \times 1 \times 1 \times 2 \times 1 \times 1 \times 1$	1,980,948	2,263,941	2,263,941	0	100%
	2	$1 \times 1 \times 1 \times 1 \times 1 \times 1 \times 1$	263,295	263,295	263,295	0	100%
	3	$3 \times 4 \times 2 \times 5 \times 4 \times 3 \times 3$	60,524	207,513	56,433	151,080	27.2%
	4	$3 \times 3 \times 4 \times 4 \times 2 \times 154 \times 42$	33,901	1,026,723	1,007,762	18,961	98.15%
	5	$2 \times 4 \times 1 \times 7 \times 3 \times 13 \times 12$	16,467	98,806	42,961	55,845	43.5%
M-Biz	1	$1 \times 1 \times 1 \times 2 \times 1 \times 1 \times 1$	1,980,948	2,263,941	2,263,941	0	100%
	2	$1 \times 1 \times 1 \times 1 \times 1 \times 1 \times 1$	263,295	263,295	263,295	0	100%
	3	$3 \times 4 \times 2 \times 5 \times 5 \times 3 \times 3$	60,699	216,784	56,433	160,351	26.03%
	4	$2 \times 3 \times 4 \times 3 \times 1 \times 154 \times 42$	33,757	1,007,906	1,007,762	144	99.98%
	5	$2 \times 4 \times 1 \times 7 \times 3 \times 15 \times 12$	17,171	107,934	42,968	64,966	39.8%

7.7.6 Execution Time

In terms of execution time, to evaluate the performance of the algorithms, we recorded the runtime of the algorithms on real-world datasets using four measures of the density to return top ten density subtensors. Then, we calculated the average runtime of the algorithms per each estimated subtensor. The results from this experiment are shown in Figure 7.4. We observe that MUST is much faster than M-Zoom and M-Biz on all the datasets. Specifically, it is up to 6.9 times faster than M-Zoom and M-Biz to estimate a subtensor. The obtained results fit well with our hypothesis and or complexity discussion in Section 7.6. The explanation for this is that in MUST the algorithm needs only a single maintaining process to get dense subtensors, while in M-Zoom and M-Biz, they repeatedly call the search function k times to be able to get k dense subtensors. The proposed method, MUST, runs nearly in constant time independent of the increase of the number of subtensors; whereas the execution times of both M-Zoom and M-Biz increase (near)linearly with respect to value of k.

7.7.7 Scalability

We also evaluate the impact of the number of estimated subtensors (k) to the performance of the algorithms. Here, we performed experiments on the Enron, YouTube, Air Force, and Enwiki datasets. With arithmetic average mass, we measured the runtime while varying k within $\{10, 20, 30, 40, 50\}$. Figure 7.5 shows the results of this experiment. On Enwiki dataset, both M-Zoom and M-Biz run out of memory when the setting value of $k \geq 30$. As shown in the figure, the execution time of M-Zoom and M-Biz increase linearly with the increasing value of k, while the running time of MUST is constant with respect to the value of k. These results conform well with our complexity analysis in Section 7.6.

In conclusion, MUST outperforms the current state-of-the-art algorithms for solving the dense subtensor detection problem, from both a theoretical and experimental perspective.

7.8 Conclusion

In this chapter, we proposed a new technique to improve the task of dense subtensor, dense subgraph detection. As discussed, the contributions are both theoretical and practical. First, we developed concrete theoretical proofs for dense subtensors estimation in a tensor problem, as well as theoretical proofs for dense subgraph detection. An important purpose of this

was to provide a guarantee for a higher lower bound density of the estimation in both dense subtensor and subgraph detection. In addition, we developed a new theoretical foundation to guarantee a high density of multiple subtensors. Second, extending existing dense subtensor detection methods, we developed a new algorithm called MUST that is less complex and thus more efficient than existing methods. Our experimental experiments demonstrated that the proposed method significantly outperformed the current state-of-the-art algorithms for dense subtensor detection problem. It is significantly more efficient and effective than the baseline methods. In conclusion, the proposed method is not only theoretically sound, but is also applicable for detecting dense subgraph and dense subtensors.

Part V

Conclusions

Chapter 8

Conclusions and Future Work

This chapter briefly summarizes the contributions of the work in this thesis, and outlines possible topics for future research.

8.1 Summary

The design of efficient and effective algorithms to detect events in changing and evolving environments is a difficult problem. One particular challenge when working with various types of structured data in dynamic environments is that data are inherently dynamic, changing and evolving over time with unknown distributions, and the characteristics of these data are not available. Moreover, these data may be dependent on changing constraints. During the evolution of the data, there is a certain probability of correlation and transition between data points and data features.

In the context of high-dimensional streaming data, where data are arriving in a form of a stream with high volume and velocity, the design of an efficient solution with limitations on memory and computational infrastructure is also a challenge. The problem lies not only in the existence of complex correlations but also in the high level of computational complexity. Since the task of event detection is generally hard, existing solutions are often very complex and involve high resource consumption. Furthermore, most currently existing approaches lack a formal theoretical foundation that can guarantee the quality of the solution, since the results and efficiency are mostly based on heuristics and empirical observations.

Motivated by the challenges discussed above, this thesis has addressed these issues by proposing novel techniques and methods for detecting events in various types of data in evolving environments. The following studies were carried out as part of this work:

1. We have presented a method of dynamic memory allocation for avoiding bias when constructing the data characteristics for an unknown data distribution, e.g., for dense and sparse data.

2. We have introduced temporal dependencies and hypotheses for detecting changes in evolving data.

3. We have discussed and simulated the correlations and transitions of features in order to sketch a stream with concept drift.

4. We have generalized the problem of multiple dense subregion detection with a better quality guarantee of the solutions and expanded this to cover both tensor and graph data.

5. We have presented a well-founded theoretical solution to prove the efficiency and correctness of our solutions.

8.2 Main Contributions

The contributions of the thesis include novel techniques, approaches and improvements to current state-of-the-art methods of event detection. The main contributions of this thesis can be summarized as follows:

1. A summary utility-list structure was proposed to reduce memory consumption and speed up the join operation in pattern detection. This structure was integrated into a novel algorithm for the efficient discovery of high utility patterns. The proposed method efficiently stores and retrieves utility-lists by dynamically allocating memory, and reuses memory during the mining process. The memory used for the utility-list is organized as a stream and is dynamically estimated. A linear time method for constructing the utility-list segments in a utility-list buffer was also introduced. Experiments showed that algorithms employing the proposed structure were up to 10 times faster than when standard utility-lists were used, and the memory required was reduced by a factor of up to six. The proposed technique also performed well for both dense and sparse datasets.

