Pattern mining in data streams

Pattern Mining in Data Streams

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Technische Universiteit Eindhoven, op gezag van de rector magnificus prof.dr.ir. C.J. van Duijn, voor een commissie aangewezen door het College voor Promoties, in het openbaar te verdedigen op maandag 2 december 2013 om 16:00 uur

door

Hoang Thanh Lam

geboren te An Giang, Vietnam

Dit is goedgekeurd door de promotoren en de samenstelling van de promotiecommissie is als volgt:

voorzitter:	prof.dr. E.H.L. Aarts
1e promotor:	prof.dr. P.M.E. De Bra
copromotor(en)	dr.ir. T.G.K. Calders (Universite Libre de Bruxelles)
leden:	prof.dr. A.P.J.M. Siebes (UU)
	dr. A. Gionis (Aalto University)
	prof.dr.ir. H.A. Reijers
	dr. J. Gama (University of Porto)
adviseur(s):	dr. M. Pechenizkiy

Figure 0-1: The first ever figure of this thesis is reserved for Linh

♡ *This thesis is dedicated to my parents who always encourage me to be a good scientist, also to my lovely daughter and my wife.*

SIKS Dissertation Series No. 2013-36.
The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

This dissertation is sponsored by the Netherlands Organization for Scientific Research (NWO) in the research project Complex Patterns in Streams (COMPASS).

Summary

Recent emerging applications in sensor networks, social media and healthcare systems produce a lot of data which continuously and rapidly grows over time. This data referred to as a data stream is usually monitored in real time. Streaming monitoring systems usually keep a compact summary of the data stream in computer memory devices for efficiently answering different types of data stream queries. Under the streaming context, efficient algorithms are needed for summarizing the stream. This summary needs to be updated quickly in order to keep pace with the high speed data generation and the dynamics of the streams. Moreover, since data streams are usually huge and may be produced by a mixture of parallel independent processes with possible noise, traditional frequent pattern mining approaches usually return very redundant or meaningless sets of patterns.

In this thesis, we study pattern mining problems in the context of data streams. First, we propose memory-efficient data stream summarization approaches and efficient approximation algorithms for mining several well-known types of patterns from data streams. However, mining these types of patterns usually results in very redundant sets of patterns. In order to tackle the redundancy issue, we propose data compression based methods for mining non-redundant and meaningful sets of sequential patterns from a data stream. Finally, we propose a data compression based algorithm to decompose a data stream into independent and parallel components. Having the decomposition of the stream in a preprocessing step, further monitoring of the stream can be done more conveniently with each independent component of the stream separately.

Acknowledgements

This thesis would not have been successfully completed without the guidance and the help of several people who contributed their valuable assistance during the preparation and the development of the thesis work.

I would like to thank professor Toon Calders and professor Paul De Bra for providing me with an excellent chance and freedom to work on the research problems that interest me, also for the patient guidance that helps me to raise my solid steps in a competitive research community like data mining.

I would like to thank professor Pierpaolo Degano at University of Pisa, Dr. Raffaele Perego and Dr. Fabrizio Silvestri from CNR Italy for their useful supports and guidance during my stay in Italy.

I would like to thank Dr. Aristide Gionis for his kindness when he sent a recommendation letter that helped me to successfully get the current PhD position.

I would like to thank Dr. Fabian Mörchen and Dr. Dmitriy Fradkin for providing me with an excellent chance to visit Siemens Corporate Research in Princeton. The core part of my thesis is the result of that enjoyable visit to Siemens.

I would like to thank group members: Mykola, George, Faisal, Sicco, Jorn, Pedro, Julia, Alexander, Yongmin, Jie, Wenjie, Jeroen, Riet and Ine for the support and useful help during my stay at Eindhoven.

I would like to thank the Netherlands Organization for Scientific Research (NWO) for its generous grant to fund my PhD thesis project, Siemens company for its generous internship to fund my visit at its research center in Princeton.

Contents

Chapter 1

Introduction

1.1 Data streams

Recent developments of information technology enable us to collect huge amounts of data from different sources much more easily than a decade ago. The concept of *big data* emerges as one of the most attractive research topics in data science. Nowadays, not only data scientists but also businessmen, social scientists, biologists, and even journalists love to talk about big data. The concept big data does not only regard the *scale* of the data, but it also concerns the *speed* of data generation and the *dynamics* of the data.

In some applications, data is collected once and immediately sent for analysis. Those applications assume that the desired properties of the data do not change over time. A big snapshot of the data is enough for acquiring important properties and statistics of the data. However, in many applications data arrives continuously with high speed, and thus *real-time* data analysis is required to catch up with the dynamics of the data. This type of data, usually called *streaming data*, is part of the big data concept. Different from applications with static big data, data stream applications are more concerned about the speed and the dynamics of the data than the size of the data.

There are many examples of data streams in practice. For instance, twitter streams generated by Twitter[1] can be monitored in real-time for discovering the dynamics of important topics that people are currently talking about. Another example of a data stream application is the *InfraWatch*[2] project in which time series data is collected from a sensor

[1] `https://twitter.com/`
[2] `http://infrawatch.liacs.nl/`

network embedded in the body of a bridge in the Netherlands called *Hollandse Brug*. The data is useful for monitoring the current status of the bridge structure or for estimating the traffic loads on the bridge. Other good examples of data stream applications are *Google Latitude*[3] which collects real-time locations of users through mobile devices for location recommendations, or *MoveBank*[4] which collects animal location data for animal tracking and animal migration study. In general, any streaming algorithm must possess the following desired properties:

- *Memory efficient*: it is impractical to store the whole data stream because a data stream grows indefinitely. For instance, the amount of non-video data generated by the sensor network in the InfraWatch project is almost 5GB per day. In most of the cases, only a window of the most recent data instances is kept in the main memory, the rest of the data must be thrown away or kept in low latency disks. Ideally, when the data mining task is known, only a small summary of the stream is kept on the main memory for solving that mining problem incrementally.
- *Quick summary update*: because data arrives continuously with high speed, summary updates must be quick to catch up with the speed of data generation.
- *Single pass*: some applications require not more than a single pass through data due to the expensive costs of multiple passes.

Example 1 (Data Stream Mining)**.** *Figure 1-1 shows an example of different applications generating time series, spatio-temporal data and text streams. The streams are summarized by an online monitoring system and useful patterns are extracted from the summary. In the figure, we show a set of important patterns (keywords) extracted from a twitter stream by one of the streaming algorithms proposed in this thesis.*

1.2 Pattern mining in Data Streams

1.2.1 The problem

Given rich examples of many types of data streams described in section 1.1, one of the most important tasks of data stream analysis is data exploration. Pattern mining is part

[3]Shutdown by Google and replaced by Google+ `https://plus.google.com/`
[4]`https://www.movebank.org/`

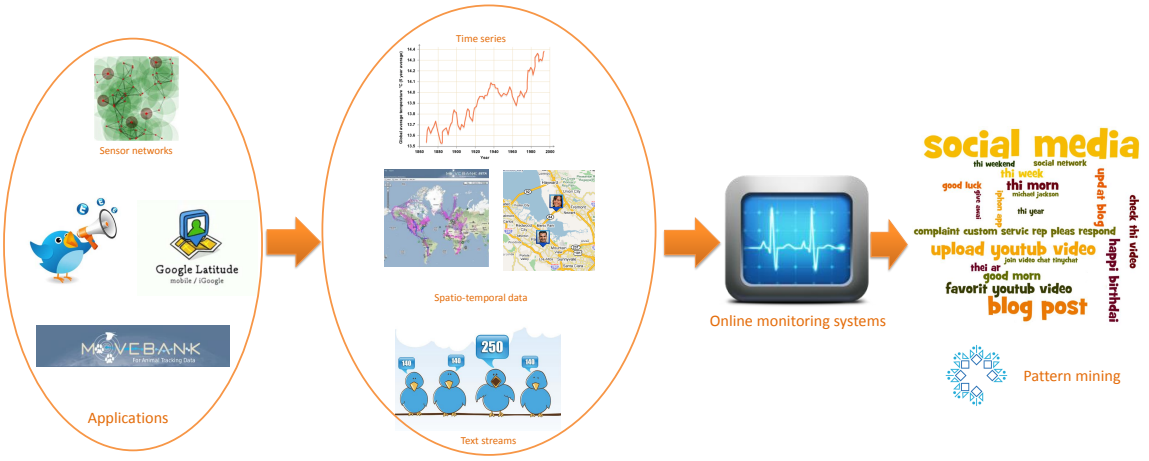


Figure 1-1: Data streams are generated by different types of applications. Monitoring systems summarize these streams and extract useful patterns from the summary. In the figure, we show a set of important patterns (keywords) extracted from a twitter stream by one of the streaming algorithms proposed in this thesis.

of the data exploration process which helps us to reveal important structures of the data such as frequent patterns, association rules etc. [32]. These structures are then visualized with various visualization tools to help people understand the data better [32]. Pattern mining can also be useful for other data mining tasks when the extracted patterns are used as features for classification [15, 34, 26] or clustering [17].

Patterns mining in static datasets with moderate size is a well-studied research problem. However, many pattern mining algorithms proposed for static datasets cannot be used for data stream applications because these algorithms do not handle incremental stream updates. It is important to propose new efficient pattern mining solutions that fit the requirements of data stream applications.

Moreover, traditional frequent pattern mining approaches search for all frequent patterns that occur in the data more frequently than a given support threshold. These approaches often return a large number of patterns that are either very similar to each other or meaningless. This issue usually called *pattern explosion* is a well-known problem in the frequent pattern mining community [75, 11].

One of the negative reasons of the pattern explosion problem is the *redundancy issue*. In the pattern mining problem if the support threshold is set to a low number, we obtain a huge number of patterns. To mine all of them requires very expensive computational efforts. Most of frequent patterns are very similar to each other because when a set is frequent, all

Pattern	Support	Pattern	Support
algorithm algorithm	0.376	method method	0.250
learn learn	0.362	algorithm result	0.247
learn algorithm	0.356	Data set	0.244
algorithm learn	0.288	learn learn learn	0.241
data data	0.284	learn problem	0.239
learn data	0.263	learn method	0.229
model model	0.260	algorithm data	0.229
problem problem	0.258	learn set	0.228
learn result	0.255	problem learn	0.227
problem algorithm	0.251	algorithm algorithm algorithm	0.222

Figure 1-2: The 20 most frequent non-singleton closed sequential patterns from the JMLR abstracts datasets. This set, despite containing some meaningful patterns, is very redundant.

of its subsets are also frequent. Another artifact of the pattern explosion problem concerns the usefulness of the set of frequent patterns. In fact, in many cases, frequent patterns are just combinations of items being frequent but unrelated to each other. As a result, the set of frequent patterns often contains a lot of trivial and meaningless patterns.

Example 2 (Pattern explosion)**.** *In Figure 1-2, we show the top 20 most frequent closed sequential patterns mined from the abstracts of the articles in the Journal of Machine Learning Research (JMLR) dataset. The set of patterns is very redundant, in fact, many patterns are very similar, e.g. "algorithm algorithm", "algorithm algorithm algorithm" etc. Most of them are meaningless because they are just combinations of frequent unrelated (or loose connected) terms such as "problem learn", "problem algorithm" etc.*

Recent work in pattern mining focuses on resolving the pattern explosion issue. The most successful approaches use the minimum description length (MDL) principle and ideas from data compression algorithms [75]. However, so far none of the existing work in the literature addresses these problems in the context of data streams. Hence, this dissertation progresses the state-of-the-art by extending the MDL-based techniques for mining non-redundant patterns to the data stream context.

1.2.2 Challenges

The challenge of pattern mining problems in data streams concerns the increasing complexity when more constraints are added to the problems. In fact, most of the pattern mining

problems are hard. In particular, some of the problems considered in this work belong to the class of NP-hard problems. The complexity of these problems increases significantly with additional constraints on the memory usage, the number of passes through the data and the quick update speed.

For instance, the pattern explosion issue is usually resolved using the MDL principle. In order to apply the MDL-based approaches in streams, first, a new encoding of the data taking into account the temporal aspects of the data is proposed. In the specific encoding that we proposed, for given datasets finding the optimal encoding of the data is already challenging because it belongs to the class of NP-hard and inapproximable problems. The problem becomes more complicated with additional streaming constraints.

1.2.3 Approaches and Contributions

In this thesis, we aim to solve the pattern mining problems in data streams and the pattern explosion problem discussed earlier in subsection 1.2.1. The thesis can be divided into two parts. In the first part, we propose efficient approaches to solve the following pattern mining problems in data streams:

- In chapter 2, we study the top-k most frequent item mining problem with respect to the new measure called *Max Frequency* measure under the *flexible sliding window model*. We provide a theoretical lower-bound on the memory usage of any deterministic algorithm for this problem. A memory efficient approximation algorithm is proposed to solve this problem.

- Chapter 3 discusses the problem of online mining the top-k most similar motifs in time series. We study the lower-bound on the memory usage of any deterministic algorithm and propose an efficient and memory-optimal exact algorithm for this problem.

- Chapter 4 discusses the problem of mining the top-k largest tiles in streams. We propose both exact and approximation algorithms with theoretical guarantee for this problem.

Part one concerns efficient solutions for pattern mining in data streams. However, since the interestingness of patterns extracted by these algorithms is measured based on pattern frequency, the set of patterns usually contains a lot of trivial (or meaningless) patterns and

Method	Patterns			
GOKRIMP	**support vector machin**	**state art**	**neural network**	**well known**
	real world	**high dimension**	experiment result	special case
	machin learn	reproduc hilbert space	sampl size	solv problem
	data set	**larg scale**	supervis learn	signific improv
	bayesian network	independ compon analysi	support vector	object function
Zips	**support vector machin**	featur select	**high dimension**	cross valid
	data set	**machine learn**	paper propose	decis tree
	real world	**bayesian network**	graphic model	**neutral network**
	learn algorithm	model select	**larg scale**	**well known**
	state art	optim problem	result show	hilbert space

Figure 1-3: The first 20 patterns extracted from the JMLR dataset by two algorithms proposed in this thesis work, the non-streaming algorithm GoKrimp and the streaming algorithm Zips. Common patterns discovered by all the two algorithms are bold.

is very redundant. Therefore, in the second part of this thesis, we study compression based methods to resolve these issues:

- In chapter 5, we propose an approach for mining non-redundant sets of sequential patterns in sequence databases. A new encoding for sequence data is proposed. Some complexity results regarding the problem of mining compressing sequential patterns are introduced and efficient heuristic solutions are discussed. Since the given solutions work for only moderate-sized and static datasets, in chapter 6, we solve the problem of mining compressing sequential patterns in data streams. A new online encoding and a scalable linear time algorithm is proposed for data streams.

- Chapter 7 discusses the independent component decomposition problem for sequence data. Decomposing streams in a preprocessing step can resolve the pattern explosion issue when patterns are mined from each decomposed stream separately. We first show the one-to-one correspondence between an independent decomposition and the optimal encoding using the decomposition. This theoretical correspondence encourages us to find the decomposition that results in the optimal encoding for solving the independent decomposition problem. A compression-based algorithm is proposed to find that decomposition for a given sequence.

Example 3 (Compressing patterns)**.** *In Figure 1-3, we show the top 20 compressing sequential patterns mined from the JMLR dataset by two approaches proposed in this thesis, GoKrimp, a non-streaming algorithm (chapter 5) and Zips, a streaming algorithm (chapter 6). The set of patterns makes a lot of sense and is much more interesting than the set of frequent closed patterns depicted in Figure 1-2.*

Part I

Efficient Pattern Mining Algorithms for Data Streams

Chapter 2

Mining Frequent Items in Streams

We study the problem of finding the k most frequent items in a stream of items for the recently proposed max-frequency measure[1]. Based on the properties of an item, the max-frequency of an item is counted over a sliding window of which the length changes dynamically. Besides being parameterless, this way of measuring the support of items was shown to have the advantage of a faster detection of bursts in a stream, especially if the set of items is heterogeneous. The algorithm that was proposed for maintaining all frequent items, however, scales poorly when the number of items becomes large. Therefore, in this work we propose, instead of reporting all frequent items, to only mine the top-k most frequent ones. First we prove that in order to solve this problem exactly, we still need a prohibitive amount of memory (at least linear in the number of items). Yet, under some reasonable conditions, we show both theoretically and empirically that a memory-efficient algorithm exists. A prototype of this algorithm is implemented and we present its performance w.r.t. memory-efficiency on real-life data and in controlled experiments with synthetic data.

[1]This chapter was published as Hoang Thanh Lam and Toon Calders, *Mining top-k frequent items in a data stream with flexible sliding windows.* ACM KDD 2010

2.1 Introduction and related work

In this work we study the problem of mining frequent items in a stream under the assumption that the number of different items can become so large that the stream summary cannot fit the system memory. This occurs, e.g., when monitoring popular search terms, identifying IP addresses that are the source of heavy traffic in a network, finding frequent pairs of callers in a telecommunication network, etc. In stream mining it is usually assumed that the data arrives at such a high rate that it is impossible to first store the data and subsequently mine patterns from the data. Typically there is also a need for the mining results to be available in real-time; at every moment an up-to-date version of the mining results must be available.

Most proposals for mining frequent items in streams can be divided into two large groups: on the one hand those that count the frequency of the items over the complete stream [21] and on the other hand, those based on the most recent part of the stream only [41, 28]. We are here concerned with the latter type. The current frequency of an item is in this context usually defined as either the weighted average over all transaction where the weight of a transaction decreases exponentially with the age of the transaction or by only considering the items that arrived within a window of fixed length (measured either in time or in number of transactions) from the current time. As argued by *Calders et al.* [9, 10], however, setting these parameters, the decay factor and the window size, is far from trivial. As often the case in data mining, the presence of free parameters rather represents additional burden on the user than it provides more freedom. Even worse, in many cases no single optimal choice for the parameters exists, as the most logical interval to measure frequency over may be item-dependent.

For these reasons, the max-frequency was introduced. The max-frequency of an item is parameterless and is defined as the maximum of the item frequency over all window lengths. As such, an item is given the "benefit of the doubt"; for every item its most optimal window is selected. The window that maximizes the frequency can grow and shrink again as the stream grows. Experiments have shown that "*this new stream measure turns out to be very suitable to early detect sudden bursts of occurrences of itemsets, while still taking into account the history of the itemset. This behavior might be particularly useful in applications where hot topics, or popular combinations of topics need to be tracked*" [9].

An algorithm was introduced in [9], based on the observation that even though the

maximal window can shrink and grow again as the stream passes, only a few points in the stream are actually candidate for becoming the starting point of a maximal window; many time points can be discarded. Only for these candidates, called the *borders*, statistics are maintained in a summary. However, the algorithm scales linearly in the number of distinct items in the stream. Therefore, this algorithm is clearly not suitable for systems with limited memory, such as for instance, a dedicated system monitoring thousands of streams from a sensor network. Under such circumstances, a very limited amount of memory is allocated for each stream summary. In this work we extend the work on max-frequency by studying how this problem can be solved efficiently. Our contributions are as follows:

- We show a lower bound on the memory requirements for answering the frequent items query exactly. This bound shows that the dependence on the number of distinct items is inherent to the problem of mining all frequent items from a stream exactly.

- Therefore, we propose, instead of reporting all frequent items, to only mine the top-k with the highest max-frequency. First we prove that in order to solve this problem exactly, again we need a prohibitive amount of memory (at least linear in the number of items). This negative result motivates why our study is extended to finding efficient approximation algorithms for the top-k problem.