2. The first study focused on detecting changes in high utility itemsets by considering both a utility measure and the recency of transactions in the data stream. We introduced an algorithm to detect changes, including both *local* and *global* drift, in the utility distributions of

itemsets in a stream of quantitative customer transactions. This algorithm took into account the evolving behavior of the streams by utilizing a fading function to quickly adapt to changes in a data stream. We proposed an approach that used statistical testing based on Hoeffding's inequality and a Bonferroni correction to report significant changes to the user. The proposed method was capable of discovering both increasing and decreasing trends. In addition, we proposed a new distance measure that generalized the cosine similarity by taking into account the distance between pairs of HUIs, in order to detect the changes in the structure of HUIs. An extensive experiment was conducted to evaluate the proposed method, which demonstrated the feasibility, effectiveness, and efficiency of our algorithm.

3. We proposed a new and efficient method of detecting changes in streaming data by exploring the temporal dependencies of the data in the stream. A new statistical model was introduced to compute the probabilities of finding change points in the stream using the temporal dependency information between different observed data points in a stream. The computed probabilities were used to generate a distribution, which was used in statistical hypothesis tests to determine the candidate changes. We also used the proposed model to develop a new algorithm to detect change points in linear time, making it applicable in real-time applications.

    - We introduced a new model of the high-order temporal dependencies and data distributions in streaming data.
    - We developed a method that was able to handle data arriving in a high-velocity stream by using continuously updated estimation factors as part of our model.
    - We proposed an efficient real-time algorithm based on pivotal statistic tests for change detection.
    - In order to demonstrate the feasibility, efficiency, and generality of our method, we conducted a thorough evaluation using several real-life datasets and a comparison with related approaches.
    - Our method showed the best performance of the state-of-the-art methods compared in the experiment. In addition, our method had a linear time performance, meaning that it can be deployed online in real-world stream applications.

4. We performed an initial study of the correlation and transition between groups of data features to address the main drawback of most

current approaches, which assume that the forgetting factor is constant at every point of observation and over the whole process, a constraint that is too restrictive for real-world applications. We proposed a novel robust method for sketching streaming histograms with limited computing resources, based on an ensemble randomization method. An algorithm was developed that used an evolving model with adaptive coefficients to obtain the elements of the histogram. This algorithm considered the timestamps of different observations in each coefficient and solved an optimization problem to evolve the values of the coefficients to optimal values.

- We proposed an evolving model with adaptive, auto-tuned coefficients to investigate the correlations between data, and simulated the transitions between features and groups of features in dynamically changing data.
- We developed a novel algorithm that could sketch the histograms for a data stream using multiple weighted factors, while taking into account the time-sensitive changes in the stream using different adaptive weighted factors for different groups of data, at each observation point.
- We carefully evaluated our approach through extensive experiments on both synthetic and real-life datasets. Our proposed method quickly adapted to concept drift. Our algorithm was able to achieve an overall classification accuracy up to 3.99% higher than the baseline algorithms, and the error rate was 12 times better than state-of-the-art methods with concept drift.

5. We proposed novel concrete proofs for problems involving the detection of dense subtensors from tensor data, and dense subgraphs in a graph. We introduced a better theoretical density guarantee for both dense subtensor and dense subgraph detection using approximation algorithms. The new boundary was higher; it did not depend solely on the dimension of the data space, but was also constrained to the size of the densest subtensor/subgraph. The contributions of this stage were threefold, (*i*) *novel proofs*: we proposed novel proofs to guarantee a better density of both dense subgraphs and dense subtensors; (*ii*) *new better guarantee in subgraph*: we provided a new theoretical higher density guarantee for dense subgraphs with a new bound, $\frac{1}{2}(1 + \frac{1}{min(m,\sqrt{n})})$; (*iii*) *new better guarantee in subtensor*: we proved a better density guarantee for dense subtensors with a new higher bound, $\frac{1}{N}(1 + \frac{N-1}{min(m,\sqrt{n})})$.

6. We proposed a new technique that improved the task of dense subtensor detection. The contributions of this stage were both theoretical and practical solutions. We developed concrete theoretical proofs for the estimation of dense subtensors in a tensor problem, and provided a guarantee for a higher lower bound density of the estimated subtensors. In addition, we developed a new theoretical foundation to guarantee a high density for multiple subtensors. A new algorithm was proposed that was not only less complex, but also guaranteed the correctness of its effectiveness. This method can be used to detect dense subtensors, and thus is more efficient than existing methods.

    - We presented a novel theoretical foundation, along with proofs showing that it is possible to maintain multiple subtensors with a high density guarantee.
    - We provided a new method that was capable of estimating subtensors with a density guarantee that was higher than those provided by existing methods.
    - We proved that there exist at least $min(1+\frac{n}{2N}, 1+N(N-1))$ subtensors in the tensor with a density greater than a lower bound.
    - We performed an extensive experimental evaluation using real-world datasets to demonstrate the efficiency of our solution. The proposed method was up to 6.9 times faster than state-of-the-art methods, and the resulting subtensors had a density up to two million times higher.

8.3 Future Work

The work presented in this thesis mainly focuses on addressing a set of research questions related to event detection topic in various types of data where the characteristics of data evolve over time. An extensive analysis and evaluation were performed to demonstrate the efficiency of our proposed methods in terms of overcoming the limitations of event detection algorithms for dynamic, evolving data. In addition to providing a new approach to solving the problem, our new insights into the limitations, advantages and disadvantages of these methods opens up many interesting avenues for future work.

Sketching with Data Dependencies: In this work, we investigated ways of exploiting the temporal dependencies of data to detect changes in a data stream. The method proposed in this thesis was inspired by the ideas behind linear Markov process models, in which the main principle

is to estimate the probability of a transition between states. The number of states k is fixed, and k sequential data points are used to construct the dependencies in the whole process at each observation point. This limitation suggests several directions for extending this work. Firstly, it would be valuable to investigate how the number of different CCPs at different data points affects the dependency model. At each observation, the temporal dependency may change such that data items involved in the evolvement are changing. The number of evolving data can be adaptive at different observation. Secondly, the involved data for temporal dependence are not necessarily sequential, and may be discretely distributed in the stream. This observation suggests another interesting area for future work. Finally, in the data stream, a large volume of data arrives at a high speed, meaning that it is infeasible to store information on all of the data. The development of sketching algorithms combining with data dependencies to detect drift and outliers is another possible area for further study.

Transition Between Features: Achieving efficient processing of data streams with limited computing resources is a challenging but crucial task. The study presented in Chapter 6 is our first study of correlation and transition between features in an evolving model that adapts quickly to concept drift. Our proposed method was motivated by the observation that existing techniques are based on the restrictive assumption that the forgetting factor is constant at every point of observation and over the whole process. Despite the efficiency of our method, several challenges arose during this work, and these gave rise to several directions for future research. Firstly, the time complexity at evolving steps is high for solving the optimization problem, and the performance strongly depends on the solver used and its convergence speed. Secondly, the value of the hyper-parameters are dependent on both the application and the data. Heuristics methods are typically used to select these parameters. The optimal choice of parameters for a problem is quite relatively challenging, especially for very sparse, small, or high dimensional datasets. Moreover, when performing our extensive evaluation, we were also aware of a tradeoff between performing the evolving at every observation point with evolving after a certain period (window). Furthermore, each data point (POI) may have different characteristics, and this leads to the need for an evolving model for each POI or group of POIs (e.g., different time unit). Adapting the window size at each step of the evolution and time unit factor for each group of data, together with considering above discussions, are areas that it would be valuable to study further in future research.