- Under some reasonable conditions, we show both theoretically and empirically that memory-efficient algorithms exists. Based on how much knowledge we have about the distribution generating the data stream, different algorithms are given. Prototypes of the algorithms have been implemented and we present their memory-efficiency on real-life and synthetic data.

2.2 Background and Preliminaries

In this work we use the following simple, yet expressive stream model. We assume the existence of a countable set of items $\mathcal{I}$. A stream S over $\mathcal{I}$ is a, possibly infinite, sequence of items from $\mathcal{I}$. We will adopt the notations given in Table 3.2.

Definition 1 (Max-frequency)**.** *Let S be a stream, s be an item, and n a positive integer.*

Table 2.1: Summary of Notations

Notations	Descriptions
s_i	the i^{th} element in the sequence
$S_{j,n}$	the sub-stream $\langle s_j, \ldots, s_n \rangle$ of S
S_n	the sequence $S_{1,n}$
$count(s, S_n)$	the number of instances of s in S_n
$\otimes_k(s)$	the sequence $ss\cdots s$ with length k
ε	an empty sequence
$freq(s, A)$	*Relative Frequency* of item s in the suffix A of S: $freq(s, A) = \frac{count(s,A)}{\lvert A \rvert}$

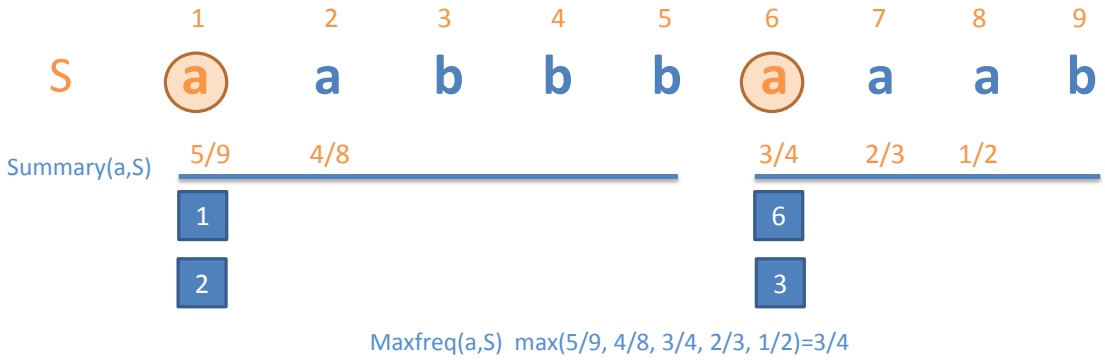


Figure 2-1: Border points of a in S correspond to the highlight positions. $Summary(a, S)$ and $MaxFreq(a, S)$ are also shown in this figure.

The max-frequency of s in S at time n, denoted $MaxFreq(s, S_n)$ is defined as:

$$MaxFreq(s, S_n) := \max_{k=1\ldots n} freq(s, S_{k,n})$$

The smallest k that maximizes the fraction is called the maximal border *and will be denoted $MB(s, S_n)$:*

$$MB(s, S_n) := \min argmax_{k=1\ldots n} freq(s, S_{k,n})$$

In other words, the max-frequency is defined as the maximal relative frequency of s in a suffix of S_n. For example, in Figure 2-1 we can see a stream of items S_n ($n = 9$). The first ever occurrence of a is associated with a relative frequency equal to $\frac{count(a,S_{1,9})}{|S_{1,9}|} = \frac{5}{9}$. Similarly, the second occurrence of a is associated with a relative frequency equal to $\frac{count(a,S_{2,9})}{|S_{2,9}|} = \frac{4}{8}$. With every occurrence of a in S_n a relative frequency is associated. In this

case $MaxFreq(a, S_9) = \frac{3}{4}$ and the maximal border is $MB(a, S_9) = 6$ (the third a from the beginning of the item stream). It is worth noting that the more traditional sliding window approaches consider only one prefix of a user-defined fixed length.

Obviously, for most data streams it is impossible to keep the whole stream into memory and to check, at every timestamp n, all suffixes of S_n in order to find the suffix which gives the maximal frequency. Luckily, however, not every point can become a maximal border. Points that can still potentially become maximal borders are called the *border points*:

Definition 2 (Borders)**.** *An integer $1 \leq j \leq n$ is called a* border *for s in S_n if there exists a continuation of S_n in which j is the maximal border. The set of all borders for s in S_n will be denoted $\mathcal{B}(s, S_n)$:*

$$\mathcal{B}(s, S_n) := \{j \mid 1 \leq j \leq n \mid \exists T \text{ s.t. } S_n \text{ is a prefix of } T \text{ and } MB(s, T) = j\}$$

In Calders et al. [9] the following theorem exactly characterizing the set of border points was proven:

Theorem 1 (Calders et al. [9])**.** *Let S be a stream, s be an item, and n a positive integer, j is in $\mathcal{B}(s, S_n)$ if and only if for every suffix B of S_{j-1} and every prefix A of $S_{j,n}$,*

$$freq(s, B) < freq(s, A)$$

Intuitively, the theorem states that a point j is a border for item a if and only if for any pair of blocks where the first block lies Before and the second After point j, the frequency of a in the first block should be lower than in the second one. For example, we can take a look at the data stream in Figure 2-1 again. Let us consider the fourth occurrence of a corresponding to the seventh position in the stream S_9. If we choose $B = S_{6,6}$ and $A = S_{7,9}$ we have $freq(a, B) = 1 > freq(a, A) = \frac{2}{3}$ which does not satisfy the condition in Theorem 1 and hence this occurrence of a is not a border point. According to Theorem 1, of the 5 instances of a in S_9 only the two highlighted positions could ever become maximal borders. As was shown in [9], this property is a powerful pruning criteria effectively reducing the number of border points that need to be checked.

Clearly, if j is not a border point in S_n, then neither it is in S_m for any $m > n$. Therefore, in [9], the following summary of S_n is maintained and updated whenever a new item arrives:

Table 2.2: Border set summary

j_1	j_2	...	j_b
$count(s, S_{j_1,j_2-1})$	$count(s, S_{j_2,j_3-1})$	...	$count(s, S_{j_b,n})$

let $j_1 < \ldots < j_b$ be the borders of s in S_n. The border set summary is depicted in Table 2.2. This summary can be maintained easily, is typically very small, and the max frequency can be derived immediately from it. For example, in Figure 2-1 we see the border set summary of item a in S_9.

The following theorem allows for reducing the border set summary even further, based on a minimal support threshold:

Theorem 2 (Calders et al. [9])**.** *Let S be a stream, s be an item, n a positive integer, and j in $\mathcal{B}(s, S_n)$. Let T be such that S_n is a prefix of T, and j is the maximal border of s in T. Then, $MaxFreq(s, T) \leq freq(s, A)$ for any prefix A of $S_{j,n}$.*

Hence, if there exists a block A starting on position j such that the frequency of s in A does not exceed a minimal threshold *minfreq*, j can be pruned from the list of borders because whenever it may become the maximal border, its max-frequency won't exceed the minimal frequency threshold anyway.

2.3 Theoretical Results

In this section, we are going to investigate some theoretical aspects of the *mining top-k frequent items with flexible windows problem*. First, we will prove that the number of border points in a data stream with multiple items can increase along with the data stream size and therefore this number will eventually reach the memory limit of any computing system. As a result, it is emerging to design a memory-efficient algorithm that maintains just partial information about the border point set while it is still able to answer the top-k queries with high accuracy.

Besides, we will also show that any deterministic algorithm solving our problem exactly requires a memory space with size being at least linear in the number distinct items of the stream. Thus, under the context that the system memory is limited, we need to design either a randomized exact algorithm or an approximation deterministic algorithm.

We propose a deterministic algorithm to approximate the problem's solutions. More importantly, we will show that the proposed algorithm theoretically uses less memory than the straightforward approach storing the complete set of border points while it is able to answer the top-k queries with surprisingly high accuracy. "High accuracy" here refers to that the algorithm reports the top-k answers at any time point with a bounded low probability of having false negative, without any false positive and the order between the items in the reported top-k list is always correct.

The Number of Border Points

We first start by showing that there exists an item stream such that the number of border points increases along with the data stream size regardless how many distinct items in the data stream there are. Hence, when the item stream evolves the number of border points will eventually reach the memory limit of any computing system.

Assume that we have a set of $m+1$ distinct items $\mathcal{I} = \{a_1, a_2, \cdots, a_m, b\}$ and let w_l denote the sequence $\otimes_l(a_1) \otimes_l (a_2) \cdots \otimes_l (a_m)b$. Consider the following data stream: $W_k = w_0 w_1 w_2 \cdots w_k$ for $k \geq 1$. The data stream size $|W_k|$ is equal to $m\frac{k(k+1)}{2} + k + 1$. The following lemma gives the number of border points of the data stream W_k:

Lemma 1. *Data stream W_k has exactly $mk+1$ border points for any $k \geq 1$.*

Proof. First, regarding the item b there is a unique border point corresponding to the position of the first occurrence of b in W_k. Moreover, we will prove that for every item a_i there are exactly k border points corresponding to the first elements of the sequences $\otimes_j(a_i)$, for $j = 1, 2, \cdots, k$. These k occurrences of the item a_i divide W_k into $k+1$ portions.

For instance, consider a case with $W_k = b\ a_1a_2a_3b\ a_1a_1a_2a_2a_3a_3b\ a_1a_1a_1a_2a_2a_2a_3a_3a_3b$, that is when $k = 3$ and $m = 3$. The three first occurrences of a_1 in every sequence $\otimes_j(a_1)$ divide S_k into 4 portions (separated by $|$): $b|a_1a_2a_3b|a_1a_1a_2a_2a_3a_3b|a_1a_1a_1a_2a_2a_2a_3a_3a_3b$.

Let $P_{i0}, P_{i1}, \cdots, P_{i(k+1)}$ be the portions. The number of instances of the item a_i inside the portion P_{ij} is j and the size of P_{ij} is equal to $m \cdot j + i$. Hence, for every a_i we have a partition of S_k corresponding to a strictly increasing sequence of fractions:

$$0 < \frac{1}{m+i} < \frac{2}{2m+i} < \cdots < \frac{k}{mk+i} \tag{2.1}$$

By the result of the work [9], we can imply that all the considered positions correspond to

the border points of the items a_i in W_k. Thus, for every item a_i we have exactly k border points and thus in total W_k has exactly $m \cdot k + 1$ border points. □

A direct consequence of this lemma is that given a fixed number of distinct items m when the data stream evolves, the number of border points also evolves along with k. It is important to note that the upper bound on the number of border points presented in the work [9] is only for a single item, from this result it is not easy to derive a similar upper bound for the multiple-item case. The result in this section is not as strong as the result presented in [9] for the single-item case. However, for the multiple-item case, it is enough to support the fact that there is no modern computing system being able to store the complete set of border points inside its limited memory when k becomes extremely large.

The Least Space Requirement for Deterministic Exact Algorithms

In this section, we will derive a lower bound on the memory usage that every deterministic algorithm will need in order to solve the top-k frequent items mining problem exactly. In particular, if we assume that the data stream has at least m distinct items we can show that there is no deterministic algorithm solving the top-k problem exactly with memory less than m. By the terminology deterministic algorithm we mean the model in which whenever a new item arrives in the data stream the algorithm has to decide whether it needs to evict an existing border point in the memory or just ignore it deterministically.

The main theoretical result of this section is shown in the following lemma:

Lemma 2. *Let m be the number of distinct items in the data stream. If the system memory limit is $m - 1$, there is no deterministic algorithm being able to answer the top-k ($k > 1$) frequent items queries exactly all the time even for the special case $k = 2$.*

Proof. First of all, at a time point t, if the system has information about an item we call it a *recorded item* and otherwise it is called a *missing item*. Given a data stream, the most recent item in the data stream is always the highest MaxFreq item, so in order to answer the top-2 query exactly this item must be recorded as long as it arrives in the stream.

Therefore, let consider the stream $S = \otimes_m(a_m) \otimes_{m-1} (a_{m-1}) \cdots \otimes_1 (a_1)$ with m distinct items. In this data stream, it is clear that at the moment a_1 must be a recorded item. Since we have assumed that the system has limited memory which is less than m there is always

at least one missing item in S. Without loss of generality we assume that a_n ($n \neq 1$) is the missing item.

We extend the data stream S with $l \geq 1$ instances of the item a_1, s.t. $S = \otimes_m(a_m) \otimes_{m-1} (a_{m-1}) \cdots \otimes_2 (a_2) \otimes_{l+1} (a_1)$. In doing so, a deterministic algorithm does not have any information about a_n so far, hence, the item a_n remains missing for any value of l .

Moreover, every item a_i for $i = 2, 3, \cdots, m$ has a unique border point. This point corresponds to the maximum point of a_i the MaxFreq of which is equal to $\frac{i}{\frac{i(i+1)}{2}+l}$. We will prove that there exists l such that a_n can become the second highest MaxFreq. In fact, in order to prove this, we have to show that the following inequalities are true for some value of l for any $i \neq n$ and $n \geq 2$:

$$\frac{n}{\frac{n(n+1)}{2} + l} > \frac{i}{\frac{i(i+1)}{2} + l} \tag{2.2}$$

We rewrite the above inequality as follows:

$$(n - i)(l - \frac{ni}{2}) > 0 \tag{2.3}$$

If l satisfies $\frac{n(n+1)}{2} > l > \frac{n(n-1)}{2}$ (when $n \geq 2$ there always exists such l) then the inequality (2) is true for all $i \neq n$ and $n \geq 2$. Hence, with such value of l the item a_n becomes the second highest MaxFreq. On the other hand, a_n so far is not a recorded item so it will be missing in the top-2 list reported by every deterministic algorithm. □

Lemma 2 allows to derive a lower bound as large as m, which is the number of distinct items in the stream, on the memory usage of any deterministic algorithm for solving the top-k problem exactly. When m is greater than the memory limit, solving the problem exactly with a deterministic exact algorithm is no longer possible. Therefore, in the following sections we will focus on approximation approaches which are memory-efficient.

Effective Approximation Algorithm

In this subsection we propose an approximation algorithm for the top-k frequent items mining problem. We show that the proposed algorithm can answer top-k queries with high accuracy and consumes less memory than the straightforward approach of storing the complete set of border points. First, we assume that the item distribution in the data stream

is know in advance and does not change over time. We make this assumption to simplify the analysis of the proposed algorithm. For the cases with unknown data distribution we propose another algorithm in the next section.

Prior to the description of the approximation algorithm we revisit a useful property of the border points stated in Theorem 2, Section 2.2. According to this theorem, if the tight lower bound on the MaxFreq of the top-k frequent items is known in advance, then we can safely prune any border point p of an item a which has frequency being strictly less than this bound without effect on the accuracy of the top-k results. We call the value that we use to do pruning *the pruning threshold*. Obviously, zero is a feasible lower bound. However, it is not a meaningful pruning threshold, because there is no border point with relative frequency being less than this threshold.

Indeed, let us revisit the data stream S shown in the proof of Lemma 2. If we extend the data stream S with l instances of a_1 and let l go to infinite we will have a data stream in which the only feasible non-negative lower bound on the MaxFreq of the top-k items is zero, in other words, there is no meaningful pruning threshold for this data stream. Thus, it is impossible to devise a meaningful pruning threshold value for every data stream such that there is no accuracy loss in the top-k results. However, if the item distribution is known in advance we can estimate a good pruning threshold such that the accuracy loss is negligible. We first start with the following lemma:

Lemma 3. *Given a time point n and a parameter k, we assume that Y_n is the size of the smallest suffix of the item stream S_n that contains exactly k distinct items. The top-*k *frequent items of the data stream S_n have MaxFreq at least as big as $\frac{1}{Y_n}$*

Proof. Since the window with size Y_n contains k distinct items their relative frequency in this window is at least $\frac{1}{Y_n}$. On the other hand, the top-k frequent items are always at least as frequent as the least frequent item in this window so the Lemma 3 holds. □

The consequence of Lemma 3 is that at every time point the reciprocal of the smallest suffix of S_n containing exactly k distinct items is the lower bound on the MaxFreq of the top-k frequent items. This lower bound is not a fixed value but it may change when the data stream evolves. Let the size of the smallest suffix containing exact k distinct items be denoted by the random variable X_k. Estimating the expected value of X_k is well-known in the literature as the classical *Coupon Collector Problem* (CCP) [55]. We will revisit this

Algorithm 1 $MeanSummary(k, l)$

```
1: B ⟵ ∅
2: while New item a arrives do
3:    if previous occurred item is not a then
4:       create a new border point for a and include it into B
5:    end if
6:    Delete all the elements in B that are no longer border points
7:    Update relative frequency of all elements in B
8:    Delete all the border points in B that have relative frequencies strictly less than β = 1/(lE(X_k))
9: end while
```

problem later in the analysis part. At this point, we will start with the description of the MeanSummary algorithm summarizing the data stream as in Algorithm 1.

Recall that X_k is the random variable standing for the size of the smallest window containing exact k distinct items. Algorithm 1 assumes that $E(X_k)$ is known in advance for any k and uses $\frac{1}{lE(X_k)}$ as a pruning threshold, where k and l are two user-defined parameters. Having a stream summary the system just needs to take the k highest relative frequency items present in this summary and report this list as the answer to the top-k query. To understand how precise the top-k list produced by MeanSummary is we make an analysis in the subsequent part.

Accuracy Analysis

Lemma 4. *Given two positive integers k and l, independently of the item distribution in the data stream, the probability that a top-*k *item is missing in the answer list reported by MeanSummary is less than $\frac{1}{l}$*

Proof. Given a time point, since $\frac{1}{X_k}$ is the lower bound on the MaxFreq of the top-k frequent items, the event that the least frequent item in the top-k frequent items has its MaxFreq less than $\frac{1}{lE(X_k)}$ is less probable than $\frac{1}{X_k} \leq \frac{1}{lE(X_k)}$. On the other hands, according to the Markov's inequality $Pr(X_k \geq lE(X_k)) \leq \frac{1}{l}$ which proves the lemma. □

By the result of Lemma 4, the probability of having false negative is less than $\frac{1}{l}$, independently of the item distribution. In the next section we will derive a better bound for this type of error when the data set follows the uniform distribution in which the expectation and the variance of X_k are known.

Finally, we consider another interesting property of the proposed stream summary: MeanSummary is able to answer top-k queries without false positive, that is, the reported

top-k list is always a subset of the right answer and the item order in the reported top-k list is preserved.

Lemma 5. *Using MeanSummary we are able to answer the top-*k *queries without false positive, moreover, the item order in the reported top* $-$ k *list always correct.*

Proof. Given a time point, assume that a is the $k-th$ most frequent item of the item stream S. If a is not present in MeanSummary, that is, when $MaxFreq(a, S) < \beta$, the other items less frequent than a must also be absent in the summary. In this case, the top-k list produced by taking only items present in the summary will not contain any other items than the real top-k items.

On the other hand, if a is present in the summary, that is when $MaxFreq(a, S) > \beta$ and the MaxFreq of other top-k frequent items must also larger than β, therefore, the border points corresponding to the maximum points of these items must be also present in the summary. In other words, the summary will report the right MaxFreq of them, and hence these items remain in the top-k of the summary and are all present in the answer list.

Since the MaxFreqs of the top-k items we report based on the data stream summary are always exact, the items in the answer list will always be in the correct order. □

Uniform Distribution

Algorithm 1 uses prior knowledge about the item distribution to define the pruning threshold. In particular, it requires the expected value of X_k for any $k > 1$. Estimating $E(X_k)$ is well-known as the classical Coupon Collector Problem [55]. Unfortunately, to the best of our knowledge, there is no work being able to present the value of $E(X_k)$ in a closed form for all types of distributions [25, 59]. Indeed, in [25], the authors have tried to present $E(X_k)$ in form of integral formulae which can be estimated by classical numerical methods. These methods however are computationally demanding, especially when k is large [25]. Estimation of this expected value for any distribution is out of the scope of this work.