Improving the Quality of a Slice Cut: When developing the method

for detecting dense subtensors, we observed that existing approaches (including ours) treat each tensor slice independently, and that they do not consider the relationships between the slices of a tensor. However, there may be connections among these slices when we perform a projection onto a way of the tensor, in turn, it highly impacts to the density of the remaining part. As a result, the density obtained for the subtensor may be low. We therefore need to consider a slice to remove in both its own weight and connections between it and other slices in the tensor when projecting on a way of the tensor. On the other hand, the density of a subtensor depends also on a suitable chosen measure. Providing higher density guarantee and proposing an effective measure could be a potential direction for future work. In addition, we intend to explore the application of our method to graph data of many different types, such as bipartite, plane, directed and undirected graphs, and to use it to solve real-world event detection problems such as change and anomaly detection. The issue of dense subtensor detection still remains, and is even more challenging for the dense subgraph detection problems with a more complex structure. This gives rise to further possible directions for future work in this area.

References

[1] Strategic Research Area, BigData, NTNU. (https://www.ntnu.edu/ie/bigdata).

[2] The MUSED Project, NTNU. (https://www.ntnu.edu/idi/mused).

[3] Iris Adä and Michael R. Berthold. EVE: a Framework for Event Detection. *Evolving Systems*, 4(1):61–70, 2013.

[4] U. Adhikari, T. H. Morris, and S. Pan. Applying Hoeffding Adaptive Trees for Real-Time Cyber-Power Event and Intrusion Classification. *IEEE Transactions on Smart Grid*, 9(5):4049–4060, 2018.

[5] R. Agrawal and R. Srikant. Quest Synthetic Data Generator. Available at. (<http://www.almaden.ibm.com/cs/quest/syndata.html>), 1994.

[6] Rakesh Agrawal and Ramakrishnan Srikant. Fast Algorithms for Mining Association Rules. *VLDB*, pages 487–499, 1994.

[7] C. Ahmed, S. Tanbeer, B. S. Jeong, and Y. K. Lee. Efficient Tree Structures for High Utility Pattern Mining in Incremental Databases. *IEEE Transactions on Knowledge and Data Engineering*, 21(12):1708–1721, 2009.

[8] Christoforos Anagnostopoulos, Dimitris K. Tasoulis, Niall M. Adams, Nicos G. Pavlidis, and David J. Hand. Online Linear and Quadratic Discriminant Analysis with Adaptive Forgetting for Streaming Classification. *Statistical Analysis and Data Mining*, 5(2):139–166, 2012.

[9] Reid Andersen. A Local Algorithm for Finding Dense Subgraphs. *ACM Transactions on Algorithms*, 6(4):60:1–60:12, 2010.

[10] Reid Andersen and Kumar Chellapilla. Finding Dense Subgraphs with Size Bounds. In *Proceedings of Algorithms and Models for the Web-Graph*, pages 25–37, 2009.

[11] Yuichi Asahiro, Refael Hassin, and Kazuo Iwama. Complexity of Finding Dense Subgraphs. *Discrete Applied Mathematics*, 121(1):15–26, 2002.

[12] Yuichi Asahiro, Kazuo Iwama, Hisao Tamaki, and Takeshi Tokuyama. Greedily Finding a Dense Subgraph. *Journal of Algorithms*, 34(2):203–221, 2000.

[13] Manuel Baena-García, José del Campo-Ávila, Raúl Fidalgo, Albert Bifet, Ricard Gavaldá, and Rafael Morales-Bueno. Early Drift Detection Method. In *The 4th International Workshop on Knowledge Discovery from Data Streams*, 2006.

[14] Oana Denisa Balalau, Francesco Bonchi, T-H. Hubert Chan, Francesco Gullo, and Mauro Sozio. Finding Subgraphs with Maximum Total Density and Limited Overlap. In *Proceedings of the 8th ACM International Conference on Web Search and Data Mining, WSDM*, pages 379–388, 2015.

[15] Antonio Balzanella and Rosanna Verde. Summarizing and Detecting Structural Drifts from Multiple Data Streams. In *Proceedings of Classification and Data Mining*, pages 105–112, 2013.

[16] Roberto S.M. Barros, Danilo R.L. Cabral, Paulo M. Gonçalves, and Silas G.T.C. Santos. RDDM: Reactive Drift Detection Method. *Expert Systems with Applications*, 90(Supplement C):344–355, 2017.

[17] Albert Bifet. Classifier Concept Drift Detection and the Illusion of Progress. In *Artificial Intelligence and Soft Computing*, pages 715–725, Cham, 2017. Springer International Publishing.

[18] Albert Bifet and Ricard Gavaldà. Learning from Time-Changing Data with Adaptive Windowing. In *Proceedings of the 2007 SIAM International Conference on Data Mining, SDM*, pages 443–448, 2007.

[19] Albert Bifet and Ricard Gavaldà. Adaptive Learning from Evolving Data Streams. In *Proceedings of the 8th International Symposium on Intelligent Data Analysis*, pages 249–260, 2009.

[20] Albert Bifet, Geoff Holmes, Richard Kirkby, and Bernhard Pfahringer. MOA: Massive Online Analysis. *The Journal of Machine Learning Research*, 11:1601–1604, 2010.

[21] Albert Bifet, Jesse Read, Indrė Žliobaitė, Bernhard Pfahringer, and Geoff Holmes. Pitfalls in Benchmarking Data Stream Classification and How to Avoid Them. In *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases, ECML PKDD*, pages 465–479, 2013.

[22] Dean A. Bodenham and Niall M. Adams. Continuous Monitoring for Changepoints in Data Streams using Adaptive Estimation. *Statistics and Computing*, 27(5):1257–1270, 2017.

[23] Abdelhamid Bouchachia. Fuzzy Classification in Dynamic Environments. *Soft Computing*, 15(5):1009–1022, 2011.

[24] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122, 2011.

[25] A. Broder. On the Resemblance and Containment of Documents. In *Proceedings of the 1997 Compression and Complexity of Sequences, SEQUENCES*, pages 21–29, 1997.

[26] W. Bryc. A Uniform Approximation to the Right Normal Tail Integral. *Applied Mathematics and Computation*, 127(2):365–374, 2002.

[27] Moses Charikar. Greedy Approximation Algorithms for Finding Dense Components in a Graph. In *Approximation Algorithms for Combinatorial Optimization*, pages 84–95, 2000.

[28] Swarup Chattopadhyay, C.A. Murthy, and Sankar K. Pal. Fitting Truncated Geometric Distributions in Large Scale Real World Networks. *Theoretical Computer Science*, 551:22–38, 2014.

[29] James Cheng, Yiping Ke, and Wilfred Ng. A Survey on Algorithms for Mining Frequent Itemsets over Data Streams. *Knowledge and Information Systems*, 16(1):1–27, 2008.

[30] Lianhua Chi and Xingquan Zhu. Hashing Techniques: A Survey and Taxonomy. *ACM Computing Surveys*, 50:11:1–11:36, 2017.

[31] P. Comon. Tensors : A Brief Introduction. *IEEE Signal Processing Magazine*, 31(3):44–53, 2014.

[32] Laurent Condat. Fast Projection onto the Simplex and the ℓ_1 Ball. *Mathematical Programming*, 158(1):575–585, 2016.

[33] Graham Cormode and S. Muthukrishnan. An Improved Data Stream Summary: The Count-Min Sketch and its Applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[34] Graham Cormode and S. Muthukrishnan. Summarizing and Mining Skewed Data Streams. In *Proceedings of the 2005 SIAM International Conference on Data Mining, SDM*, pages 44–55, 2005.

[35] Thu-Lan Dam, Sean Chester, Kjetil Nørvåg, and Quang-Huy Duong. Efficient Top-k Recently-Frequent Term Querying over Spatio-Temporal Textual Streams. *Information Systems*, 97:101687, 2021.

[36] Thu-Lan Dam, Kenli Li, Philippe Fournier-Viger, and Quang-Huy Duong. An Efficient Algorithm for Mining top-rank-k Frequent Patterns. *Applied Intelligence*, 45(1):96–111, 2016.

[37] Thu-Lan Dam, Kenli Li, Philippe Fournier-Viger, and Quang-Huy Duong. An Efficient Algorithm for Mining top-k On-shelf High Utility Itemsets. *Knowledge and Information Systems*, 52(3):621–655, 2017.

[38] Thu-Lan Dam, Kenli Li, Philippe Fournier-Viger, and Quang-Huy Duong. CLS-Miner: Efficient and Effective Closed High Utility Itemset Mining. *Frontiers of Computer Science*, 13:357—381, 2017.

[39] Thu-Lan Dam, Heri Ramampiaro, Kjetil Nørvåg, and Quang-Huy Duong. Towards Efficiently Mining Closed High Utility Itemsets from Incremental Databases. *Knowledge-Based Systems*, 165:13–29, 2019.

[40] Siddharth Dawar, Veronica Sharma, and Vikram Goyal. Mining top-k High-utility Itemsets from a Data Stream under Sliding Window Model. *Applied Intelligence*, 47(4):1240–1255, 2017.

[41] Janez Demšar. Statistical Comparisons of Classifiers over Multiple Data Sets. *The Journal of Machine Learning Research*, 7:1–30, 2006.

[42] G. Ditzler and R. Polikar. Incremental Learning of Concept Drift from Streaming Imbalanced Data. *IEEE Transactions on Knowledge and Data Engineering*, 25(10):2283–2301, 2013.

[43] John Duchi, Shai Shalev-Shwartz, Yoram Singer, and Tushar Chandra. Efficient Projections Onto the ℓ_1-ball for Learning in High Dimensions. In *Proceedings of the 25th International Conference on Machine Learning, ICML*, pages 272–279, 2008.

[44] Quang-Huy Duong, Philippe Fournier-Viger, Heri Ramampiaro, Kjetil Nørvåg, and Thu-Lan Dam. Efficient High Utility Itemset Mining using Buffered Utility-lists. *Applied Intelligence*, 48(7):1859–1877, 2018.

[45] Quang-Huy Duong, Bo Liao, Philippe Fournier-Viger, and Thu-Lan Dam. An Efficient Algorithm for Mining the top-k High Utility Itemsets, Using Novel Threshold Raising and Pruning Strategies. *Knowledge-Based Systems*, 104:106–122, 2016.

[46] Quang-Huy Duong, Heri Ramampiaro, and Kjetil Nørvåg. Applying Temporal Dependence to Detect Changes in Streaming Data. *Applied Intelligence*, 48(12):4805–4823, 2018.

[47] Quang-Huy Duong, Heri Ramampiaro, and Kjetil Nørvåg. Sketching Streaming Histogram Elements using Multiple Weighted Factors. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management, CIKM*, pages 19–28, 2019.

[48] Quang-Huy Duong, Heri Ramampiaro, and Kjetil Nørvåg. Multiple Dense Subtensor Estimation with High Density Guarantee. In *Proceedings of the 36th IEEE International Conference on Data Engineering, ICDE*, pages 637–648, 2020.

[49] Quang-Huy Duong, Heri Ramampiaro, Kjetil Nørvåg, and Thu-Lan Dam. Density Guarantee on Finding Multiple Subgraphs and SubTensors. *ACM Transactions on Knowledge Discovery from Data*, pages 1–32, 2021.

[50] Quang-Huy Duong, Heri Ramampiaro, Kjetil Nørvåg, Philippe Fournier-Viger, and Thu-Lan Dam. High Utility Drift Detection in Quantitative Data Streams. *Knowledge-Based Systems*, 157:34–51, 2018.

[51] P. Erdös and A. Rényi. On Random Graphs I. *Publicationes Mathematicae Debrecen*, 6:290, 1959.

[52] U. Feige, D. Peleg, and G. Kortsarz. The Dense k-Subgraph Problem. *Algorithmica*, 29(3):410–421, 2001.

[53] Philippe Fournier-Viger, Antonio Gomariz, Ted Gueniche, Azadeh Soltani, Cheng-Wei Wu, and Vincent S. Tseng. SPMF: A Java Open-Source Pattern Mining Library. *Journal of Machine Learning Research*, 15:3569–3573, 2014.

[54] Philippe Fournier-Viger, Chun-Wei Lin, Quang-Huy Duong, Thu-Lan Dam, Lukas Sevcik, Dominik Uhrin, and Miroslav Voznak. PFPM: Discovering Periodic Frequent Patterns with Novel Periodicity Measures. In *Proceedings of the 2nd Czech-China Scientific Conference 2016*. 2017.

[55] Philippe Fournier-Viger, Jerry Chun-Wei Lin, Quang-Huy Duong, and Thu-Lan Dam. FHM+: Faster High-Utility Itemset Mining Using Length Upper-Bound Reduction. In *Trends in Applied Knowledge-Based Systems and Data Science*, pages 115–127, 2016.

[56] Philippe Fournier-Viger, Jerry Chun-Wei Lin, Quang-Huy Duong, and Thu-Lan Dam. PHM: Mining Periodic High-Utility Itemsets. In *Proceedings of the 16th Industrial Conference in Data Mining*, pages 64–79, 2016.

[57] Philippe Fournier-Viger, Cheng-Wei Wu, Souleymane Zida, and VincentS. Tseng. FHM: Faster High-Utility Itemset Mining Using Estimated Utility Co-occurrence Pruning. In *Foundations of Intelligent Systems*, volume 8502 of *Lecture Notes in Computer Science*, pages 83–92. Springer International Publishing, 2014.

[58] Philippe Fournier-Viger, Peng Yang, Jerry Chun-Wei Lin, Quang-Huy Duong, Thu-Lan Dam, Jaroslav Frnda, Lukas Sevcik, and Miroslav Voznak. Discovering Periodic Itemsets Using Novel Periodicity Measures. *Advances in Electrical and Electronic Engineering*, 17(1):33–44, 2019.