Fortunately, for the uniform distribution a simple presentation of $E(X_k)$ is well-known [55, 25]. In addition to that the variance of X_k, denoted by σ_k^2, has a closed formula. Having knowledge of the expected value and the variance of X_k we can derive a tighter bound on the false negative error for this particular distribution as follows:

Lemma 6. $Pr(X_k \geq lE(X_k)) \leq \frac{1}{(l-1)^2+1}$ *for any* $l > 1$ *and* $k < \frac{m}{2}$

Table 2.3: Recall (%) of the top-1 000 answer produced by MeanSummary

Value of l	$l = 0.5$	$l = 0.7$	$l = 0.9$	$l = 1.0$	$l = 2.0$
Min	50.1	71.1	92.5	100	100
Average	51.2	72.4	94.5	100	100
Max	52.4	74.1	96.2	100	100

Proof. First, we have the well-known formulae [55, 25] of $E(X_k)$ and σ_k^2:

$$E(X_k) = \sum_{i=0}^{k-1} \frac{m}{m-i} \tag{2.4}$$

$$\sigma_k^2 = \sum_{i=0}^{k-1} \frac{mi}{(m-i)^2} \tag{2.5}$$

Recall that m is the number of distinct items in the data stream and that it is much larger than k. Since $k < \frac{m}{2}$, for any $i < k$ we have $\frac{i}{m-i} < 1$ and $\sigma_k < \sqrt{E(X_k)} < E(X_k)$.

By the one-sided version of Chebyshev's inequality we get: $Pr(X_k - E(X_k) \geq (l-1)\sigma_k) \leq \frac{1}{(l-1)^2+1}$, which implies $Pr(X_k \geq lE(X_k)) \leq \frac{1}{(l-1)^2+1}$. □

Lemma 6 gives a tighter bound on the false negative than the bound in Lemma 4: the false negative probability is less than $Pr\left(\frac{1}{X_k} \leq \frac{1}{lE(X_k)}\right) = Pr(X_k \geq lE(X_k))$, hence, less than $\frac{1}{(l-1)^2+1}$.

Experiment with Uniform Data

In order to demonstrate the effectiveness of the proposed algorithm we have carried out an experiment on a synthetic data set which follows the uniform distribution. We simulate an item stream with 10 000 uniformly distributed distinct items until the stream size reaches the number 100 000. We have measured the performance of MeanSummary in terms of memory consumption and the quality of the top-1 000 answer. The results are reported in Figure 2-2 and Table 2.3.

According to Lemma 5, MeanSummary does not cause any false positive so it always produces the top-k answer with maximum *Precision Value*, i.e. 100%. On the other hand, MeanSummary may produce false negative which are usually measured by the *Recall Value*, i.e. the fraction of the number of real top-k items reported by MeanSummary. In Table 2.3 we show the Average Recall of the top-1 000 answers produced by MeanSummary over

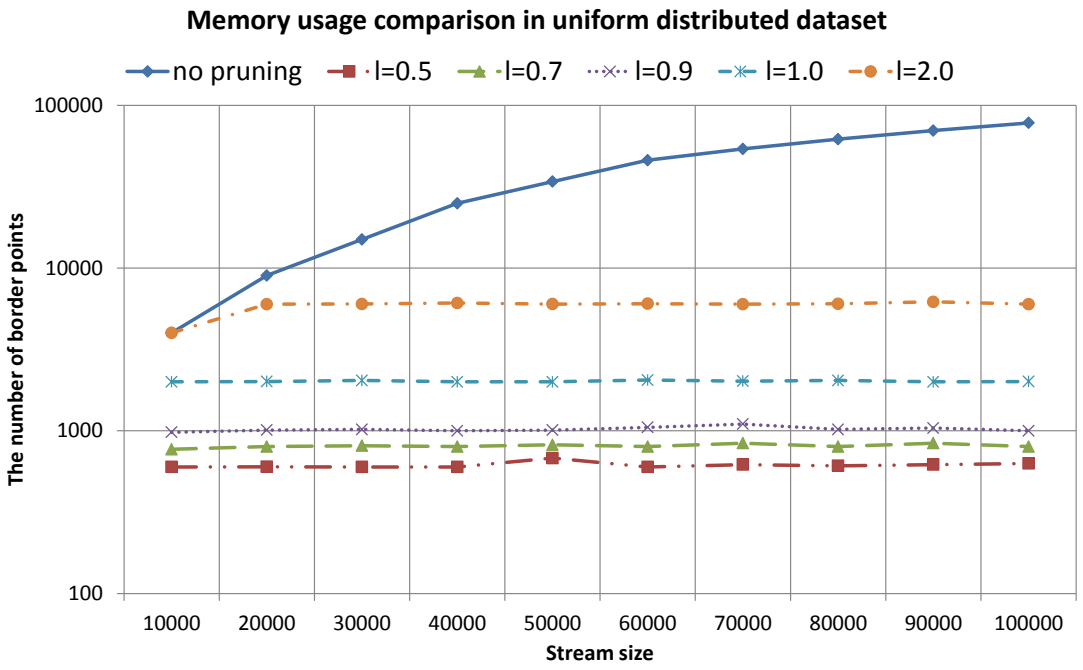


Figure 2-2: The comparison of memory usage in terms of the number of border points each algorithm has to store in memory. The plot shows how this number evolves when the data stream evolves. MeanSummary memory usage is shown for different values of l. It is clear from the figure that MeanSummary uses significantly less memory than the complete set of border points (No Pruning).

different pruning thresholds when the data stream evolves. It is clear from the context that when the value of l is higher, MeanSummary is able to answer the top-1 000 queries more accurately on average. The average recall reaches its maximum value i.e. 100% when l is above 1. In order to show the stability of the obtained results we also report the Maximum and the Minimum Recall values of the top-1 000 answers in this table. It is clear from the context that the obtained results are also quite stable as the differences between the minimum, the maximum recall and the average recall are not large.

It is important to note that the higher the value of l, the lower the pruning threshold becomes, resulting in less border points being pruned. In other words, when l is high MeanSummary may use more memory but it will produce more accurate answers. This is a tradeoff between the result quality and the memory usage. In order to show the effectiveness of MeanSummary in terms of memory usage we plot the number of border points that MeanSummary has to store over different values of the pruning threshold in Figure 2-2. It is clear that when l is higher, more memory space is required. Fortunately, the number of

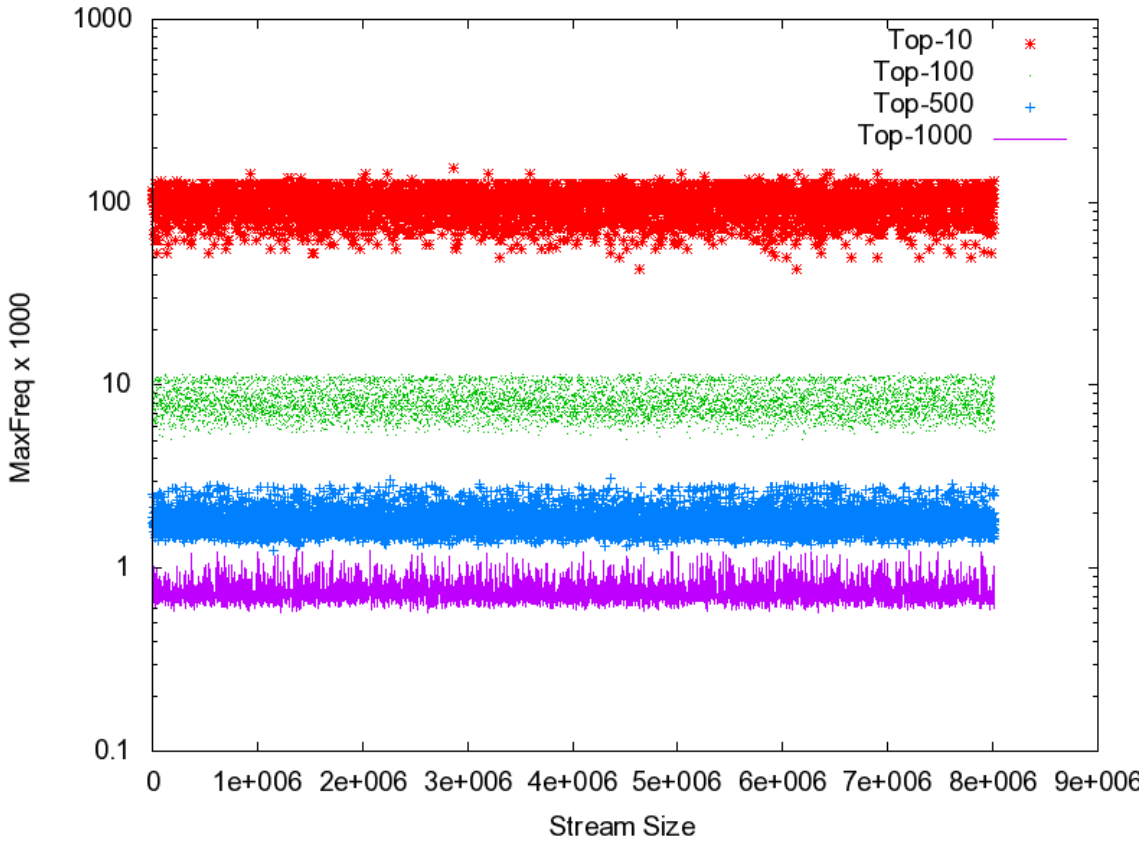


Figure 2-3: The MaxFreq of the *k-th* most frequent item of the Kosarak data stream over time.

border points that MeanSummary has to store seems to be bounded regardless of the stream size. Concretely, at the end of the algorithm execution the number of border points that Meansummary has to store when $l = 2.0$ is always less than 3 000 which is almost ten times smaller than the total number of border points of the data stream (28 793). It is important to notice that when $l = 2.0$, Meansummary produces no errors at all for the top-1 000 queries (see Table 2.3). This fact emphasizes the extreme significance of MeanSummary.

2.4 Pruning Algorithm in Practice

We have seen the effectiveness of using MeanSummary in Algorithm 1 to answer the top-k queries with the assumption that the expected value of X_k could be computed in advance. However, in practice, this effort may not be possible due to non-trivial expected value computation. Therefore, using $\frac{1}{lE(X_k)}$ as a pruning threshold is impractical when the data

Algorithm 2 $MinSummary(k, l)$

```
1: B ← ∅
2: β ← 0
3: while New item a arrives do
4:    if previous occurred item is not a then
5:       create a new border point for a and include it into B
6:    end if
7:    Delete all the elements in B that are no longer border points
8:    Update frequency of all elements in B
9:    Delete all the elements in B that have frequencies strictly less than β
10:   α ← MaxFreq of the least frequent item in B
11:   |B| ← the number of distinct items in B
12:   if |B| ≥ l AND α > β then
13:      β ← α
14:   end if
15: end while
```

distribution is unknown in advance or the data distribution changes over time. Even in the case that the data distribution is supposed to be known in advance, for instance, with the Zipfian distribution, there is no closed representation of $E(X_k)$ that is easy to compute [25].

For the situation that the expectation of X_k is unknown, we now propose another summary algorithm which still has similar properties as MeanSummary. Before continuing with a detailed description of the proposed summary in Algorithm 2, we explain the crucial idea behind our proposal in Figure 2-3. In this figure, we plot the MaxFreq (multiplied by 1 000) of the *k-th* most frequent item of the Kosarak data stream (see section 2.5 for information about this dataset). Four lines correspond to different values of k from which we can observe that they are quite well-separated from each other. Intuitively, if we consider the upper bound on the top-100 line as a pruning threshold we can prune a lot of border points while the top-10 items are warranted to be present in the summary with high probability.

The aforementioned observation from the Kosarak data set is intuitive, allowing us to propose a summary algorithm which is then shown to be effective in the experiments with real-life datasets. The approach is briefly described in Algorithm 2. MinSummary is similar to MeanSummary, the only difference is that it uses a dynamic pruning threshold β instead of a static value. This threshold is updated in each step such that its value is monotonically increasing, thus, more border points are pruned in the further steps. The algorithm admits two user-defined parameters k and l. The bigger the value of l, the bigger the summary will become, but the more precise the top-k queries will be answered.

In order to prevent β from increasing too fast we only update the value of β when the summary $\mathcal{B}$ contains at least l different items. By doing so, we keep β increasing

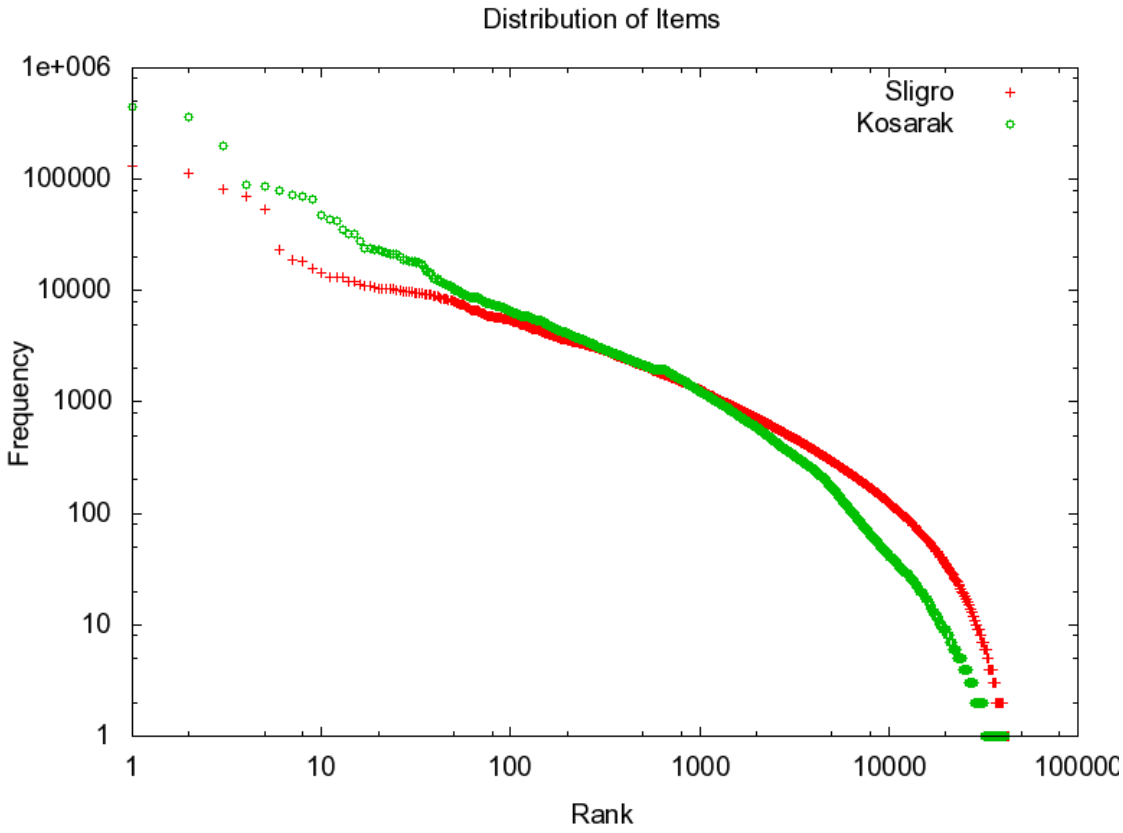


Figure 2-4: The Kosarak and Sligro data-sets follow Zipfian distribution but with different levels of skewed.

but always less than the upper bound of the *l-th* most frequent item. As a result, β is less than the frequency of *k-th* most frequent item with high probability as explained in the aforementioned intuitive example. The following lemma shows that MinSummary has similar properties as MeanSummary:

Lemma 7. *Using MinSummary we are able to answer top-*k *queries without false positive and the item order in the reported top-*k *items is preserved.*

It is important to note that β in MinSummary is kept increasing over time to let MinSummary have similar property like MeanSummary. In doing so, the proof of Lemma 7 proceeds in a very similar way as the proof of Lemma 5.

Table 2.4: Data Sets Summary

Data Sets	N. Trans	Stream Size	Size (MB)	N. Items
Kosarak	990002	8019015	32 MB	41269
Sligro	542194	7671058	38 MB	43082

Table 2.5: Recall value of top-k answers produced by MinSummary

		Kosarak					Sligro				
$k = 100$	**Value of l**	100	120	140	180	200	100	120	140	180	200
	Min	36	49	60	84	100	21	43	60	97	100
	Avg.	66.8	78.2	88.4	98.8	100	67.9	85.4	96.1	99	100
	Max	96	100	100	100	100	99	100	100	100	100
$k = 1\,000$	**Value of l**	1 000	1 100	1 200	1 500	2 000	1 000	1 100	1 200	1 300	1 400
	Min	42.8	53.4	56.8	70.1	92.5	61.1	85.9	100	100	100
	Avg.	60.3	65.7	70.7	85.6	99.4	83.7	91.5	100	100	100
	Max	100	100	100	100	100	99.9	100	100	100	100

2.5 Experiments and Results

Data sets

We use two real world data sets to conduct our experiments. The characteristics of them are summarized in Table 2.4. The Kosarak is a publicly available dataset[2] containing click-stream data of a Hungarian online news portal while the Sligro data set is released under restricted conditions containing information about products purchased by Sligro company's customers in a specific city in the Netherlands from August 2006 to October 2008.

Both data sets follows a Zipfian distribution as we can see in Figure 2-4 in which we plot the item frequency (vertical axis) from the most frequent to the least frequent items (horizontal axis). Generally, the Sligro data set follows a Zipfian distribution but the occurrence of every specific item varies in different periods of the year. Some items may be suddenly frequent in a short period of time and may completely disappear in another period, e.g. seasonal products. We conduct the experiment on the Sligro dataset to see the behavior of MinSummary in the context of rapidly changing frequencies.