[59] Philippe Fournier-Viger and Souleymane Zida. FOSHU: Faster On-shelf High Utility Itemset Mining – with or Without Negative Unit Profit. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing, SAC*, pages 857–864, 2015.

[60] Eugene Fratkin, Brian T. Naughton, Douglas L. Brutlag, and Serafim Batzoglou. MotifCut: Regulatory Motifs Finding with Maximum Density Subgraphs. *Bioinformatics*, 22(14):e150–e157, 2006.

[61] Ivani Inocencio Frías-Blanco, José del Campo-Ávila, Gonzalo Ramos-Jiménez, Rafael Morales-Bueno, Agustín Alejandro Ortiz-Díaz, and Yailé Caballero-Mota. Online and Non-Parametric Drift Detection Methods Based on Hoeffding's Bounds. *IEEE Transactions on Knowledge and Data Engineering*, 27(3):810–823, 2015.

[62] Esther Galbrun, Aristides Gionis, and Nikolaj Tatti. Top-k Overlapping Densest Subgraphs. *Data Mining and Knowledge Discovery*, 30(5):1134–1165, 2016.

[63] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. A Survey on Concept Drift Adaptation. *ACM Computing Surveys*, 46(4):1–37, 2014.

[64] João Gama, Pedro Medas, Gladys Castillo, and Pedro Pereira Rodrigues. Learning with Drift Detection. In *Proceedings of Brazilian Symposium on Artificial Intelligence*, pages 286–295, 2004.

[65] João Gama, Raquel Sebastião, and Pedro Pereira Rodrigues. On Evaluating Stream Learning Algorithms. *Machine Learning*, 90(3):317–346, 2013.

[66] João Gama, Indre Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. A Survey on Concept Drift Adaptation. *ACM Computing Surveys*, 46(4):44:1–44:37, 2014.

[67] C.F. Gauss. *Theoria Combinationis Observationum Erroribus Minimis Obnoxiae*. Number v. 1 in Commentationes Societatis Regiae Scientiarum Gottingensis recentiores: Classis Mathemat. H. Dieterich, 1823.

[68] David Gibson, Ravi Kumar, and Andrew Tomkins. Discovering Large Dense Subgraphs in Massive Graphs. In *Proceedings of the 31st International Conference on Very Large Data Bases, VLDB*, pages 721–732, 2005.

[69] A. Globerson, G. Chechik, F. Pereira, and N. Tishby. Euclidean Embedding of Co-occurrence Data. *The Journal of Machine Learning Research*, 8:2265–2295, 2007.

[70] A. V. Goldberg. Finding a Maximum Density Subgraph. Technical report, 1984.

[71] Amit Goyal, Hal Daumé III, and Graham Cormode. Sketch Algorithms for Estimating Point Queries in NLP. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, EMNLP-CoNLL*, pages 1093–1103, 2012.

[72] Sudipto Guha, Kyuseok Shim, and Jungchul Woo. REHIST: Relative Error Histogram Construction Algorithms. In *Proceedings of the 30th International Conference on Very Large Data Bases, VLDB*, pages 300–311, 2004.

[73] F.A. Haight. *Handbook of the Poisson Distribution.* Publications in Operations Research. Wiley, 1967.

[74] R. W. Hamming. Error Detecting and Error Correcting Codes. *The Bell System Technical Journal*, 29(2):147–160, 1950.

[75] J. W. Han, J. Pei, and Y. W. Yin. Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach. *Data Mining and Knowledge Discovery*, 8(1):53–87, 2004.

[76] Jiawei Han, Micheline Kamber, and Jian Pei. Data Mining: Concepts and Techniques. In *Data Mining (Third Edition)*, The Morgan Kaufmann Series in Data Management Systems, pages 1–38. 2012.

[77] Michael Bonnell Harries, Claude Sammut, and Kim Horn. Extracting Hidden Context. *Machine Learning*, 32(2):101–126, 1998.

[78] Ruining He and Julian McAuley. Ups and Downs: Modeling the Visual Evolution of Fashion Trends with One-Class Collaborative Filtering. In *Proceedings of the 25th International Conference on World Wide Web, WWW*, pages 507–517, 2016.

[79] A. Heidarian and M. J. Dinneen. A Hybrid Geometric Approach for Measuring Similarity Level Among Documents and Document Clustering. In *2016 IEEE Second International Conference on Big Data Computing Service and Applications (BigDataService)*, pages 142–151, 2016.

[80] Wassily Hoeffding. Probability Inequalities for Sums of Bounded Random Variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

[81] Daniel N. Holtmann-Rice, Benjamin S. Kunsberg, and Steven W. Zucker. Tensors, Differential Geometry and Statistical Shading Analysis. *Journal of Mathematical Imaging and Vision*, 60:968–992, 2018.

[82] Bryan Hooi, Hyun Ah Song, Alex Beutel, Neil Shah, Kijung Shin, and Christos Faloutsos. FRAUDAR: Bounding Graph Fraud in the Face

of Camouflage. In *Proceedings of the 22nd ACM International Conference on Knowledge Discovery and Data Mining, SIGKDD*, pages 895–904, 2016.

[83] Piotr Indyk and Rajeev Motwani. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In *Proceedings of the 13th Annual ACM Symposium on Theory of Computing, STOC*, pages 604–613, 1998.

[84] Sergey Ioffe. Improved Consistent Sampling, Weighted Minhash and L1 Sketching. In *Proceedings of the 2010 IEEE International Conference on Data Mining, ICDM*, pages 246–255, 2010.

[85] M. Jiang, A. Beutel, P. Cui, B. Hooi, S. Yang, and C. Faloutsos. A General Suspiciousness Metric for Dense Blocks in Multimodal Data. In *Proceedings of the 2015 IEEE International Conference on Data Mining, ICDM*, pages 781–786, 2015.

[86] Ravi Kannan and V Vinay. *Analyzing the Structure of Large Graphs*. Bonn: Rheinische Friedrich-Wilhelms-Universität Bonn, 1999.

[87] Samir Khuller and Barna Saha. On Finding Dense Subgraphs. In *Proceedings of the 36th International Colloquium on Automata, Languages and Programming: Part I, ICALP*, pages 597–608, 2009.

[88] Daniel Kifer, Shai Ben-David, and Johannes Gehrke. Detecting Change in Data Streams. In *Proceedings of the 13th International Conference on Very Large Data Bases - Volume 30, VLDB*, pages 180–191, 2004.

[89] Ralf Klinkenberg. Learning Drifting Concepts: Example Selection vs. Example Weighting. *Intelligent Data Analysis*, 8(3):281–300, 2004.

[90] Y. S. Koh. CD-TDS: Change Detection in Transactional Data Streams for Frequent Pattern Mining. In *Proceedings of the 2016 International Joint Conference on Neural Networks, IJCNN*, pages 1554–1561, 2016.

[91] Tamara G. Kolda and Brett W. Bader. Tensor Decompositions and Applications. *SIAM Review*, 51(3):455–500, 2009.