Accuracy of the Top-k Answer

It is worth noting that by the results of Lemma 7, MinSummary always produces the top-k list with the maximum precision value, i.e. 100%. So in order to measure the accuracy of

[2]http://fimi.cs.helsinki.fi/data/

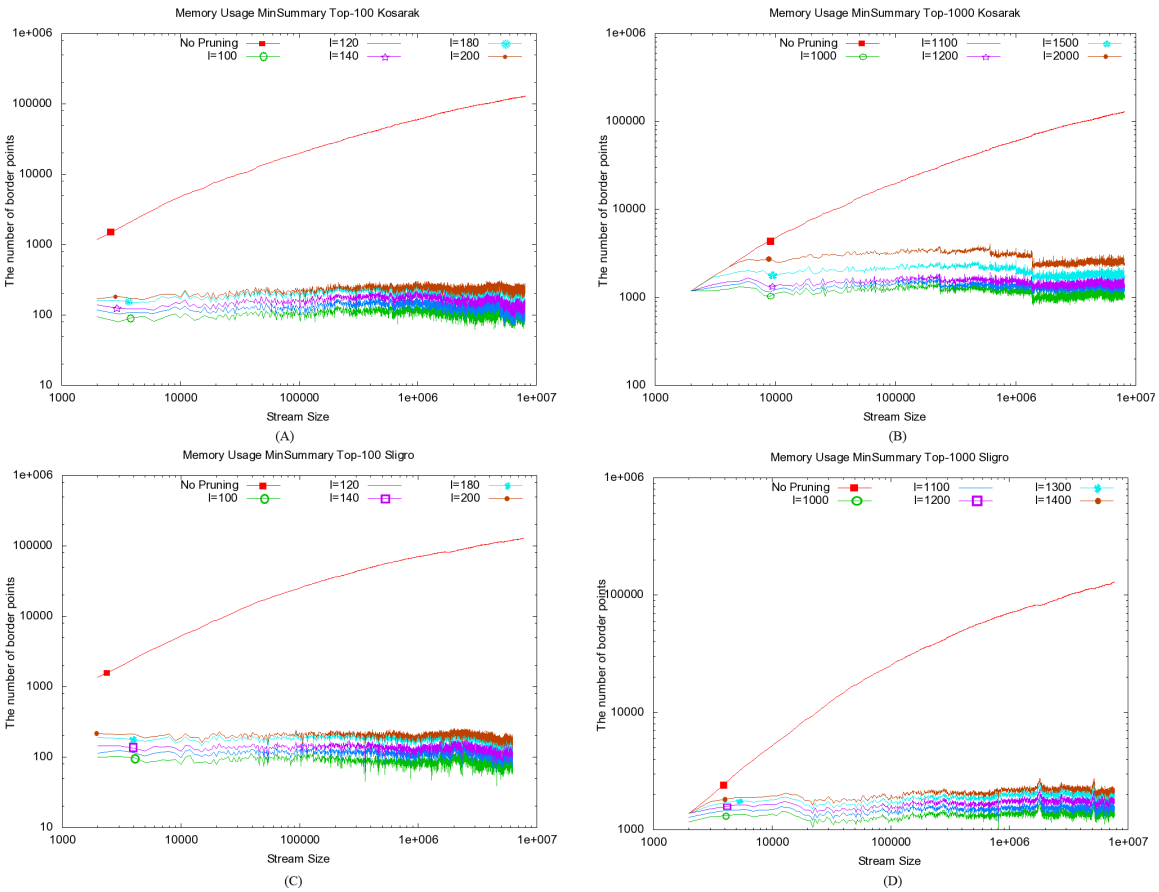


Figure 2-5: A comparison of memory usage in terms of the number of border points each algorithm has to store in memory. The plot shows how this number evolves when the data stream evolves. MinSummary's memory usage is shown with different values of l. It is clear from the figure that MinSummary uses significantly less memory than the case with no border points are being pruned (No Pruning).

MinSummary we just need to measure the recall values of the top-k lists.

Concretely, each time when a new item arrives in the data stream we do querying from MinSummary for the top-k frequent items. The obtained top-k list from MinSummary will then be compared with the true top-k set to estimate the recall value. We average the recall value over time and show the results in Table 2.5. We also present the maximum and the minimum recall values in this table to see the deviation of the recall from its average value.

We present results for different values of the parameters. The value of k is set to 100 and 1 000 respectively while l is set to k and higher. Recall that l is a parameter that controls the memory usage in MinSummary. The higher value of l the more memory is used to maintain MinSummary.

In Table 2.5 we can observe that whenever l is increased the obtained top-k list is more accurate. In particular, we are able to reach the maximum recall value with proper choice of l. The obtained results with higher values of l are also very stable since the deviation of maximum and minimum recall from its average value is negligible. In summary, using MinSummary with a proper choice of its parameters we are able to answer the top-k query with very high accuracy for these particular real world datasets.

Evolution of the Pruning Threshold

In order to illustrate the relation between the pruning threshold and the performance of MinSummary, we plot the value of β with different setups of parameter l in Figure 4-9. We also plot the MaxFreq of the $k-th$ highest frequent item in the stream corresponding to $k = 100$ and $k = 1\,000$.

It is clear from the context that β starts with a very low value and increases every time a new item arrives in the stream. With a proper choice of l we can let β increase up to a tight lower bound on the MaxFreq of the top-k frequent items. In doing so, as the pruning threshold grows, we prune more border points while preserving the accuracy of the top-k answers. Concretely, assume that we intend to answer the top-1 000 frequent item query, $l = 2\,000$ or $l = 3\,000$ are the proper choices because in these cases β (lines "l=2 000" and "l=3 000") increases up to the lower bound on the MaxFreq of the top-1 000 frequent items (line "Top-1 000"). Obviously, when $l = 3\,000$ we have a more accurate answer because there is no overlap between the line "l=3 000" and the line "Top-1 000", but MinSummary will consumes more memory as compared to the case $l = 2\,000$. For top-100 queries it is clear that $l = 1\,000$ is a very good choice.

Memory Usage

Following section 2.5 about the accuracy of the obtained top-k lists, we plot the memory usage of MinSummary in Figure 2-5. In each plot we compare the size of MinSummary in terms of the number of border points that it has to store with the complete set of border points of the data stream (No Pruning line). We can observe that the size of MinSummary increases with increasing l. Yet, if we compare these value with the complete set of border points we see that MinSummary consumes significantly less memory and the memory consumption seems to be independent from the stream size.

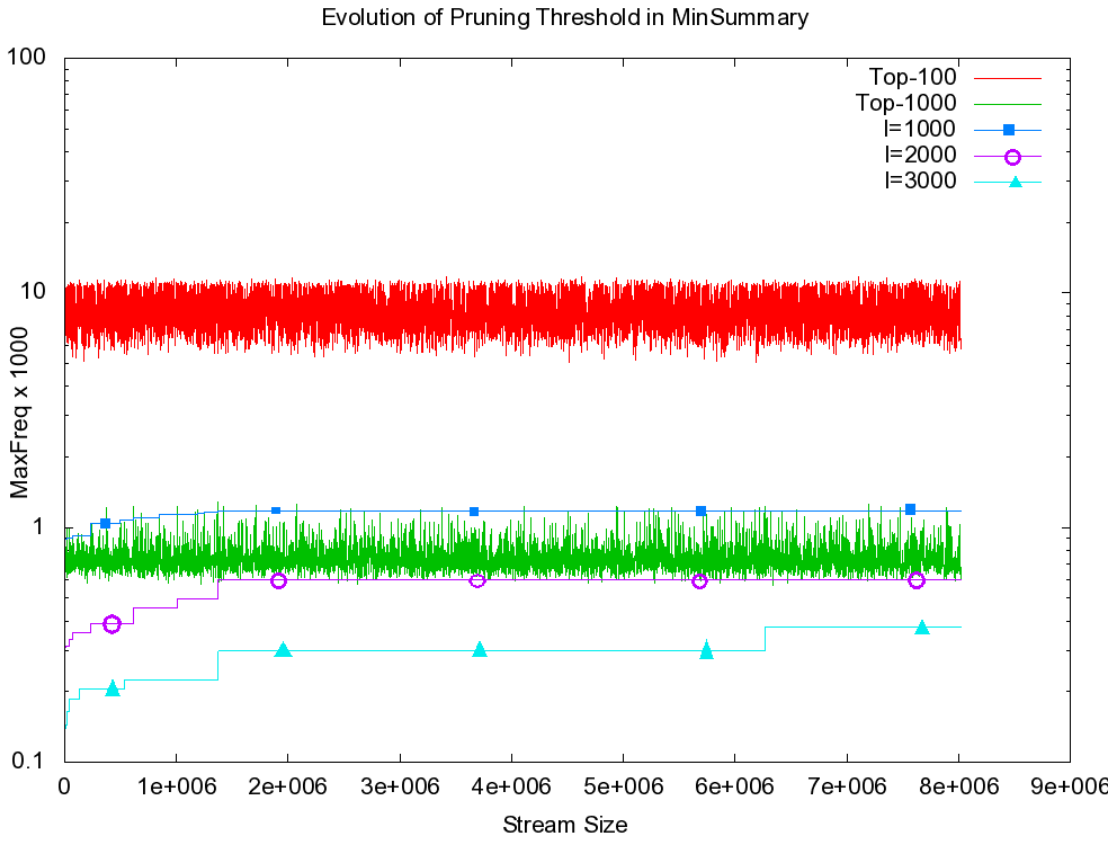


Figure 2-6: Evolution of the pruning threshold β over time. β starts with low value and increases up to the tight lower bound of the top-k MaxFreq

For instance, consider the case in Figure 2-5.B in which we plot the memory usage of the MinSummary with the Kosarak data stream and $k = 1\,000$. When the program finishes its execution, MinSummary stores less than $4\,000$ border points for all values of l while the complete set of border points still keeps evolving and reaches its highest value $128\,249$. That means MinSummary is 300 times more memory-efficient than the straightforward approach. Moreover, we have seen in Table 2.5 that when $l = 2\,000$, MinSummary produces top-k answers with negligible errors. This fact confirms the significance of MinSummary in comparison with the straightforward approach.

The results with the Sligro dataset in Figure 2-5.C and 2-5.D confirm again the significant performance of MinSummary. Another important point we can observe is that given a value of l, the size of MinSummary seems to be bounded regardless of the stream size. This means when the stream evolves the set of the border points will eventually reach any limit but MinSummary size will not.

Chapter 3

Online Mining Time Series Motifs

A motif is a pair of non-overlapping sequences with very similar shapes in a time series. This chapter discusses the online *top-k* most similar motif discovery problem[1]. A special case of this problem corresponding to $k = 1$ was investigated in the literature by *Mueen and Keogh* [56]. We generalize the problem to any k and propose space-efficient algorithms for solving it. We show that our algorithms are optimal in term of space. In the particular case when $k = 1$, our algorithms achieve better performance both in terms of space and time consumption than the algorithm of *Mueen and Keogh*. We demonstrate our results by both theoretical analysis and extensive experiments with both synthetic and real-life data. We also show possible application of the *top-k* similar motifs discovery problem.

[1]This chapter was published as: Hoang Thanh Lam, Toon Calders, Ninh Pham: *Online Discovery of Top-k Similar Motifs in Time Series Data.* SDM 2011

3.1 Introduction

Usually, timeseries data is easy to collect from many sources. For instance, a set of sensors mounted on a single bridge in a city in the Netherlands produces GBs of data per day [39]. Online managing such amounts of data is very challenging. Typically, such data is processed online either at the distributed devices themselves, or at a designated and centralized system. In both cases, the limitations of system resources such as processing power and memory challenge any data mining task. We study the online motif discovery problem under that context.

Time series motifs are repeating sequences which have different meanings depending on the application. For example, Figure 3-1 shows an example of one-second long motifs discovered by our algorithm in a time series corresponding to brain activity. These motifs reflect the repetition of the same sequence of actions in a human brain and can be useful in predicting the seizure period during an epileptic attack [80]. Besides, motif discovery was shown to be useful in many other applications as well, including time series data compression, insect time series management and even in the robotics domain

In [56], *Mueen and Keogh* proposed several methods for summarizing the time series effectively and answering the motif query exactly at any time point in the sliding window setting. Their work, however, only focuses on the discovery of a single motif. Therefore, some interesting motifs could be missed accidently. For example, Figure 3-2 shows two motifs in the *Insect* time series. The first one in red likely corresponds to an idle or steady state of the device; this type of motif is probably not interesting. The second motif in green, however, is more interesting because it is likely that some important events happen at these time intervals. Therefore, an inherent advantage of the *top-k* motifs discovery for $k > 1$ is that the users are allowed to choose the meaningful motifs for their applications and remove the meaningless ones while this is not allowed in the special case $k = 1$. Motivated by this reason we extend the prior work to the general case for any k. In summary, our contributions to this work are as follows:

- We extend the exact motif discovery problem to the *top-k* motifs problem and motivate this extension.
- We derive a theoretical lower-bound on the memory usage for any exact algorithm solving the *top-k* motifs discovery problem. We show that the obtained lower-bound

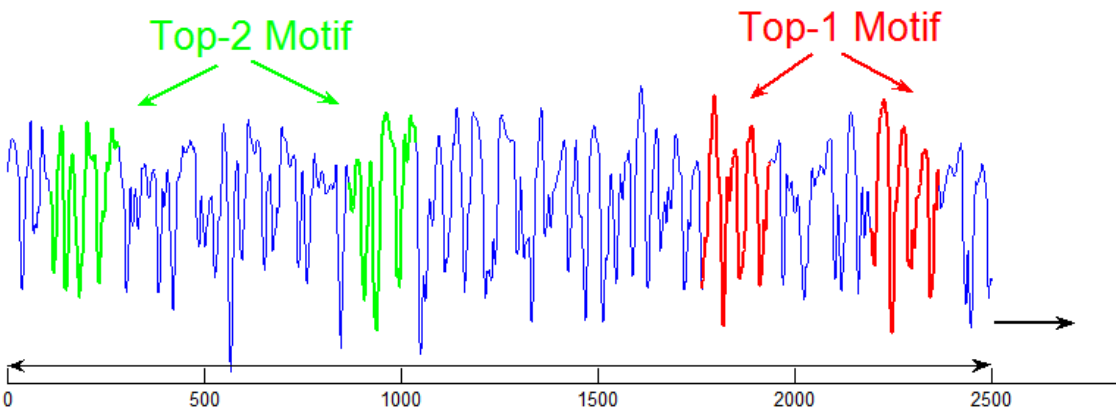


Figure 3-1: An Example of motifs in brain activity data. Two non-overlapping subsequences with similar shapes are defined as a motif. Picture looks better in color.

Table 3.1: Summary of Notations

Notations	**Descriptions**
r_i	i^{th} element of a time series
S_t	time series at time point t
m	motif length
w	number of m-dimensional points in the window
W	number of float values in the window
P_i	the m-dimensional point ending on time-stamp i
$d(P_i, P_j)$	Euclidean distance from P_i to P_j
L_i	forward nearest neighbor list or promising list

is tight by showing a trivial algorithm that achieves the bound.

- As the trivial algorithm is not very efficient at query time, we propose two new algorithms, *nMotif* and its refinement *kMotif*. Both proposed algorithms are not only close to optimal in terms of memory usage but also fast in comparison to the existing algorithm denoted as *oMotif* as for the original motif discovery algorithm for $k = 1$ [56].

- We demonstrate the significance of our work not only with theoretical analysis but also with extensive experiments with real-life and synthetic data.

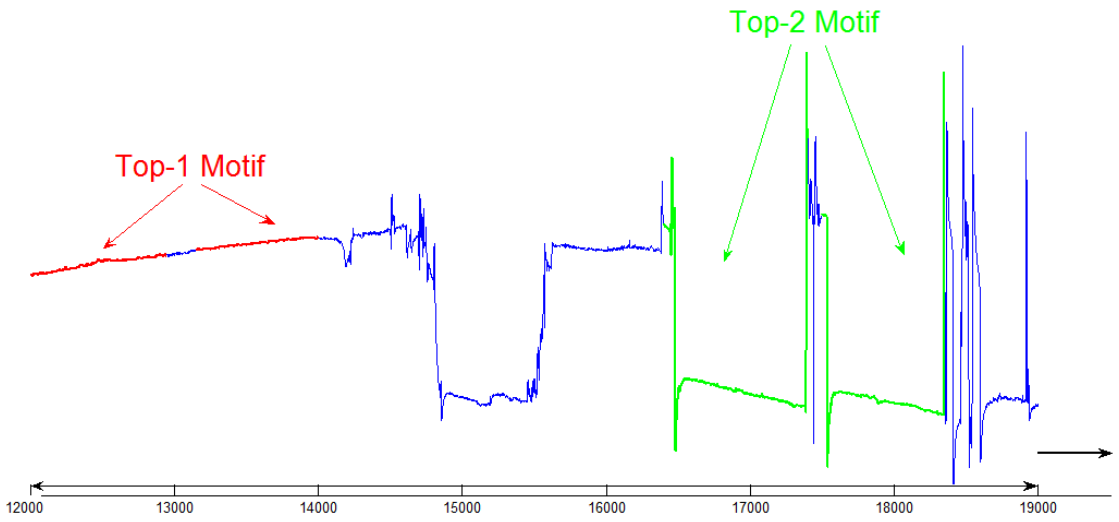


Figure 3-2: An Example of motifs in the insect data set. The motif in red likely corresponds to a steady state and hence does not present useful information. The second motif in green, however, may correspond to an important event. Figure looks better in color.

3.2 Related Work

The importance of motif discovery in time series has been addressed in several communities, including motion-capture, telemedicine and severe weather prediction. Several algorithms tackling this problem are based on searching a discrete approximation of the time series [16, 62, 43, 24, 80, 12].

In recent years, some algorithms have been proposed for finding exact motifs directly in the raw time series data. The approach of [58] is the first tractable exact motif discovery algorithm based on the combination of early abandoning the Euclidean distance calculation and a heuristic search guided by the linear ordering of data. The authors also introduced for the first time a disk-aware algorithm for exact motif discovery for massive disk-resident datasets [57].

Although there has been significant research effort spent on efficiently discovering time series motifs, most of the literature has focused on fast and scalable approximate or exact algorithms for finding motifs in static offline databases. In fact, for many domains including online compression, robotics and sensor-networks, it requires online discovery of time series[56].

In [56], Mueen et al. propose an algorithm for solving the *exact motif discovery problem*. Their approach is denoted as *oMotif* in our work as for the original motif discovery

algorithm. The key idea behind the algorithm *oMotif* is to use a nearest neighbors data structure to answer the motif query fast. Although this approach demonstrates a space-efficient algorithm for exact motif discovery in real time, in several real applications with large sliding window sizes, this algorithm is very space demanding. Furthermore, in many real applications, the discovery of top-k motifs may be more useful and meaningful than a single motif. It is non-trivial to modify the *oMotif* algorithm to the problem of online top-k motifs discovery.

3.3 Problem Definition

Let $S_t = r_1, r_2, \cdots, r_t$ be a time series, in which all r_i are float values. In our theoretical analysis, we will assume that storing a float value requires one memory unit. When the time series S_t is evolving its length also evolves. In many applications, due to the limitation of the system's memory only a window of the W most recent float values in the time series, corresponding to the values $r_{t-W+1}, \ldots, r_t$, are kept in memory while the less recent ones are discarded. When a new value appears in the time series, the window is shifted one step further; the new item is appended to the sliding window, and the oldest one is removed. People usually refer to this as the *sliding windows model*.

Given a window of length W, a sequence of m consecutive float values in this window can be seen as a point in a m-dimensional space. Hence, W float values in the window form a set of $w := W-m+1$, m-dimensional points, which will be denoted as $P_{t-w+1},\ P_{t-w+2}, \cdots,\ P_t$. Notice that W refers to the number of float values in the window and w refers to the number of m-dimensional points in the window. Each point P_i is associated with the time-stamp i of its last coordinate. The smaller the time-stamp, the earlier the point is. In [56], Mueen and Keogh defined a motif as a pair of points (P_i, P_j), $(i < j)$ which are closest to each other according to the Euclidean distance. We extend this notion and define the *top-k* motifs as the set of k pairs such that they are amongst the *top-k* closest pairs according to the Euclidean distance. Sometimes we will require that points in a *top-k* pair are non-overlapping, i.e. $j - i \geq m$ [56].

Example 4. *For instance, assume $m = 2$, $k = 3$ and $w = 6$, Figure 3-3.a shows a set of 2-dimensional points. The current window contains 6 points P_1—P_6 and the* top-k *motifs are* $\{(P_1, P_2), (P_5, P_6), (P_1, P_3)\}$. *Consider Figure 3-3.d where a new point P_7 enters the time*

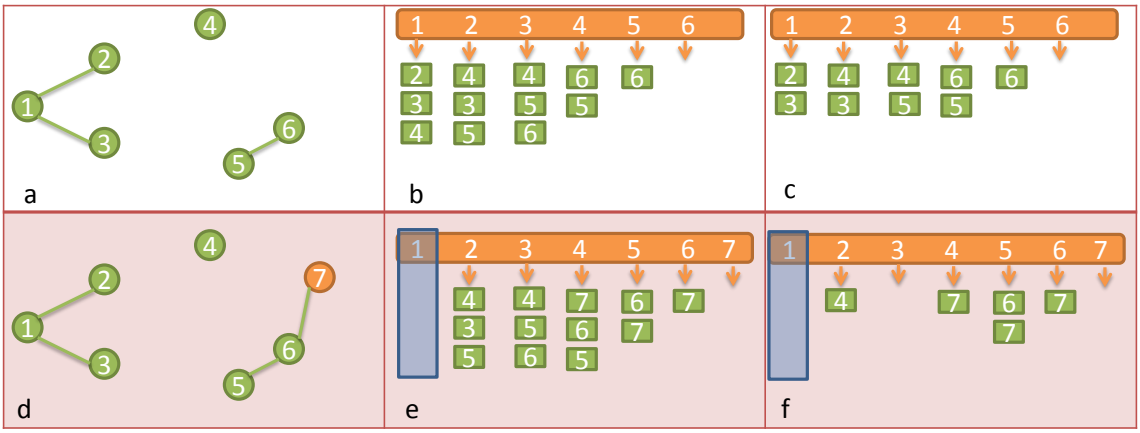


Figure 3-3: A toy example shows a set of 6 2-dimensional points and the data summaries used in algorithm *nMotif* and *kMotif* when $k = 3$ and $w = 6$.

series. The window is shifted one point and consists of points P_2—P_7. *Since* P_1 *is expired, we get a new set of* top-k *motifs, which is* $\{(P_5, P_6), (P_6, P_7), (P_2, P_4)\}$. *Notice that in this toy example for simplicity we did not consider the constraint that points* (P_i, P_j) *in a motif pair should be non-overlapping* $(j - i \geq m)$. *If the overlapping constraint is considered, the top-*k *motifs in Figure 3-3.a become* $\{(P_1, P_3), (P_2, P_4), (P_4, P_6)\}$.

The top-k most similar motifs problem is defined as follows:

Definition 3 (top-k Motifs Problem)**.** *Given a time series* S_t, *a window length* w, *a motif length* m *and a parameter* k, *at any time point* t, *maintain a summary of the time series from which we can answer the query for the top-k motifs exactly.*

The *Top-k Motifs problem* is a generalization of the *Exact Motif Discovery Problem* of Mueen and Keogh [56], which corresponds to the case $k = 1$. We propose space-efficient algorithms for the generalized problem. First, we will derive a memory lower-bound for any exact algorithm solving the top-k motifs problem. Next, we show that the given lower-bound is tight by showing a trivial algorithm that consumes an amount of memory deviating only a constant factor from the lower-bound. This algorithm, however, does not scale up with large-scale applications monitoring thousands of time series, such as, e.g. in sensor network. Therefore, we will propose other algorithms which are almost as space-efficient and achieve much better update times, and even outperform *oMotif* when k is set to 1.

The comparison of the theoretical complexity of the algorithms is summarized in Table 3.2. In all cells of the table we show the worst case complexity of the algorithms except the

places where we denote *avg.* for on average and *amo.* for amortized. Notice that in this table we show the space complexity in term of the number of floats while the *lower-bound* in lemma 8 is based on the number bits.

3.4 Memory Lower-bound for any exact and deterministic algorithm

In this section, we will derive a lower-bound on the memory requirement in order to summarize the time series and answer the top-k motifs query exactly. We further show that this lower-bound is tight by introducing several algorithms achieving the given bound.

Memory Lower-bound

Deriving the memory lower-bound introduces an insight into a research problem. Sometimes this step is very helpful in designing an optimal algorithm. The worst-case memory lower-bound of any exact algorithm solving the top-k similar motifs discovery problem is derived as follows:

Lemma 8 (Lower-bound)**.** *Given a time series S_t, a window of size w, a motif length m and a parameter k. Any algorithm being able to answer the top-k motifs query exactly at any time point requires at least $\Omega(w \log(w))$ bits to summarize the time series.*

Proof. We will prove the lemma in the special case when $k = 1$ from which the correctness of the lemma with $k > 1$ will automatically hold. Consider a time series of length $w - 1$: $S_{w-1} := r_1 r_2 \dots r_{w-1}$ in which all r_i are distinct natural numbers and $1 \leq r_i \leq w - 1$. We will show that a summary of S_{w-1} denoted as SUM_{w-1} must allow for reconstructing perfectly all $w - 1$ entries of S_{w-1}. Notice that since the window length is $W = w + m - 1$

Table 3.2: Theoretical Complexity Comparison

	Alg.	**Space (floats)**	**Time per Update**
$k = 1$	*oMotif*	$\mathbf{O}(w^{\frac{3}{2}})$ amo.	$\mathbf{O}(wm)$
	nMotif	$\mathbf{O}(w)$	$\mathbf{O}(wm)$
$k > 1$	*nMotif*	$\mathbf{O}(kw)$	$\mathbf{O}(wm + w \log k)$
	kMotif	$\mathbf{O}(w)$ avg.	$\mathbf{O}(wm + \lvert V \rvert (k + \log \lvert V \rvert))$ *

*V denotes the so-called *promising points* and has size $2k \log(w)$ on average

when we extend S_{w-1} with a sequence of m values $Y = y_1, y_2, \cdots, y_m$ no element of S_{w-1} is expired.

As the summary of S_{w-1} represents the full state of the machine recording the stream, the summary SUM_{w+m-1} of $S_{w+m-1} := r_1 \dots r_{w-1} y_1 \dots y_m$ only depends on SUM_{w-1} and $Y = y_1, \dots, y_m$. Furthermore, SUM_{w+m-1} contains enough information for answering the top-1 query for S_{w+m-1} and the SUM_{w-1} does not depend on Y.

The following procedure will reconstruct the sequence S_{w-1} from the summary SUM_{w-1}: Let $Y = y_1, \dots, y_m$ be any subsequence with length m where $1 \leq r_i \leq w-1$. Assume now that $y_1 = r_i$, $y_2 = r_{i+1}$, $\dots$, $y_m = r_{i+m-1}$ for some $0 < i < w - m$, then the answer to the top-1 query for S_{w+m-1} will correspond to the subsequences $r_i \dots r_{i+m-1}$ and $y_1 \dots y_m$ with distance 0. Since Y is an arbitrary sequence with length m we can choose Y to reconstruct S_{w-1} perfectly from the summary SUM_{w-1}.

The aforementioned procedure does not produce two different sequences when it is given the same summary SUM_{w-1} at the input. There are totally $(w-1)!$ different subsequences $S_{w-1} := r_1 r_2 \dots r_{w-1}$, therefore, there must be at least the same number of corresponding different summaries. According to information theory, storing $(w-1)!$ different sequences requires at least $(w-1)\log(w-1)$ bits, this proves the lemma. □

Let us consider a very simple approach achieving the lower-bound in lemma 8. The algorithm stores the entire window as the summary which requires $\mathbf{O}(wb)$ bits where b is the maximal number of bits representing any float value r_i in the stream. The value of b can be arbitrarily large depending on how large r_i are. Let us make an optimistic assumption that b is upper-bounded by $\log(w)$, the number of bits minimally needed to store w different numbers. Consequently, the summary is as large as $w \log(w)$ bits, matching the lower-bound of Lemma 8. The lemma actually shows that there is no hope w.r.t. memory consumption, of doing better than just storing the complete window. The top-k query processing for such a summary, however, is very time demanding as we need to compute the Euclidean distances between all pairs of points, which results in a quadratic complexity.

Therefore, the given approach is impractical for online applications which require not only a compact summary but also a quick response time to the query. The naive algorithm described in the following subsection will take into account both time and space efficiency.

Algorithm 3 nMotif(w, m, k)

```
1: L ⟵ ∅
2: while new point P_t arrives do
3:    L.PopFront()
4:    L_t ← ∅
5:    L.PushBack(L_t)
6:    for i = t − w + 1 to t-1 do
7:       Let P_l be the last element of L_i
8:       upper-bound← d^2(P_i, P_l)
9:       d ← distance(P_t, P_i,upper-bound)
10:      if d < d(P_i, P_l) then
11:         Insert P_t to list L_i such that L_i is sorted in ascending order according to the
            distance to P_i
12:         if L_i.Size()> k then
13:            L_i.PopBack()
14:         end if
15:      end if
16:   end for
17: end while
```

Algorithm 4 distance(X, Y,upper-bound)

```
1: i ⟵ 0
2: sum ⟵ 0
3: while sum < upper-bound and i < m do
4:    sum ← sum +(X[i] − Y[i])^2
5:    i = i + 1
6: end while
7: return sqrt(sum)
```

A Naive Algorithm

An online motif discovery algorithm should consider the trade-off between memory usage and processing time. As time series data arrives continuously and unpredictably, summary update should be done as fast as possible while the summary should be as small as possible. In addition to that the query time also should be fast for online decision. However, usually it is the case that we have to trade in processing time for smaller space usage and vice versa. We will now first introduce the *nMotif* algorithm, which has similarities with the *oMotif* algorithm by Mueen and Keogh [56]. Nevertheless, we will show both theoretically and empirically that *nMotif* with k set to 1 outperforms *oMotif* w.r.t. both space usage and query performance. Then we will further improve the *nMotif* algorithm to arrive at the more efficient *kMotif* algorithm for any $k > 1$.

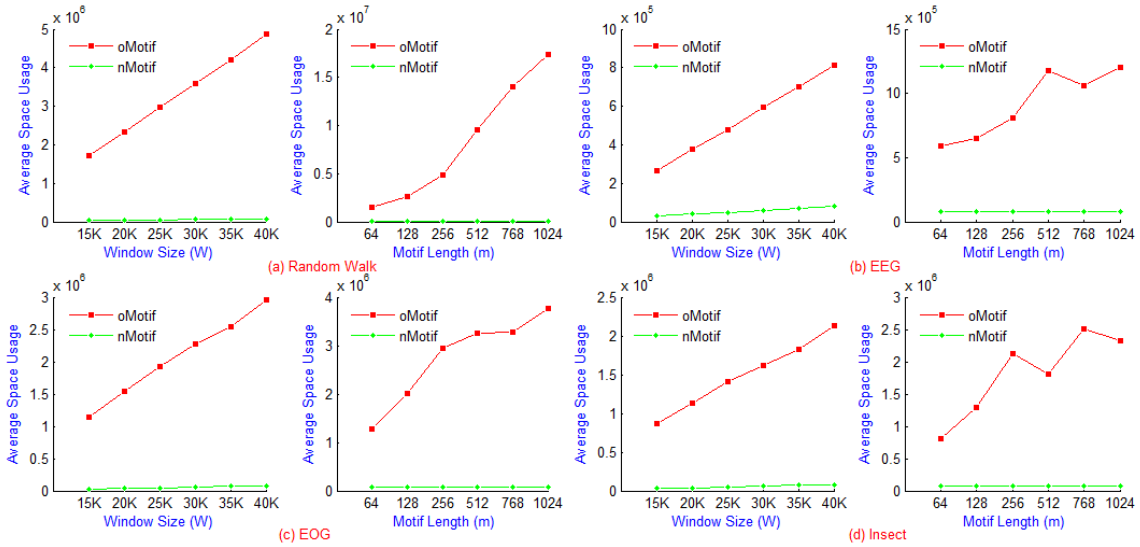


Figure 3-4: A comparison of the average space usage of two algorithms nMotif and oMotif in the special case $k = 1$.

The *oMotif* algorithm of Mueen and Keogh [56] roughly works as follows: instead of only storing the window, also for every point in the window its so-called *backward nearest neighbor list* (BNN) is stored. The BNN of a point P_i stores the time-stamp of all points with an earlier timestamp than P_i, ordered w.r.t. increasing distance to P_i. When a new point enters the window and hence the oldest point of the window is discarded, the lists are updated. Mueen and Keogh [56] discuss several optimizations and show that the size of the BNNs is $\sqrt{w}$ with an amortized analysis, leading to a summary of size $w^{3/2}$. For details we refer to [56]. In contrast to *oMotif*, our *nMotif* algorithm will store a *forward top-k nearest neighbor list* instead of a BNN. This seemingly trivial difference, however, has huge implications, as we will see.

The pseudo-code in Algorithm 3 shows how *nMotif* works. It maintains a list $\mathcal{L} = L_{t-w+1}L_{t-w+2}\cdots L_t$ where each entry L_i corresponds to a point P_i in the current time series window (lines 3-5). Each list L_i contains the set of the k points closest to P_i among the points with later time-stamp than i. For this reason we call L_i a *forward top-k nearest neighbor list*. In order to update these forward nearest neighbor lists when a new point P_t arrives we need to calculate the distance from P_t to any P_i (lines 8-9). Subsequently, P_t is inserted into the forward nearest neighbor list of P_i if P_t is closer to P_i than point P_l is, where P_l is the furthest point to P_i in the current list L_i (lines 10-15).

Example 5. *For instance, Figure 3-3.b shows this data structure for the points in Figure*

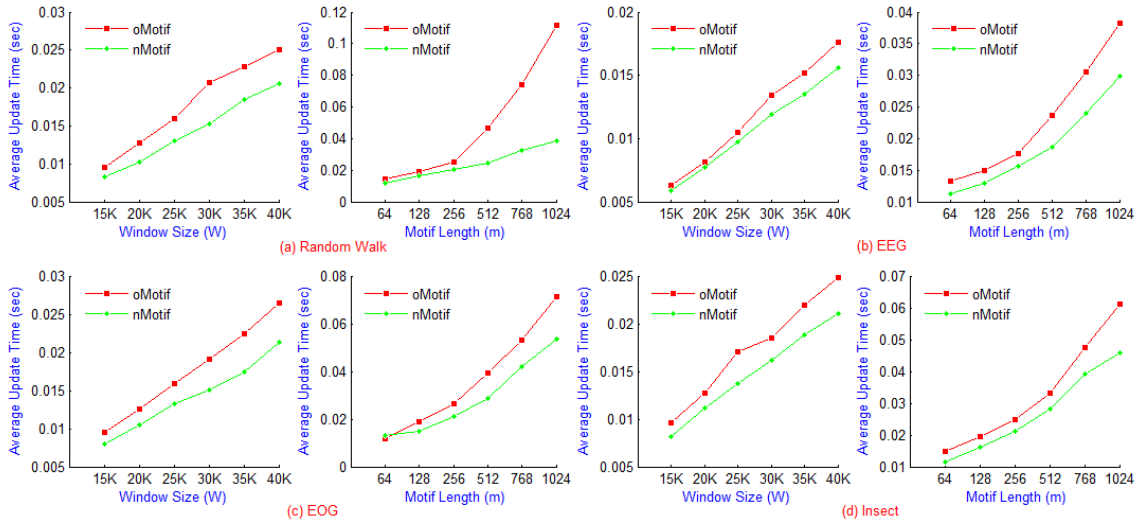


Figure 3-5: A comparison of the average update time of two algorithms nMotif and oMotif in the special case $k = 1$.

3-3.a for $k = 3$. Corresponding to point P_1, the forward nearest neighbors are P_2, P_3 and P_4 sorted in ascending order according to their distance to P_1. Similarly, the forward nearest neighbor list associated with P_2 contains P_4, P_3 and P_5 in that order. Recall that each list L_i associated with point P_i only contains the forward nearest neighbors. Therefore, although point P_1 is the nearest neighbor of point P_2 it is not contained in L_2 because P_1 appears earlier than P_2 in the time series. Figure 3-3.e shows how the data structure is updated when P_1 is expired and point P_7 arrives. In particular, we delete the entry corresponding to point P_1 because it has been expired. Then, the Euclidean distance from P_7 to every point P_i $(1 < i < 7)$ is calculated and the forward nearest neighbor list L_i is updated subsequently.

Euclidean distance calculation: when the motif length is large, the update time is dominated by the Euclidean distance calculation process. In order to reduce the computational effort of this process an early termination technique was introduced in Mueen and Keogh [56]. The pseudo-code describing the Euclidean distance computation with early termination is shown in Algorithm 4. An extra parameter *upper-bound* is passed to the algorithm, reflecting the current "distance to beat" to be added in the forward top-k nearest neighbor list of X. As soon as in the distance computation it becomes clear that the distance between X and Y will exceed this bound, the distance computation is aborted.

Given the forward nearest neighbor data structure, the query phase is performed by first sorting all the pairs of points resident in the forward nearest neighbor structure in

the ascending order according to Euclidean distance. The top-k pairs whose Euclidean distances are shortest are returned as the answer to the top-k motifs query.

Lemma 9 (Correctness of *nMotif*)**.** *In the forward nearest neighbor data structure handled by the algorithm nMotif, we can find the true top-k motifs.*

Comparing *nMotif* to *oMotif*

First recall that *oMotif* only finds the *top-1* motif in the current window. Therefore, *oMotif* should be compared to *nMotif* with parameter k set to 1. Concerning the space consumption, *nMotif* uses $\mathbf{O}(kw)$ float numbers in order to store the entire forward nearest neighbor structure, which is much better than the $\mathbf{O}(w^2)$ float numbers in the worst case and $\mathbf{O}(w^{\frac{3}{2}})$ float numbers in the amortized analysis needed by *oMotif*. Furthermore, every update operation of algorithm *nMotif* consists of the Euclidean distance computation cost ($\mathbf{O}(wm)$) and the summary update cost ($\mathbf{O}(w\log(k))$). Overall, the aggregated cost for each update operation is $\mathbf{O}(w(m+\log(k)))$ which is comparable to that of *oMotif* when $k=1$. Hence, besides being more space-efficient *nMotif* is theoretically as time-efficient as the *oMotif* approach.

In order to illustrate the significance of our algorithm with $k=1$ in comparison to algorithm *oMotif*, we plot the time and space consumption of the two algorithms in Figure 3-4 and Figure 3-5. In particular, Figure 3-4 compares the average space usage of two algorithms in terms of the number of elements stored in the data summary plus the window size. The comparison is done on 4 different datasets (see section 3.6 for the description of the datasets). We observe in all datasets that *nMotif* is significantly more space-efficient than *oMotif*. Besides, Figure 3-5 shows that *nMotif* is faster than *oMotif* although theoretical analysis shows that two algorithms are comparable in term of update time.

The reason for this is two fold. First, the algorithm *nMotif* uses tighter bounds in the Euclidian distance computation, since there are less points in the summary. Secondly, the summary of *nMotif* is much smaller allowing more efficient access and update. This analysis shows how a small change to the algorithm—maintaining the forward nearest neighbor instead of all backward nearest neighbors—has a huge impact on the efficiency of the algorithm. In the following we will only compare to *nMotif* as it extends to arbitrary values of k and outperforms *oMotif* anyway.

3.5 A Space Efficient Approach

In this section, we introduce a space-efficient algorithm. The new algorithm is called *kMotif* and uses $\mathbf{O}(w)$ float numbers on average. Compared to *nMotif*, algorithm *kMotif* is about k times more space-efficient on average, and has, in the worst case, still the same space and similar time complexity.

Algorithm *kMotif*

We describe the main idea behind *kMotif* with a simple example. First let us define a pair of points (P_i, P_j) with $(i < j)$ a *promising pair* if it belongs to the top-k motifs of the window starting from point P_i. In the *kMotif* algorithm, we will store for every point P_i not its complete forward top-k nearest neighbor list, but instead only those timestamps j such that (P_i, P_j) forms a promising pair. This list is always a subset of the list maintained by *nMotif*.