[92] J. Zico Kolter and Marcus A. Maloof. Dynamic Weighted Majority: An Ensemble Method for Drifting Concepts. *The Journal of Machine Learning Research*, 8:2755–2790, 2007.

[93] Srikumar Krishnamoorthy. Pruning Strategies for Mining High Utility Itemsets. *Expert Systems with Applications*, 42(5):2371–2381, 2015.

[94] S. Kullback and R. A. Leibler. On Information and Sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, 1951.

[95] Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, and Andrew Tomkins. Trawling the Web for Emerging Cyber-communities. In *Proceedings of the Eighth International Conference on World Wide Web, WWW*, pages 1481–1493, 1999.

[96] Ravi Kumar, Maithra Raghu, Tamás Sarlós, and Andrew Tomkins. Linear Additive Markov Processes. In *Proceedings of the 26th International Conference on World Wide Web, WWW*, pages 411–419, 2017.

[97] Jérôme Kunegis. KONECT: The Koblenz Network Collection. In *Proceedings of the 22nd International Conference on World Wide Web, WWW*, pages 1343–1350, 2013.

[98] Guo-Cheng Lan, Tzung-Pei Hong, and Vincent S. Tseng. An Efficient Projection-Based Indexing Approach for Mining High Utility Itemsets. *Knowledge and Information Systems*, 38(1):85–107, 2014.

[99] Theodoros Lappas, Kun Liu, and Evimaria Terzi. Finding a Team of Experts in Social Networks. In *Proceedings of the 15th International Conference on Knowledge Discovery and Data Mining, SIGKDD*, pages 467–476, 2009.

[100] S. Lee and J. S. Park. Top-k High Utility Itemset Mining Based on Utility-list Structures. In *International Conference on Big Data and Smart Computing, BigComp*, pages 101–108, 2016.

[101] V. I. Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, 10:707, 1966.

[102] H. F. Li, H. Y. Huang, Y. C. Chen, Y. J. Liu, and S. Y. Lee. Fast and Memory Efficient Mining of High Utility Itemsets in Data Streams. In *Proceedings of the 8th IEEE International Conference on Data Mining, ICDM*, pages 881–886, 2008.

[103] Hua-Fu Li, Hsin-Yun Huang, and Suh-Yin Lee. Fast and Memory Efficient Mining of High-utility Itemsets from Data Streams: With and Without Negative Item Profits. *Knowledge and Information Systems*, 28(3):495–522, 2011.

[104] Peipei Li, Xindong Wu, and Xuegang Hu. Mining Recurring Concept Drifts with Limited Labeled Streaming Data. In *Proceedings of the 2nd Asian Conference on Machine Learning, PMLR*, volume 13, pages 241–252, 2010.

[105] Ping Li. 0-Bit Consistent Weighted Sampling. In *Proceedings of the 21st ACM International Conference on Knowledge Discovery and Data Mining, SIGKDD*, pages 665–674, 2015.

[106] Ping Li, Art B Owen, and Cun-Hui Zhang. One Permutation Hashing. In *Proceedings of the 25th International Conference on Neural Information Processing Systems (NIPS)*, pages 3113–3121, 2012.

[107] Xinsheng Li, Kasim Selçuk Candan, and Maria Luisa Sapino. M2TD: Multi-Task Tensor Decomposition for Sparse Ensemble Simulations. In *Proceedings of the 34th International Conference on Data Engineering, ICDE*, pages 1144–1155, 2018.

[108] Richard Lippmann, Joshua W. Haines, David J. Fried, Jonathan Korba, and Kumar Das. The 1999 DARPA Off-line Intrusion Detection Evaluation. *Computer Networks*, 34(4):579–595, 2000.

[109] Jun Liu and Jieping Ye. Efficient Euclidean Projections in Linear Time. In *Proceedings of the 26th Annual International Conference on Machine Learning, ICML*, pages 657–664, 2009.

[110] Mengchi Liu and Junfeng Qu. Mining High Utility Itemsets Without Candidate Generation. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management, CIKM*, pages 55–64, 2012.

[111] X. Liu, S. Ji, W. Glänzel, and B. De Moor. Multiview Partitioning via Tensor Methods. *IEEE Transactions on Knowledge and Data Engineering*, 25(5):1056–1069, 2013.

[112] Ying Liu, Wei-keng Liao, and Alok Choudhary. A Two-Phase Algorithm for Fast Discovery of High Utility Itemsets. In *Advances in Knowledge Discovery and Data Mining, 9th Pacific-Asia Conference, PAKDD*, pages 689–695. 2005.

[113] Mark Manasse, Frank McSherry, and Kunal Talwar. Consistent Weighted Sampling. *Microsoft Technical Report*, June 2010.

[114] Nishad Manerikar and Themis Palpanas. Frequent Items in Streaming Data: An Experimental Evaluation of the State-of-the-art. *Data & Knowledge Engineering*, 68(4):415–430, 2009.

[115] A. Markov. Extension of the Limit Theorems of Probability Theory to a Sum of Variables Connected in a Chain. In *Appendix B, Dynamic Probabilistic Systems (Volume I: Markov Models)*, pages 552–577. 1971.

[116] Diego Marron, Jesse Read, Albert Bifet, Talel Abdessalem, Eduard Ayguade, and José Herrero. Echo State Hoeffding Tree Learning. In *Proceedings of the 8th Asian Conference on Machine Learning, PMLR*, volume 63, pages 382–397, 2016.

[117] Koji Maruhashi, Fan Guo, and Christos Faloutsos. MultiAspectForensics: Pattern Mining on Large-Scale Heterogeneous Networks with Tensor Analysis. In *Proceedings of International Conference on Advances in Social Networks Analysis and Mining, ASONAM*, pages 203–210, 2011.

[118] Alan Mislove, Massimiliano Marcon, Krishna P. Gummadi, Peter Druschel, and Bobby Bhattacharjee. Measurement and Analysis of Online Social Networks. In *Proceedings of the 7th ACM Conference on Internet Measurement, SIGCOMM*, pages 29–42, 2007.

[119] Thomas M. Mitchell. *Machine Learning.* McGraw-Hill, Inc., New York, USA, 1997.

[120] Muhammad Anis Uddin Nasir, Aristides Gionis, Gianmarco De Francisci Morales, and Sarunas Girdzijauskas. Fully Dynamic Algorithm for Top-k Densest Subgraphs. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, CIKM*, pages 1817–1826, 2017.

[121] Willie Ng and Manoranjan Dash. A Test Paradigm for Detecting Changes in Transactional Data Streams. In *Proceedings of the 13th International Conference on Database Systems for Advanced Applications, DASFAA*, pages 204–219, 2008.

[122] D. Nion and N. D. Sidiropoulos. Tensor Algebra and Multidimensional Harmonic Retrieval in Signal Processing for MIMO Radar. *IEEE Trans. Sig. Proc.*, 58(11):5693–5705, 2010.

[123] Sejoon Oh, Namyong Park, Lee Sael, and U Kang. Scalable Tucker Factorization for Sparse Tensors - Algorithms and Discoveries. In *Proceedings of the 34th International Conference on Data Engineering, ICDE*, pages 1120–1131, 2018.

[124] E. S. Page. Continuous Inspection Schemes. *Biometrika*, 41(1/2):100–115, 1954.

[125] Ruoming Pang, Mark Allman, Vern Paxson, and Jason Lee. The Devil and Packet Trace Anonymization. *ACM SIGCOMM Computer Communication Review*, 36(1):29–38, 2006.

[126] Namyong Park, Sejoon Oh, and U. Kang. Fast and Scalable Method for Distributed Boolean Tensor Factorization. *The VLDB Journal*, 28(4):549–574, 2019.

[127] Russel Pears, Sripirakas Sakthithasan, and Yun Sing Koh. Detecting Concept Change in Dynamic Data Streams. *Machine Learning*, 97(3):259–293, 2014.

[128] M. Pechenizkiy, J. Bakker, I. Žliobaitė, A. Ivannikov, and T. Kärkkäinen. Online Mass Flow Prediction in CFB Boilers with Explicit Detection of Sudden Concept Drift. *SIGKDD Explorations Newsletter*, 11(2):109–116, 2010.

[129] Ali Pesaranghader and Herna L. Viktor. Fast Hoeffding Drift Detection Method for Evolving Data Streams. In *Proceedings of the 2016 Machine Learning and Knowledge Discovery in Databases, ECML PKDD*, pages 96–111, 2016.

[130] Robin L. Plackett. Some Theorems in Least Squares. *Biometrika*, 37:149–157, 1950.

[131] Viswanath Poosala, Peter J. Haas, Yannis E. Ioannidis, and Eugene J. Shekita. Improved Histograms for Selectivity Estimation of Range Predicates. In *Proceedings of the 1996 ACM International Conference on Management of Data, SIGMOD*, pages 294–305, 1996.

[132] S. W. Roberts. Control Chart Tests Based on Geometric Moving Averages. *Technometrics*, 1(3):239–250, 1959.

[133] Oliver Rösler and David Suendermann. A First Step towards Eye State Prediction using EEG. In *Proceedings of the International Conference on Applied Informatics for Health and Life Sciences, AIHLS*, 2013.

[134] Gordon J. Ross, Niall M. Adams, Dimitris K. Tasoulis, and David J. Hand. Exponentially Weighted Moving Average Charts for Detecting Concept Drift. *Pattern Recognition Letters*, 33(2):191–198, 2012.

[135] Pratanu Roy, Arijit Khan, and Gustavo Alonso. Augmented Sketch: Faster and More Accurate Stream Processing. In *Proceedings of the 2016 International Conference on Management of Data, SIGMOD*, pages 1449–1463, 2016.

[136] Polina Rozenshtein, Aris Anagnostopoulos, Aristides Gionis, and Nikolaj Tatti. Event Detection in Activity Networks. In *Proceedings of the 20th ACM International Conference on Knowledge Discovery and Data Mining, SIGKDD*, pages 1176–1185, 2014.

[137] Polina Rozenshtein, Francesco Bonchi, Aristides Gionis, Mauro Sozio, and Nikolaj Tatti. Finding Events in Temporal Networks: Segmentation Meets Densest-Subgraph Discovery. In *Proceedings of the IEEE International Conference on Data Mining, ICDM*, pages 397–406, 2018.

[138] Polina Rozenshtein, Nikolaj Tatti, and Aristides Gionis. Finding Dynamic Dense Subgraphs. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 11(3):27:1–27:30, 2017.

[139] Caitlin Sadowski and Greg Levin. SimHash: Hash-based Similarity Detection. *Google Technical Report*, 2007.

[140] Saket Sathe and Charu C. Aggarwal. Subspace Histograms for Outlier Detection in Linear Time. *Knowledge and Information Systems*, 56(3):691–715, 2018.

[141] Jeffrey C Schlimmer and Richard H Granger. Incremental Learning from Noisy Data. *Machine Learning*, 1(3):317–354, 1986.

[142] Raquel Sebastião, João Gama, and Teresa Mendonça. Fading Histograms in Detecting Distribution and Concept Changes. *International Journal of Data Science and Analytics*, 3(3):183–212, 2017.

[143] Konstantinos Semertzidis, Evaggelia Pitoura, Evimaria Terzi, and Panayiotis Tsaparas. Finding Lasting Dense Subgraphs. *Data Mining and Knowledge Discovery*, 33(5):1417–1445, 2019.

[144] J P Shaffer. Multiple Hypothesis Testing. *Annual Review of Psychology*, 46(1):561–584, 1995.

[145] Preya Shah, Arian Ashourvan, Fadi Mikhail, Adam Pines, Lohith Kini, Kelly Oechsel, Sandhitsu R Das, Joel M Stein, Russell T Shinohara, Danielle S Bassett, Brian Litt, and Kathryn A Davis. Characterizing the Role of the Structural Connectome in Seizure Dynamics. *Brain*, 142(7):1955–1972, 2019.

[146] Kijung Shin, Bryan Hooi, and Christos Faloutsos. M-Zoom: Fast Dense-Block Detection in Tensors with Quality Guarantees. In *Proceedings of European Conference on Machine Learning and Knowledge Discovery in Databases, ECML PKDD*, pages 264–280, 2016.

[147] Kijung Shin, Bryan Hooi, and Christos Faloutsos. Fast, Accurate, and Flexible Algorithms for Dense Subtensor Mining. *ACM Transactions on Knowledge Discovery from Data*, 12(3):28:1–28:30, 2018.

[148] Kijung Shin, Bryan Hooi, Jisu Kim, and Christos Faloutsos. D-Cube: Dense-Block Detection in Terabyte-Scale Tensors. In *Proceedings of the 10th ACM International Conference on Web Search and Data Mining, WSDM*, pages 681–689, 2017.

[149] Kijung Shin, Bryan Hooi, Jisu Kim, and Christos Faloutsos. DenseAlert: Incremental Dense-Subtensor Detection in Tensor Streams. In *Proceedings of the 23rd ACM International Conference on Knowledge Discovery and Data Mining, SIGKDD*, pages 1057–1066, 2017.

[150] N. D. Sidiropoulos, L. De Lathauwer, X. Fu, K. Huang, E. E. Papalexakis, and C. Faloutsos. Tensor Decomposition for Signal Processing and Machine Learning. *IEEE Transactions on Signal Processing*, 65(13):3551–3582, 2017.

[151] Shaden Smith, Jee W. Choi, Jiajia Li, Richard Vuduc, Jongsoo Park, Xing Liu, and George Karypis. FROSTT: The Formidable Repository of Open Sparse Tensors and Tools, 2017.

[152] Wei Song, Yu Liu, and Jinhong Li. BAHUI: Fast and Memory Efficient Mining of High Utility Itemsets Based on Bitmap. *International Journal of Data Warehousing and Mining*, 10(1):1–15, 2014.