For example, consider again the illustration in Figure 3-3.a. In the window starting from P_1 and finishing at point P_6 the *top-3* motifs are (P_1, P_2), (P_1, P_3) and (P_5, P_6). Thus, in the forward nearest neighbor list of point P_1 (Figure 3-3.b), P_4 is redundant because (P_1, P_4) is not a promising pair. This pair will never become a *top-3* motif in any sliding window starting earlier than P_1. As a result, we can safely remove point P_4 from the forward nearest neighbor list of point P_1 without any effect on the accuracy of the algorithm.

Similarly, if we consider the window starting from P_2 and finishing at point P_6, the pairs (P_2, P_4), (P_2, P_3) and (P_5, P_6) are the *top-3* motifs. Hence, point P_5 can be excluded from the forward nearest neighbor list of P_2. If we repeat this action for the other windows we will end up with a very compact list shown in Figure 3-3.c. This list has an important property that only promising pairs are present in it. Therefore, we call this data structure the *promising neighbors structure*.

The pseudo-code in Algorithm 5 describes how *kMotif* works. It always maintains and updates promising neighbors lists $\mathcal{L}$. In particular, when a new point P_t occurs, we traverse $\mathcal{L}$ backwards from the end to the beginning to check whether all pairs remain promising and remove unpromising ones (lines 7-21).

Lines 8-15 check whether an existing pair (P_l, P_i), where $P_l \in L_i$, is promising. In particular, we need to compare the value $d(P_l, P_i)$ to the distance between the current k^{th}

Algorithm 5 kMotif(w, m, k)

```
1: L ⟵ ∅
2: while New point P_t arrives do
3:    L.PopFront()
4:    L_t ← ∅
5:    L.PushBack(L_t)
6:    Let V be an empty dequeue
7:    for i = t − 1 to t − w + 1 do
8:       while P_l ≠ nil do
9:          P_l ⟵ L_i.next()
10:         if d(P_l, P_i) < V[k − 1] then
11:            V.insert(d(P_l, P_i))
12:         else
13:            L_i.delete(P_l)
14:         end if
15:      end while
16:      d(P_t, P_i) = distance(P_t, P_i, V[k − 1])
17:      if d(P_t, P_i) < V[k − 1] then
18:         V.insert(d(P_t, P_i))
19:         L_i.insert(P_t)
20:      end if
21:   end for
22: end while
```

closest pair, e.g $V[k-1]$ (line 10). If (P_l, P_i) is promising, i.e. $d(P_l, P_i) < V[k-1]$ the value $d(P_l, P_i)$ is inserted into a sorted dequeue V (line 11). Otherwise, we remove P_l from L_i (line 13).

We perform the same check with every incoming pair (P_t, P_i) in lines 16-20. First we calculate the Euclidean distance from P_t to P_i (line 16). In the next step we check if (P_t, P_i) is promising and insert P_t into the sorted promising list L_i (lines 17-20). In doing so, the summary update phase costs about $\mathbf{O}(|V|(k + log|V|))$ in the worst case. Notice that in order to guarantee the correctness of the algorithm the insert and delete element functions are designed such that the list is always sorted in ascending order according to the Euclidean distance.

Lemma 10 (Correctness of *kMotif*)**.** *Using the promising data structure handled by algorithm kMotif, we can find the true top-k motifs. Besides, the promising summary is at most as large as the forward nearest neighbor adopted by nMotif algorithm.*

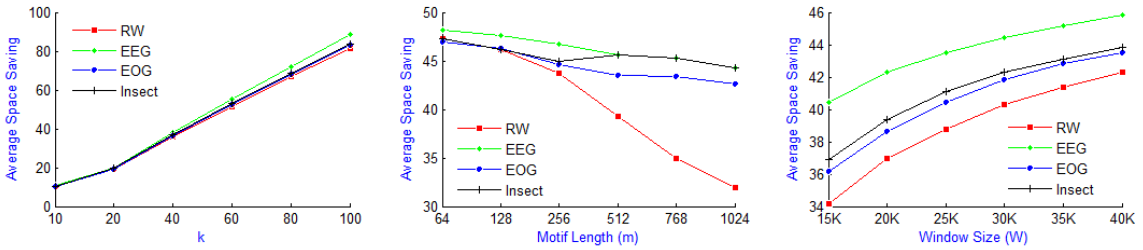


Figure 3-6: A comparison of *kMotif* and *nMotif* in terms of space saving. The space saving ratio is measured as the fraction between memory usage of algorithm *nMotif* and algorithm *kMotif*. Algorithm *kMotif* is 30-50 times more space-efficient than algorithm *nMotif* when $k = 50$.

Complexity of *kMotif*

The working space of algorithm *kMotif* consists of a portion of memory storing the window and another one containing the lists $\mathcal{L}$ and V. The former portion requires $\mathbf{O}(w)$ float numbers. The size of the latter portion depends on the number of promising pairs after every summary update. We consider this number as a random variable X. The following lemma shows how large X is on average :

Lemma 11 (Average Memory)**.** *Given a window, assuming that any pair* (P_i, P_j) *has the same opportunity being in the top-k closest pairs of that window, the average number of promising pairs at any time point is* $E(X) \simeq 2k \log(w)$

Proof. Let $X_{ij}(j > i)$ be the indicator of the event that (P_i, P_j) is a promising pair. The probability of $X_{ij} = 1$ is equal to the probability of (P_i, P_j) is a top-k closest pairs among $\frac{(w-i)(w-i+1)}{2}$ pairs of points in this window if $\frac{(w-i)(w-i+1)}{2} > k$ and is equal to 1 otherwise. Since we assume that this probability is the same for any pair in the window we have:

$$Pr(X_{ij} = 1) = \begin{cases} \frac{2k}{(w-i)(w-i+1)} & if \ \frac{2k}{(w-i)(w-i+1)} < 1 \\ 1 & otherwise \end{cases}$$

On the other hand, as X stands for the number of promising pairs in list L we get:

$$X = \sum_{i=1}^{w} \sum_{j=i+1}^{w} X_{ij} \tag{3.1}$$

from which we further imply that:

$$E(X) = \sum_{i=1}^{w} \sum_{j=i+1}^{w} E(X_{ij}) \tag{3.2}$$

$$\simeq \sum_{i=1}^{w} \frac{2k}{(w-i+1)} \tag{3.3}$$

$$\simeq 2k \log w \tag{3.4}$$

The last equation proves the lemma. □

It is important to note that in this analysis we allow overlapping motifs. However, a similar result $E(X) \simeq 2k \log(w-m)$ with non-overlapping motifs can be proved similarly. Generally, the expected number of promising pairs is small compared to the size of the window w. We will see that for most of the datasets in our experiments the number of promising pairs does not deviate too far from this theoretical average value. Let us denote this theoretical average value as $E_p = 2k \log(w)$. We can theoretically bound the probability of the event that the number of promising pairs deviating from E_p by a constant factor as follows:

Lemma 12 (By Markov's inequality)**.** *Given a constant a, the probability that the number of promising pairs deviating from its expected value by the constant factor a is bounded by* $Pr(X > aE_p) \leq \frac{1}{a}$

The aforementioned bound is naive but it is useful because there is no condition on the independence of X_{ij}. However, it is not tight since we can further tighten the bound based on the assumption that X_{ij}'s are independent from each other. In particular, we have the following result:

Lemma 13 (By the Chernoff's bound)**.** *If we assume that* X_{ij} *are independent from each other we have:* $Pr(X > eE_p) \leq \frac{1}{e^{E_p}}$

Proof. According to the Chernoff's bound we have:

$$Pr(X > (1+\delta)\mu) < \left(\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}} \right)^{\mu} \tag{3.5}$$

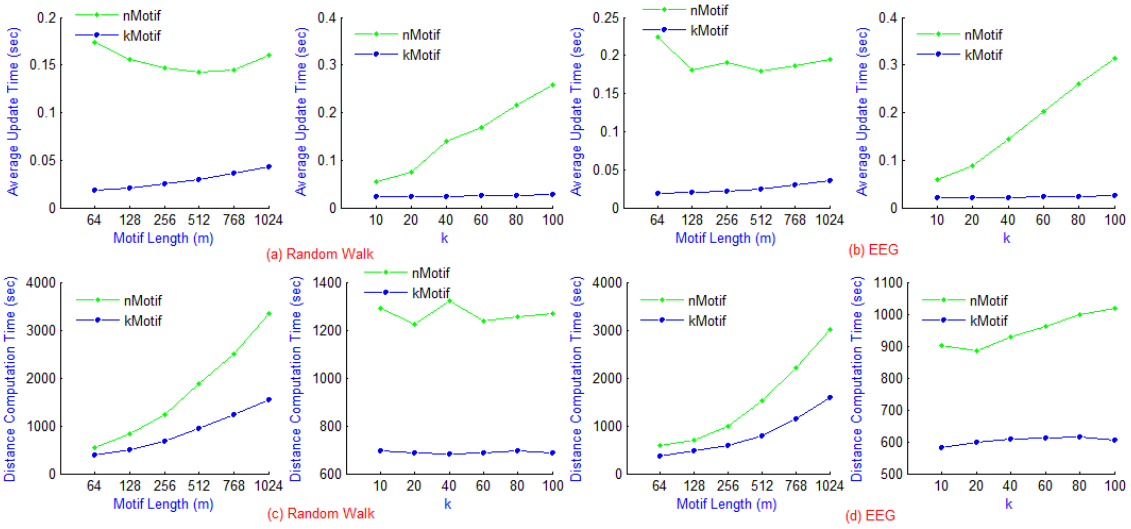


Figure 3-7: Update time and Euclidean distance calculation time for *kMotif* and *nMotif*.

If we replace the value of δ by $e-1$ and the value of μ by E_p then we have:

$$Pr(X > eE_p) < \left(\frac{e^{(e-1)}}{e^e}\right)^{E_p} \quad (3.6)$$

$$< \frac{1}{e^{E_p}} \quad (3.7)$$

The last inequality proves the lemma. □

Theoretically, Lemma 13 shows a tighter upper-bound on the probability of the event that the number of promising pair deviates far from E_p with a constant factor e. That probability is small, i.e. about 10^{-44} when k is in order of hundreds.

Space usage: *kMotif* vs. *nMotif*

In summary, algorithm *kMotif* consumes $\Omega(w + 2k\log(w))$ float numbers on average which is k times more space-efficient than algorithm *nMotif*. Another interesting property of algorithm *kMotif* is that although it seems to be theoretically less time-efficient than *nMotif* ($\mathbf{O}(wm + |V|(k + \log|V|)$ versus $\mathbf{O}(w(m + \log(k)))$, where $|V| = 2klogw$ on average. However, in practise it turns out to be faster in all experiments in both real-life and synthetic datasets. The reason of this effect will be discussed in the experimental section.

3.6 Experiments and Results

We implemented the algorithms *kMotif* and *nMotif* in C++ with the STL library. We ran all the programs on a 2.4 GHz core 2 dual Windows platform with 2GB of RAM. The datasets we use in the experiments are the same as in [56], including 3 real-life time series *EEG, EOG*, the *Insect* trace data and a synthetic dataset *random walk*. In all experiments Z-normalization is applied to all vectors and overlapping motifs are not allowed.

The source code with demo and related resources of our work are available for download on our project website[2]. It is also worth noting that in this particular section we aim at discussing the significance of *kMotif* in comparison to *nMotif*. The significance of *nMotif* compared to *oMotif* is already discussed in section 3.4 where it was shown that *nMotif* outperforms *oMotif*.

In the first experiment, we fix $k = 50$ and $W = 50000$ while m is varied from 64 to 1024. For each value of m the ratio between the working space of *nMotif* and *kMotif* is calculated. The larger the ratio is, the more effective algorithm *kMotif* is. Figure 3-6 in the middle shows the result of this experiment on 4 datasets. On all datasets *kMotif* achieves 30 to 50 times space-saving as compared to *nMotif*. Another important observation is that the space-saving slightly degrades when m is large. The explanation of this effect will be discussed later.

Similarly, we carry out another experiment with fixing $m = 256$ and $W = 50000$ and varying the value of k. Figure 3-6 on the left shows the dependency of space-saving ratio on k. Four lines corresponding to the 4 datasets follow the following rule: the space-saving ratio increases linearly with the value of k. This result is consistent with our theoretical analysis in section 3.5 stating that when m and w are fixed the space-saving ratio is approximately equal to k. Finally, Figure 3-6 on the right illustrates the result when $m = 256$ and $k = 50$ are fixed while W is varied. We observe that the space-saving ratio is steadily increasing from 30 up to 50 along with the window length in this experiment.

Update time: *kMotif* versus *nMotif*

We plot the update time and the Euclidean distance calculation time of *kMotif* and *nMotif* in Figure 3-7. It is clear from Figure 3-7.a and 3-7.b that *kMotif* is steadily several times

[2]http://www.win.tue.nl/∼lamthuy/projects/kmotif.html

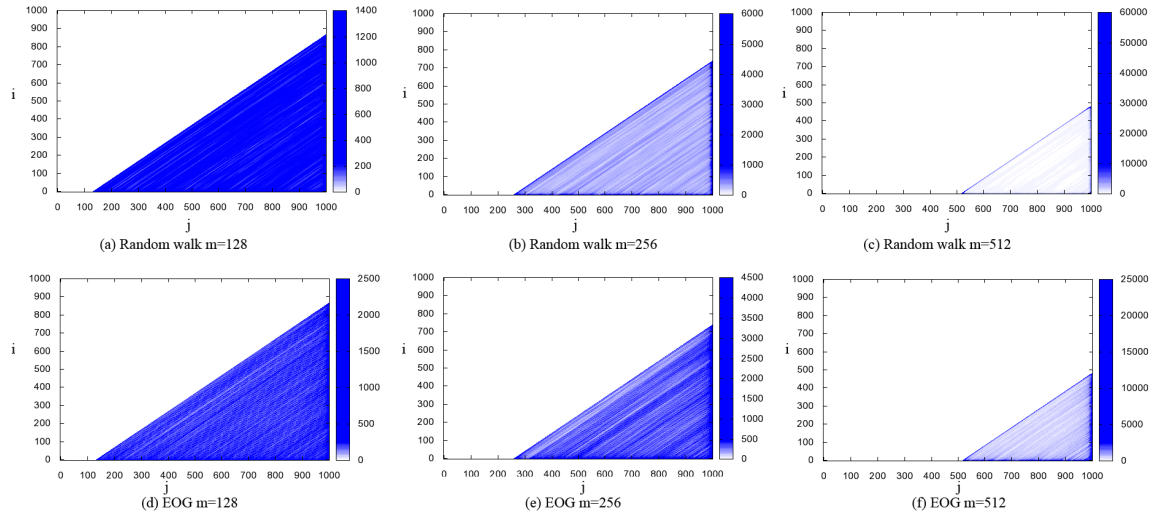


Figure 3-8: Density of the event (P_i, P_j) being a top-k motifs where i, j are the time-stamps at which the points start relative to the beginning of the window. When m is large parts close to the triangle edges are more dense.

faster than *nMotif*. One of the reasons for the speed-up is explained in Figure 3-7.c and Figure 3-7.d in which the Euclidean distance calculation times are compared. *kMotif* adopts tighter upper-bounds in the distance calculation function leading to earlier termination of the calculation process. As a result, the summary update phase is more efficient. For brevity reason, the result is illustrated with only two datasets but similar behavior is observed with the other ones.

Distribution of promising pairs

Recall that in order to derive the average number of promising pairs we had to assume a prior knowledge about the distribution of X_{ij}. In this section, we will discuss the validity of the assumption that we use in lemma 11. We first analyze how far the number of promising pairs deviates from the theoretical average value $E_p = 2k\log(w)$. In order to do so, the number of promising pairs in each window is divided by the theoretical expectation E_p to obtain the ratio which we call the *deviation ratio*. When we measure the deviation ratio for various values of k, m and w, we observe that it mostly depends on m. Therefore, we plot the deviation ratio when k and w are fixed to 100 and 1000 respectively and m varies from 128 to 512 in Figure 3-9.

Each portion of this figure corresponds to a dataset and the deviation ratios are plotted for 3 values of m corresponding to three lines $m = 128$, $m = 256$ and $m = 512$. The first

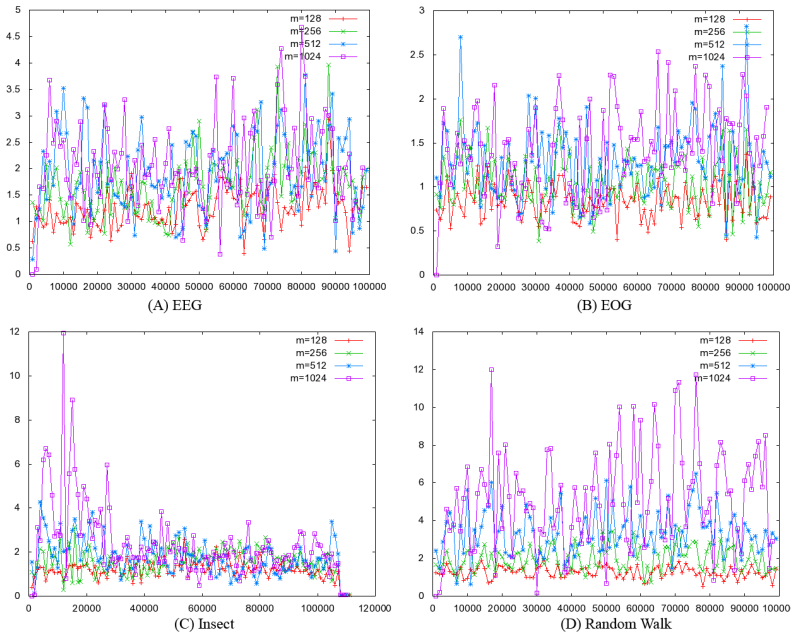


Figure 3-9: The ratio between the number of promising pairs and the theoretical average number of promising pairs E_p. Picture looks better in color.

behavior we can observe in all datasets is that the number of promising pairs is not deviated very far from the theoretical expectation E_p. In this particular setting a constant factor of 12 is an upper-bound for the deviation ratio.

Another important observation is that the variance of the deviation ratio in creases with increasing m. This explains the role of m in the degradation of the space-saving ratio as shown in subsection 3.5. In order to get an insight into the reason of this behavior we plot the distribution of the event (P_i, P_j) being a top-k motif in Figure 3-8 where i, j are the offsets of the time-stamps from the beginning of the window. In particular, we fix the value of $k = 100$ and $w = 1000$ and vary m as 128, 256 and 512. Then we count the number of times (P_i, P_j) is a top-k motif in the window of length w. The plot shows the result of only two datasets but similar behavior was observed in the other datasets.

Since we are only interested in non-overlapping motifs ($j - i \geq m$) only the right bottom triangle is dense. If the assumption in lemma 11 is true we must observe the same density at any point in this triangle. Actually, this is the case when m is small; cfr. Figures 3-8.a and 3-8.d. However, when m is large the *kMotif* algorithm favors the points close to the triangle's edges causing an interesting effect which can be observed in Figures 3-8.b.c.e.f. The areas closed to the triangle's edges are more dense.

Currently we do not have a satisfactory explanation for this observation. Because of the observed anomaly, when m becomes large the performance of *kMotif* can slightly degrade. Nevertheless, as we have seen in the experiments, when m is as large as 1024, *kMotif* still preserves very high space saving ratio (at least 30 times when $k = 50$). Besides, in many applications with multi-dimensional time series the typical dimensionality is only in order of hundreds which remains in the range where *kMotif* exposes its maximum performance.