[153] Mingwang Tang and Feifei Li. Scalable Histograms on Large Probabilistic Data. In *Proceedings of the 20th ACM International Conference on Knowledge Discovery and Data Mining, SIGKDD*, pages 631–640, 2014.

[154] Nikolaj Tatti and Aristides Gionis. Density-friendly Graph Decomposition. In *Proceedings of the 24th International Conference on World Wide Web, WWW*, pages 1089–1099, 2015.

[155] Robert Tibshirani. Regression Shrinkage and Selection via the Lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 58(1):267–288, 1996.

[156] Ryan J. Tibshirani, Jonathan Taylor, Richard Lockhart, and Robert Tibshirani. Exact Post-Selection Inference for Sequential Regression Procedures. *Journal of the American Statistical Association*, 111(514):600–620, 2016.

[157] V.S. Tseng, Bai-En Shie, Cheng-Wei Wu, and P.S. Yu. Efficient Algorithms for Mining High Utility Itemsets from Transactional Databases. *IEEE Transactions on Knowledge and Data Engineering*, 25(8):1772–1786, 2013.

[158] V.S. Tseng, Cheng-Wei Wu, P. Fournier-Viger, and P.S. Yu. Efficient Algorithms for Mining Top-K High Utility Itemsets. *IEEE Transactions on Knowledge and Data Engineering*, 28(1):54–67, 2016.

[159] Charalampos Tsourakakis, Francesco Bonchi, Aristides Gionis, Francesco Gullo, and Maria Tsiarli. Denser Than the Densest Subgraph: Extracting Optimal Quasi-cliques with Quality Guarantees. In *Proceedings of the 19th ACM International Conference on Knowledge Discovery and Data Mining, SIGKDD*, pages 104–112, 2013.

[160] Jun-Zhe Wang, Jiun-Long Huang, and Yi-Cheng Chen. On Efficiently Mining High Utility Sequential Patterns. *Knowledge and Information Systems*, 49(2):597–627, 2016.

[161] Tsachy Weissman, Erik Ordentlich, Gadiel Seroussi, Sergio Verdu, and Marcelo J Weinberger. Inequalities for the ℓ_1 Deviation of the Empirical Distribution. *Technical report, Hewlett-Packard Labs*, 2003.

[162] C. W. Wu, P. Fournier-Viger, J. Y. Gu, and V. S. Tseng. Mining Closed+ High Utility Itemsets without Candidate Generation. In *2015 Conference on Technologies and Applications of Artificial Intelligence, TAAI*, pages 187–194, 2015.

[163] Tao Wu and David F. Gleich. Retrospective Higher-Order Markov Processes for User Trails. In *Proceedings of the 23rd ACM In-*

ternational Conference on Knowledge Discovery and Data Mining, SIGKDD, pages 1185–1194, 2017.

[164] W. Wu, B. Li, L. Chen, C. Zhang, and P. Yu. Improved Consistent Weighted Sampling Revisited. *IEEE Transactions on Knowledge and Data Engineering*, 31(12):2332–2345, 2019.

[165] W. Xie, F. Zhu, J. Jiang, E. P. Lim, and K. Wang. TopicSketch: Real-Time Bursty Topic Detection from Twitter. *IEEE Transactions on Knowledge and Data Engineering*, 28(8):2216–2229, 2016.

[166] Z. Xu, M. A. T. Figueiredo, X. Yuan, C. Studer, and T. Goldstein. Adaptive Relaxed ADMM: Convergence Theory and Practical Implementation. In *Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7234–7243, 2017.

[167] Bo Yang, Ahmed S. Zamzam, and Nicholas D. Sidiropoulos. ParaSketch: Parallel Tensor Factorization via Sketching. In *Proceedings of the 2018 SIAM International Conference on Data Mining, SDM*, pages 396–404, 2018.

[168] D. Yang, B. Li, L. Rettig, and P. Cudré-Mauroux. HistoSketch: Fast Similarity-Preserving Sketching of Streaming Histograms with Concept Drift. In *Proceedings of the 2017 IEEE International Conference on Data Mining, ICDM*, pages 545–554, 2017.

[169] D. Yang, D. Zhang, V. W. Zheng, and Z. Yu. Modeling User Activity Preference by Leveraging User Spatial Temporal Characteristics in LBSNs. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 45(1):129–142, 2015.

[170] Dingqi Yang, Bin Li, and Philippe Cudré-Mauroux. POIsketch: Semantic Place Labeling over User Activity Streams. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence, IJCAI*, pages 2697–2703, 2016.

[171] Fan Yang, Fanhua Shang, Yuzhen Huang, James Cheng, Jinfeng Li, Yunjian Zhao, and Ruihao Zhao. LFTF: A Framework for Efficient Tensor Analytics at Scale. *Proceedings of the VLDB Endowment*, 10(7):745–756, 2017.

[172] Tong Yang, Yang Zhou, Hao Jin, Shigang Chen, and Xiaoming Li. Pyramid Sketch: A Sketch Framework for Frequency Estimation of

Data Streams. *Proceedings of the VLDB Endowment*, 10(11):1442–1453, 2017.

[173] Show-Jane Yen and Yue-Shi Lee. An Efficient Approach for Mining High Utility Itemsets over Data Streams. In *Proceedings of Data Science and Big Data: An Environment of Computational Intelligence*, pages 141–159, 2017.

[174] Yikun Ban and Xin Liu and Yitao Duan and Xue Liu and Wei Xu. No Place to Hide: Catching Fraudulent Entities in Tensors. In *The World Wide Web Conference, WWW*, pages 83–93, 2019.

[175] Unil Yun, Heungmo Ryang, Gangin Lee, and Hamido Fujita. An Efficient Algorithm for Mining High Utility Patterns from Incremental Databases with One Database Scan. *Knowledge-Based Systems*, 124:188–206, 2017.

[176] W. Zhang, Z. Lin, and X. Tang. Learning Semi-Riemannian Metrics for Semisupervised Feature Extraction. *IEEE Transactions on Knowledge and Data Engineering*, 23(4):600–611, 2011.

[177] Shuo Zhou, Nguyen Xuan Vinh, James Bailey, Yunzhe Jia, and Ian Davidson. Accelerating Online CP Decompositions for Higher Order Tensors. In *Proceedings of the 22nd ACM International Conference on Knowledge Discovery and Data Mining, SIGKDD*, pages 1375–1384, 2016.

[178] Indrė Žliobaitė, Albert Bifet, Jesse Read, Bernhard Pfahringer, and Geoff Holmes. Evaluation Methods and Decision Theory for Classification of Streaming Data with Temporal Dependence. *Machine Learning*, 98(3):455–482, 2015.

[179] Hui Zou and Trevor Hastie. Regularization and Variable Selection Via the Elastic Net. *Journal of the Royal Statistical Society, Series B*, 67:301–320, 2005.

ISBN 978-82-326-5838-1 (printed ver.)
ISBN 978-82-326-5874-9 (electronic ver.)
ISSN 1503-8181 (printed ver.)
ISSN 2703-8084 (online ver.)

NTNU
Norwegian University of
Science and Technology