Chapter 4

Mining Large Tiles in a Stream

Large tiles in a database are itemsets with the largest *area* which is defined as the itemset frequency in the database multiplied by its size. Mining these large tiles is an important pattern mining problem since tiles with a large area describe a large part of the database. In this chapter, we introduce the problem of mining the top-k largest tiles in a data stream under the sliding window model[1] . The straightforward approach for this problem is to recalculate the top-k largest tiles from scratch every time the window is sliding. However, the problem of mining the largest tile was shown to be NP-hard and inapproximable which makes this method inefficient. To tackle this problem, we thus propose a candidate-based approach which summarizes the data stream and produces the top-k largest tiles more efficiently than the straightforward solution for moderate window size. We also propose an approximation algorithm with theoretical bounds on the error rate to cope with large size windows. In the experiments with two real-life datasets, the approximation algorithm is up to a hundred times faster than the candidate-based solution and the baseline algorithms created based on the state-of-the-art solutions for the top-k largest tiles mining. Moreover, the proposed algorithm is very accurate in producing the top-k largest tiles with negligible false negative and false positive rate on real-life datasets.

[1]part of this chapter was published as Toon Calders, Elisa Fromont, Baptiste Jeudy and Hoang Thanh Lam. Analysis of Videos using Tile Mining. In: RealStream workshop at ECML PKDD 2013, and in Hoang Thanh Lam, Wenjie Pei, Adriana Prado, Baptiste Jeudy, Élisa Fromont, and Toon Calders. Extraction des top-k plus grandes tuiles dans un flux de données. CAP 2013

4.1 Introduction

Mining frequent patterns is an important research topic in data mining. However, instead of focusing on exhaustive search to find all possible frequent patterns, many works are now focusing on designing methods that are not only efficient in the context of very big data but also limit, e.g. with constraints or with new interestingness measures, the number of patterns output by these algorithms.

Area is a measure of pattern interestingness defined as a pattern's frequency in the database multiplied by its size. It has been shown that in some applications such as in role mining [44, 72] where the idea is, given a set of users and a set of permissions, to find a minimum set of roles such that all users will be assigned a role for which some permissions will be granted. Mining roles is equivalent to a variant of mining itemsets with large area in a database [44, 72].

Recent applications produce a large variety of transactional data streams, such as text stream from *twitter*[2] or video stream in which video frames can be converted into transactions [23]. In the context of data streams, data usually arrive continuously with high speed, hence requiring efficient mining techniques for summarizing the data stream and keeping track of important patterns over time.

In this work we tackle the problem of mining the top-k largest tiles, i.e. the k closed itemsets with the largest area in a stream of itemsets. The problem of mining the largest tile in a database is well-known to be NP-hard and inapproximable [27]. Therefore, the straightforward approach that recalculates the set of top-k largest tiles from scratch every time a sliding window is updated is not efficient. To deal with this situation, we first introduce in section 4.4 a candidate-based approach which incrementally summarizes the stream by keeping the itemsets that can be the top-k largest tiles in some future windows. For each candidate, an *upper-bound* and a *lower-bound* of the area in any future window are kept. These bounds are used to prune the candidates when they cannot be the top-k largest tiles in any future window. In doing so, the candidate-based algorithm is more efficient than the straightforward approach because updating is cheaper than recalculating the top-k tiles from scratch.

However, when the widow size is large, the candidate-based algorithm is inefficient be-

[2] `www.twitter.com`

cause the summary grows very quickly. Therefore, we introduce an approximation algorithm with theoretical bounds on the error rate. In the experiments with two real-life datasets presented in section 4.6, the approximation algorithm is up to a hundred times faster than the candidate-based solution and the baseline algorithms created based on the state-of-the-art solutions for the top-k largest tiles mining. Moreover, it is very accurate in producing the top-k largest tiles. In the experiment, we also discuss a demo of a real-life use-case with a text stream to show potential applications of the problem.

4.2 Related work

Recent works on itemset mining [3, 60, 75] propose approaches to solve the redundancy and trivial patterns issues in frequent pattern mining. For instance, the first two works focus on finding a concise representation of the set of frequent patterns. Meanwhile, the latter work solves the aforementioned issues by proposing a compression-based approach that finds patterns compressing the database well. In all these cases as well as in ours, the purpose is to limit the output of the algorithms to a set of useful patterns.

The problem of mining large tiles has already been tackled in [44, 72]. The authors of these works showed that the problem of finding roles is equivalent to variants of the tiling database problem. The author of [4] also showed that the dense rectangle in a binary matrix seems to correspond to interesting concepts. Therefore, mining large tiles may help to identify those interesting concepts from the databases. In [70] the authors work on the problem of how to output the set of tiles in a tree representation which is easily interpretable by the users.

In many applications, data arrives in a streaming fashion with high speed [1]. Besides the popularity of data streams in many applications, the temporal aspect of the patterns such as the evolution of patterns overtime [1] and the periodicity of the patterns [42] provides useful insights about the data for the users. Despite the importance of the data stream paradigm, there is no work yet addressing the problem of mining large tiles in a data stream. The algorithms introduced in this work are, to the best of our knowledge, the first to solve the problem of mining the top-k largest tiles in a stream of itemsets under the sliding window model.

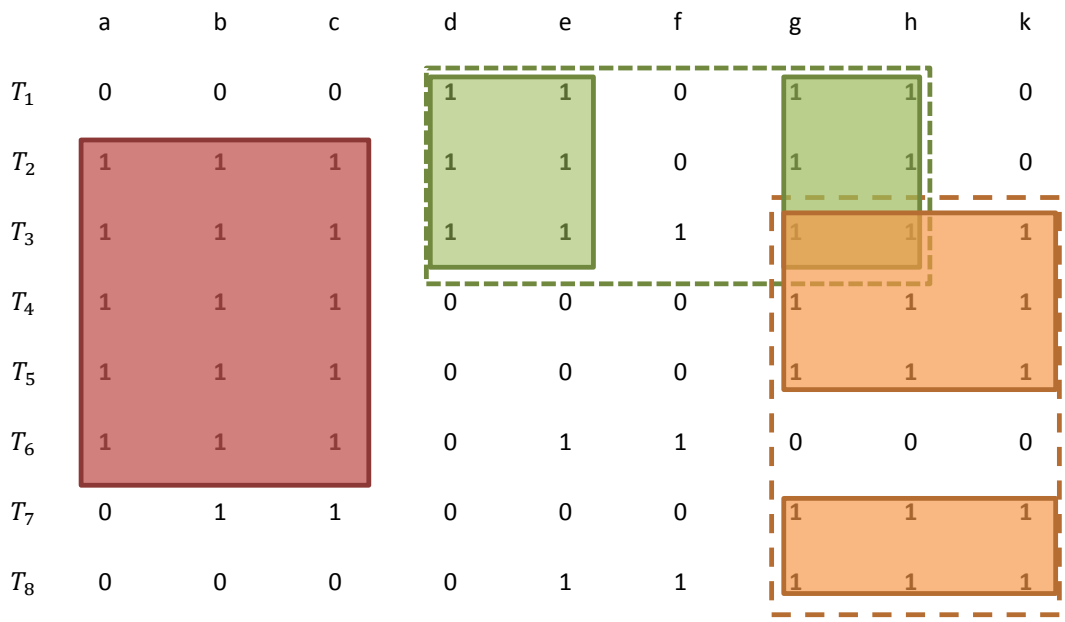


Figure 4-1: An example of large tiles in a window with $w = 8$ transactions.

4.3 Problem definition

Let $\sum$ be an alphabet of items, a transaction T is an itemset $T \subseteq \sum$. Let $S = T_1T_2 \cdots T_n$ be a stream of transactions where each transaction T_t is associated with a timestamp t indicating the order of transaction generation.

In this work we consider the sliding window model in which a window of the w most recent transactions is monitored. A sliding window of size w at timepoint t is denoted as W_t and is defined as $W_t = T_{t-w+1} \cdots T_{t-1}T_t$.

For any given itemset $I \subseteq \sum$ the frequency of I in a sliding window W_t, denoted as $f_t(I)$, is defined as the number of transactions in W_t that are the supersets of I. Let $|I|$ be the cardinality of the set I, the area of I in the window W_t denoted as $A_t(I)$ is defined as the multiplication of the frequency and the size of I, i.e. $A_t(I) = f_t(I) * |I|$.

Example 6 (Large tiles)**.** *Figure 4-1 shows a window with 8 transactions represented as rows of a binary matrix. In each row, an element is equal to 1 if the corresponding item belongs to the transaction. In this figure, three large tiles abc, degh and ghk with respective area 15, 12 and 15 are highlighted.*

An item set is closed in a window W_t if there is no superset of I with the same frequency as I in the window W_t. In this work, we are interested in mining the closed itemsets with

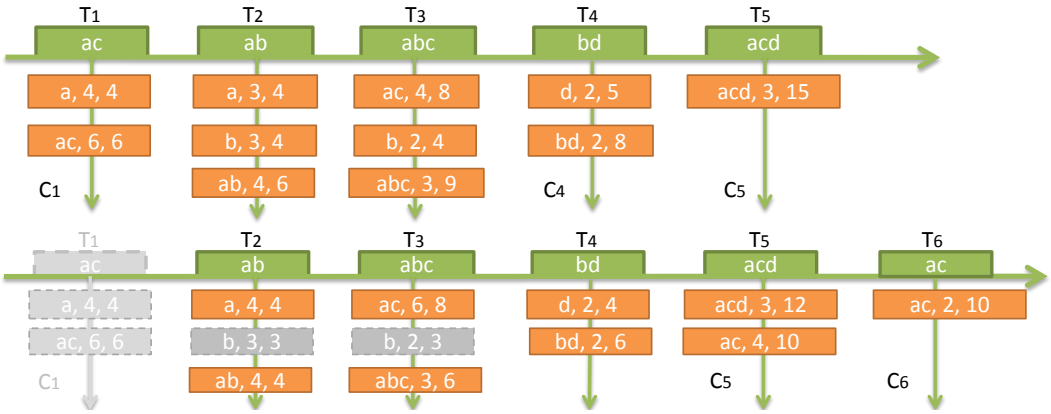


Figure 4-2: An example of the summary maintained by the *cTile* algorithm for sliding windows with size $w = 5$. The summary contains candidate itemsets which can become a top-k tile in the future. Every candidate is associated with a *lower-bound* and an *upper-bound* on the area which are used to prune the candidate list.

the largest area for every sliding window. Such itemsets are usually called large tiles in the literature [27]. From now on we use the term tiles to refer to closed tiles. The problem of mining the top-k largest tiles in a data stream can be formulated as follows:

Definition 4 (Stream Tiling)**.** *Given a data stream of itemset transactions, a parameter k and a window size w, maintaining a summary of a sliding window with size w such that from the summary we can calculate the top-*k *largest tiles of the sliding window efficiently.*

4.4 Algorithms

It was proven that the problem of mining the largest tile in a database is NP-Complete and is even inapproximable [27]. Therefore, recalculation of the top-k largest tiles from scratch every time the window is updated is very time-demanding. In this section we discuss efficient solutions for the given problem under the streaming context.

A candidate-based algorithm:

We first discuss an exact algorithm named *cTile* which maintains a summary of the sliding window containing candidate itemsets that potentially can become top-k largest tiles in any future window. This algorithm is much more efficient than the straightforward solution that always recalculates the set of largest tiles from scratch every time the window is sliding.

The general idea of the algorithm is as follows: at every transaction T_i we keep a

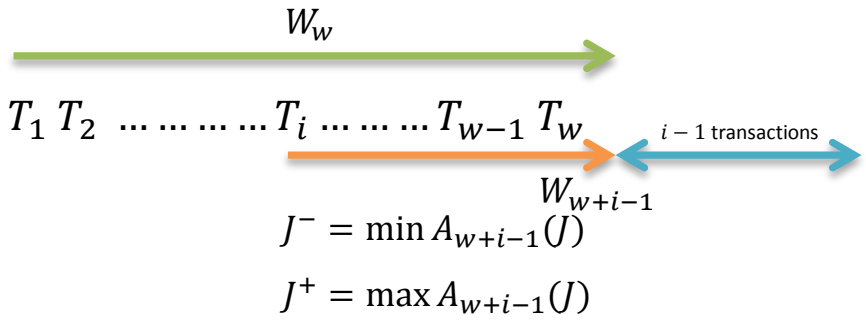


Figure 4-3: Illustration of how the bounds are calculated.

candidate list C_i of all closed subsets of the given transaction which can become a top-k tile in a future sliding window. In order to identify these closed itemsets, we keep a *lower-bound* and an *upper-bound* on the area of these itemsets in any future sliding window in which T_i is not expired. These bounds will be used to infer which itemsets can be top-k tiles and which sets for sure cannot be top-k tiles in any future window. Given a window $W = T_1 \cdots T_{w-1} T_w$, for every closed itemset J in the candidate list $C_i (1 \leq i \leq w)$ we denote J^- and J^+ as the *lower-bound* and the *upper-bound* of the area of this itemset in any future sliding window in which T_i is not expired. The lower-bound J^- at time point t is calculated as the area of J in the transactions $T_i, T_{i+1}, \cdots, T_w$ and the *upper-bound* J^+ is calculated as

$$J^+ = J^- + (i-1) \times |J| \tag{4.1}$$

Proposition 1. *J^- and J^+ are the correct lower-bound and upper-bound on the area of J in any window containing the transaction T_i*

Example 7 (Bounds)**.** *Figure 4-2 (upper-part) shows an example of a sliding window with size $w = 5$. Each candidate J in a candidate list C_i is associated with two numbers corresponding to the lower-bound and the upper-bound on the area of the candidate. For instance, for the candidate $J = ac$ associated with the candidate list C_3 the lower-bound is $J^- = 4$ because the area of ac in the set of currently observed transactions $T_3, \cdots, T_5$ is $2 * 2 = 4$. For any future window in which the transaction T_3 is not expired, the area of ac is at least as large as 4. Meanwhile $J^+ = 8$ because the area of ac in any future window that contains T_3 is at most $4 + 2 * 2 = 8$. The tight bound only happens when the new transactions T_6 and*

Algorithm 6 cTile(S_t)

1: **Input**: A stream of transactions $T_1, T_2, \cdots, T_\infty$, a sliding window size w and a parameter k
2: **Output**: Summary C for calculating top-k largest tiles
3: $C \longleftarrow \{C_1, C_2, \cdots, C_w\}$ //candidate lists
4: **for** $t = w \rightarrow \infty$ **do**
5: $C \leftarrow C \setminus \{C_{t-w+1}\}$
6: $C_{t+1} \leftarrow \{T_{t+1}\}$
7: $C \leftarrow C \cup \{C_{t+1}\}$
8: **for** $i = t \rightarrow t - w + 2$ **do**
9: $C_i \longleftarrow C_i \bigcup \text{intersect}(C_i, T_{t+1})$
10: **for** $J \in C_i$ **do**
11: $J^- = |\{i \leq l \leq t | J \subseteq T_l\}| * |J|$
12: $J^+ = J^- + (w - t + i - 1)|J|$
13: **end for**
14: Pruning(C_i)
15: **end for**
16: **end for**
17: Return C

T_7 both contain ac.

Algorithm 6 incrementally maintains the candidate lists C_i in the summary. When a new transaction is appended to the end of the window, J^+ and J^- are updated accordingly (line 10-13). The new transaction is intersected with every existing closed itemset to create new closed itemsets being added to the corresponding candidate list (line 8-9). If there exists another closed itemset candidate $B \in C_j$ kept at a younger transaction T_j ($j > i$), such that B^- is ranked k^{th} in all transactions younger than T_i (including itself) and meanwhile $B^- > J^+$, then J will never be among top-k tiles in any future sliding window, hence J is removed from the list of candidates. When a transaction is expired, it is removed from the window along with the candidate list stored at the given transaction.

Example 8 (cTile)**.** *Figure 4-2 (bottom part) shows one update step of Algorithm 6 when a new transaction $T_6 = \{a, c\}$ is appended to the window and T_1 together with C_1 are removed from the summary. First, T_6 is intersected with all the existing candidates in $C_2, C_3, \cdots, C_5$ to add new closed sets to the candidate lists. After that all the lower-bounds and the upper-bounds are updated accordingly.*

Assume that we want to get top-3 largest tiles in the sliding window, i.e. $k = 3$. Since $(b, 3, 3)$ in C_2 has the upper-bound *equal to 3. It is removed from the summary because the itemset ranked at the third position in the candidates lists of younger transactions (including*

T_2) are $(a, 4, 4)$, $(ac, 4, 10)$, $(ab, 4, 4)$ whose lower-bound *is 4. The same pruning operation can be applied for $(b, 2, 3)$ in C_3.*

Theorem 3. *Given k, w and a stream of transactions, using the summary in Algorithm 6 we can answer the top-*k *largest tiles exactly.*

Proof. Assume that there exists a top-k largest tile I in a window W_t recognized as a none top-k largest tile (false negative). Denote T_i as the first transaction of W_t containing I. False negative only happens when at the moment t $I \notin C_i$. Therefore either I is directly pruned from C_i by pruning criteria or indirectly pruned because of its closed supersets are pruned.

Since the bounds are exact the former case cannot happen. The latter case happens when I is not a closed itemset at the moment its closed supersets are pruned. In such case, the upper-bound of I is always less than the upper-bound of its closed supersets which were used to prune the supersets. Therefore, I is not a top-k tile in W_t. Both cases lead to contradiction. □

An approximation algorithm:

The size of the summary maintained by *cTile* grows quickly when the window size increases making it inefficient for monitoring large windows. The main reason is that each time a new transaction arrives, it has to be intersect with a large amount of candidates, which is a time-consuming operation. Therefore, in this section we discuss another approximation algorithm named *aTile* that approximates the set of largest tiles efficiently.

The main process of the approximation algorithm is almost the same as Algorithm 6. The only difference of these two algorithms lies in the method of candidate pruning. Instead of using an *upper-bound* on the area, *aTile* tries to approximate the area and uses this estimate of the area to prune the candidate in the same way as *cTile* does.

Given a window $W_t = T_1 \cdots T_{w-1} T_w$, assume that J is kept in the candidate list C_i of transaction T_i. The *lower-bound* J^- is calculated as in the *cTile* algorithm. Let us denote the *estimate* of the area for a itemset J as J^*, this value is calculated as follows:

$$J^* = \begin{cases} J^- + (i-1) \times |J| & \text{if } w - i + 1 \leq L \\ \frac{J^-}{(w-i+1)} \times w & \text{if } w - i + 1 > L \end{cases} \tag{4.2}$$

Equation assumes that the probability of observing the itemset J in the next $i-1$ coming transactions does not change. A parameter L is introduced as a threshold for the minimum number of transactions needed for accurately estimating the area of the candidate. Generally, the larger the value of L is, the more precise the estimate is.

All the steps of the of the $aTile$ algorithm are very similar to the $cTile$ algorithm except that it uses J^* instead of J^+ for candidate pruning. In the experiment section we empirically show that the $aTile$ algorithm is more efficient than the $cTile$ algorithm because its candidate set is more concise. An important property of the $aTile$ algorithm is that under reasonable assumptions the error rate on the accuracy of the result is bounded.

4.5 Theoretical analysis on the error rate bounds

In this section, we show theoretical bounds on the probability of errors induced by the $aTile$ algorithm. We mainly show that the error rate is extremely low when L is large enough. Let us denote the probability of observing an itemset I as a subset of a transaction in the data stream as $P(I) = \mu_I$. Therefore, the expectation of the area of the itemset I in a window of size w is $w|I|P(I) = w|I|\mu_I$. We consider two types of error event:

- *False negative (FN)*: in a window W_{t^*} for some t^*, there is a true top-k largest tile that is not present in the list of the top-k largest tiles returned by the $aTile$ algorithm.
- *False positive (FP)*: in a window W_{t^*} for some t^*, there is a non top-k largest tile that is present in the top-k largest tile list returned by the $aTile$ algorithm.

False negative bound

Let us assume that I is a true top-k largest tile in the window W_{t^*} and $I_k^{t^*}$ is ranked at position k in the list of a true top-k largest tile of the window W_{t^*}. Denote $\triangle \geq 0$ as the difference between the average area of I and $I_k^{t^*}$ in the window W_{t^*}, i.e. $\triangle = \frac{A_{t^*}(I) - A_{t^*}(I_k^{t^*})}{w}$. The following lemma show the relationship between the probability of a false negative and L, $\triangle$:

Lemma 14. *If the transactions are independent and identically distributed (i.i.d) the probability that a random top-*k *tile is not reported, is bounded as follows:*

$$Pr(FN) < 4w * e^{-\frac{L\triangle^2}{2|I|^2}}$$

Proof. A false negative happens in the window W_{t^*} with respect to the itemset I if its area is underestimated in that window. This event happens only when there exists at least one moment $t < t^*$ such that an instance of I is pruned from the candidate list C_i $(t-w < i \leq t)$.

Given a time point t, denote W_0 as a window containing transactions $T_i, T_{i+1}, \cdots, T_t$ where $|W_0| = w_o > L$ and denote $f_0(I)$ as the frequency of I in the window W_0. When t is given, the event "I is removed from C_i" (denoted as R_t) only happens when the estimate of the *upper-bound* on the area of I is less than the area of $I_k^{t^*}$, i.e. $f_{W_0}(I)|I|\frac{w}{w_0} < A_{t^*}(I_k^{t^*})$. Therefore, we have the following inequalities:

$$
\begin{aligned}
Pr(R_t) &< Pr\left(f_0(I)|I|\frac{w}{w_0} \leq A_{t^*}(I_k^{t^*})\right) \\
&< Pr\left(f_0(I)|I|\frac{w}{w_0} < A_{t^*}(I) - w\triangle\right) \\
&< Pr\left(\frac{f_0(I)}{w_0} < \frac{f_{t^*}(I)}{w} - \frac{\triangle}{|I|}\right) \\
&< Pr\left(\left|\frac{f_0(I)}{w_0} - \frac{f_{t^*}(I)}{w}\right| > \frac{\triangle}{|I|}\right) \\
&< Pr\left(\left|\frac{f_0(I)}{w_0} - \mu_I\right| + \left|\frac{f_{t^*}(I)}{w} - \mu_I\right| > \frac{\triangle}{|I|}\right) \\
&< Pr\left(\left|\frac{f_0(I)}{w_0} - \mu_I\right| > \frac{\triangle_0}{w_0}\right) + \\
&Pr\left(\left|\frac{f_{t^*}(I)}{w} - \mu_I\right| > \frac{\triangle_1}{w}\right) \\
&< Pr\left(|f_0(I) - \mu_I w_0| > \triangle_0\right) + \\
&Pr\left(|f_{t^*}(I) - \mu_I w| > \triangle_1\right) \qquad (4.3)
\end{aligned}
$$

Where $\triangle_0 = \frac{\triangle w_0}{2|I|}$ and $\triangle_1 = \frac{w\triangle}{2|I|}$. It is important to notice that $w_0\mu_I$ and $w\mu_I$ are the expectation of the frequency of I in the window W_0 and the window W_{t^*} respectively. Since the transactions are independent to each other, according to the Hoeffding inequality [54] we have:

$$
\begin{aligned}
Pr\left(|f_0(I) - w_0\mu_I| > \triangle_0\right) &< 2e^{-\frac{2\triangle_0^2}{w_0}} \qquad (4.4) \\
&< 2e^{-w_0\frac{\triangle^2}{2|I|^2}} \qquad (4.5) \\
&< 2e^{-L\frac{\triangle^2}{2|I|^2}} \qquad (4.6)
\end{aligned}
$$

Similar inequality can be obtained for bounding the second term on the right size of the inequality 4.3 as follows:

$$\begin{aligned} Pr\left(|f_{t^*}(I) - w\mu_I| > \triangle_1\right) &< 2e^{-\frac{2\triangle_1^2}{w}} && (4.7)\\ &< 2e^{-w\frac{\triangle^2}{2|I|^2}} && (4.8)\\ &< 2e^{-L\frac{\triangle^2}{2|I|^2}} && (4.9) \end{aligned}$$

Moreover, since FN happens only when there at least one moment t such that R_t happens, therefore:

$$\begin{aligned} Pr(FN) &< Pr(\cup_{t^*-w<t\leq t^*} R_t) && (4.10)\\ &< \sum_{t^*-w<t\leq t^*} Pr(R_t) && (4.11) \end{aligned}$$

Inequalities 4.3, 4.6, 4.9 and 4.11 prove the lemma. □

A direct corollary of Lemma 14 is shown in the following theorem:

Theorem 4. *If I is strictly more important than $I_k^{t^*}$, i.e. the expectation of the area of I in a transaction is strictly greater than the expectation of the area of $I_k^{t^*}$ in a transaction and $L = O(w)$ then:* $\lim_{L\longrightarrow\infty} Pr(FN) = 0$

Proof. Denote $E(A(I))$ as the expectation of the area of I in a transaction and $E(A(I_k^{t^*}))$ as the expectation of the area of $I_k^{t^*}$ in a transaction. Since $\triangle = \frac{A_{t^*}(I) - A_{t^*}(I_k^{t^*})}{w}$ we can imply that:

$$\begin{aligned} \frac{\triangle}{|I|} &= \frac{A_{t^*}(I) - A_{t^*}(I_k^{t^*})}{w|I|} && (4.12)\\ &\simeq \frac{E(A(I)) - E(A(I_k^{t^*}))}{|I|} && (4.13) \end{aligned}$$

The last equation is the result of the law of large number when L goes to ∞. From the last equation we can imply that $\lim_{L\longrightarrow\infty} 4we^{-L\frac{\triangle^2}{2|I|^2}} = 0$ from which the theorem is proved. □

False positive bound

In this subsection we prove similar bound for false positive error. The bound of false positive can be obtained in a similar way as the bound of false negative but with non-trivial reasoning

changes.

Denote J as an itemset that is not a true top-k largest tile in the window W_{t^*} but returned by the $aTile$ algorithm as a false positive tile. Denote $I_k^{t^*}$ as an itemset ranked at position k in the list of a true top-k largest tile of the window W_{t^*}. Denote $\triangle \geq 0$ as the difference between the area of J and $I_k^{t^*}$ in the window W_{t^*}, i.e. $\triangle = \frac{A_{t^*}(I_k^{t^*}) - A_{t^*}(J)}{w}$.

A false positive happens if there exists a true top-k tile I of the windows W_{t^*} and at least one moment $t < t^*$ such that an instance of I is pruned from the candidate list C_i $(t - w < i \leq t)$. We have the following inequality:

$$A_{t^*}(J) < A_{t^*}(I_k^{t^*}) \leq A_{t^*}(I) \tag{4.14}$$

The following lemma show the relationship between the probability of a false positive and L, $\triangle$:

Lemma 15. *If the transactions are i.i.d the probability of false positive, i.e. the event that a random non top-*k *tile is reported, is bounded as follows:*

$$Pr(FP) < 4w * e^{-\frac{L\triangle^2}{2|I|^2}}$$

Proof. Given t, denote W_0 as a window containing transactions $T_i, T_{i+1}, \cdots, T_t$ where $|W_0| = w_o > L$ and denote $f_0(I)$ as the frequency of I in the window W_0. The event that I is removed from C_i (denoted as R_t) only happens when the estimate of the *upper-bound* on the area of I is less than the area of J calculated from the summary at time point t^*. The latter value is less than the true area of J in W_t^* so according to inequality 4.14 and using similar proof in lemma 14, we obtain the following inequalities:

$$\begin{aligned} Pr(R_t) &< Pr\left(f_0(I)|I|\frac{w}{w_0} \leq A_{t^*}(J)\right) \\ &< Pr\left(f_0(I)|I|\frac{w}{w_0} < A_{t^*}(I) - w\triangle\right) \\ &< 4 * e^{-\frac{L\triangle^2}{2|I|^2}} \end{aligned} \tag{4.15}$$

Moreover, since FP happens only when there at least one moment t such that R_t

happens, therefore:

$$Pr(FP) < Pr(\cup_{t^*-w<t\leq t^*} R_t) \quad (4.16)$$

$$< \sum_{t^*-w<t\leq t^*} Pr(R_t) \quad (4.17)$$

Inequalities 4.15 and 4.17 prove the lemma. □

A direct corollary of Lemma 15 is shown in the following theorem:

Theorem 5. *If large tile size is bounded and if J is strictly less important than $I_k^{t^*}$, i.e. the expectation of the area of J in a transaction is strictly less than the expectation of the area of $I_k^{t^*}$ in a transaction and $L = O(w)$ then:* $\lim_{L\longrightarrow\infty} Pr(FP) = 0$

Proof. Denote $E(A(J))$ as the expectation of the area of J in a transaction and $E(A(I_k^{t^*}))$ as the expectation of the area of $I_k^{t^*}$ in a transaction. Since $\triangle = \frac{A_{t^*}(I_k^{t^*}) - A_{t^*}(J)}{w}$ we can imply that:

$$\frac{\triangle}{|I|} = \frac{A_{t^*}(I_k^{t^*}) - A_{t^*}(J)}{w|I|} \quad (4.18)$$

$$\simeq \frac{E(A(I_k^{t^*})) - E(A(J))}{|I|} \quad (4.19)$$

The last equation is the result of the law of large number when L goes to ∞. If the size of I is bounded, from the last equation we can imply that $\lim_{L\longrightarrow\infty} 4we^{-L\frac{\triangle^2}{2|I|^2}} = 0$ from which the theorem is proved. □

Theorem 5 and Theorem 4 show an interesting result that the probability of false positive and false negative decrease exponentially with $L = O(w)$. It is important to notice that the condition $L = O(w)$ can be replaced by a weaker assumption $L = k \log w$ for some constant values k. If k is large enough the bound is also closed to zero. Although the bounds are not tight, in experiments, we empirically show that false negative rate and false positive rate are negligible even when L is set to a small value.

Long Lasting Tiles

The two previous Theorems give probabilistic results. We can also show a deterministic one: if an itemset stays in the top-k largest tiles for more than $2w$ consecutive windows and its

area is at least twice the area of the k-th tile, then the *aTile* algorithm finds it. An important point of this theorem is that it does not depends on the value of L. Therefore, even if the probability of false negative or false positive is higher with a small L, the algorithm is still able to mine the most interesting tiles (we assume that a tile which appear in the top-k in only one window is less interesting than a tile which appear in $2w$ consecutive ones).

Theorem 6. *Let z be a time and J be an itemset. If the area of J in every windows W_t with $z \leq t \leq z+2w$ is larger than twice the area of the k-th largest tile in this window (i.e., $A_t(J) \geq 2 * A_t(I_k^t)$), then there is a time $z \leq t^* \leq z+2w$ such that J (or one of its superset) is in C_{t^*} and is never pruned by the* aTile *algorithm.*

Proof. We define hypothesis H as: t^* does not exist. We will show that if H is true, it leads to a contradiction and thus the theorem must be true. In all this proof, we denote (x, y) the subsequence of transactions $T_x \cdots T_y$. We denote $z \leq t_1 < \ldots < t_k \leq z + 2w$ the times of occurrence of J in the subsequence $(a, a + 2w)$.

If H is true, then for every t_i, itemset J and its supersets must be pruned from C_{t_i} at some time no later than $t_i + w - 1$. Denote t'_i as the time that J would be pruned if it was in C_{t_i}, i.e., the first time when the estimate of the upper bound J^* is lower than the area of the k-th largest tile in subsequence (t_i, t'_i). (J may not be in C_{t_i} because one of its superset is). Thanks to our hypothesis H, $t'_i < t_i + w$.

We denote S_i the subsequence $(t_i, t'_i) = T_{t_i} \cdots T_{t'_i}$. We order all the subsequences S_i by inclusion (i.e., $S_i \subseteq S_j$ if $t_j \leq t_i$ and $t'_i \leq t'_j$). Since J is a top-k tile in window W_{z+w-1}, there is at least an occurrence of J in this window (at time t_x). Let $S_0 = (t_0, t'_0)$ be a maximal subsequence that includes S_x and let $W_0 = W_{t'_0}$. Since $z \leq t_x < z + w$ and $t_0 \leq t_x \leq t'_0 < t_0 + w$, we have $z \leq t'_0 < z + 2w$ and therefore the area of J in W_0 is larger than $A_{t_0}(I_k^{t_0})$ (by theorem hypothesis).

Step 1: From now on, we consider only the maximal subsequences included in W_0, i.e., the set $\mathcal{S} = \{S_i \mid S_i \subseteq W_0 \text{ and } S_i \text{ is maximal in } \mathcal{S}\}$. Consider an occurrence of J in W_0 at time t_i. If $S_i \notin \mathcal{S}$ it means either $S_i \subseteq S_j \in \mathcal{S}$ or $S_i \not\subseteq W_0$. In the latter case, t_i cannot be smaller than t_0 because otherwise $S_0 \subseteq S_i$ and S_0 cannot be maximal. Thus, t_i is either in S_j or in S_0. All occurrences of J in W_0 are thus in one of the $S_i \in \mathcal{S}$.

Step 2: If three subsequences S_i, S_j and S_k of $\mathcal{S}$ overlap, it means that one of these subsequences is included in the union of the two other, e.g., $S_i \subseteq S_j \cup S_k$ (otherwise, one of

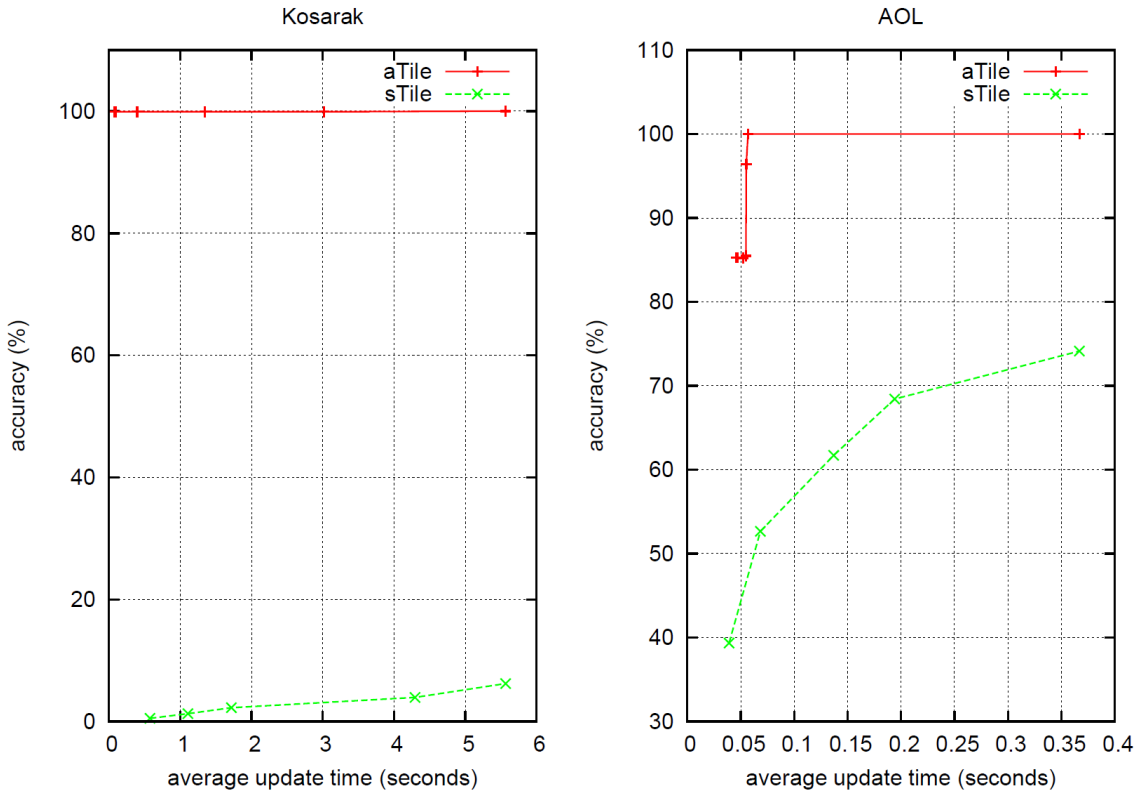


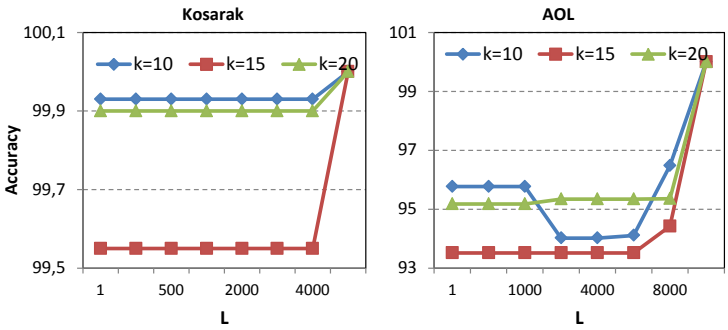
Figure 4-4: Accuracy (y-axis) versus average update time in seconds (x-axix) of the *aTile* algorithm and the *sTile* algorithm when L and the number samples increases

these subsequence is not maximal and cannot be in $\mathcal{S}$ by step 1). In this case, we remove S_i from $\mathcal{S}$.

Denote $A_J(S_i)$ as the area of J in S_i and denote A_i as the area of the tile used for pruning J in C_{t_i} at time t'_i, it means: $\frac{w.A_J(S_i)}{(t'_i - t_i + 1)} < A_i$. Let $A = \max_i A_i$, then $A_J(S_i) < A\frac{(t'_i - t_i + 1)}{w}$, make the sum on i : $\sum_i A_J(S_i) < \frac{A}{w}\sum_i((t'_i - t_i + 1))$ and since all J are covered $\sum_i A_J(S_i) > A_J(W)$ and since no 3 subsequences overlap $\sum_i((t'_i - t_i + 1)) < 2.w$ from which $A_J(W) < 2A$ which contradict the hypothesis because A always less than the area of the k-th tile in W_0. □

4.6 Experiments

In this section, we perform experiments with two real-life data streams to compare the proposed algorithms to the baseline approaches with respect to the efficiency and the accuracy



Kosarak		AOL	
w	Accuracy	w	Accuracy
500	99.1	1000	93,4
1000	99.3	2000	95,2
2000	98.3	4000	89,7
3000	99.5	6000	85,1
4000	99.8	8000	89,4
5000	99.9	10000	95,7

Figure 4-5: The average accuracy of the *aTile* algorithm when L is varied. In the table: the average accuracy of the top-*10* largest tiles returned by *aTile* when the window size is varied and $L = 0.1w$

of the results. The descriptions of the two datasets are as follows:

- Kosarak: contains about one million transactions of click log by the users of a websites. Each item corresponds to a page in the website. Kosarak dataset contains some large transactions with size up to 500.

- AOL: contains 100000 transactions of queries by the users of the AOL search engine. Each item corresponds to a keyword in the search query. This dataset contains a lot of short queries and the longest query contains only 26 keywords.

The baseline algorithms for comparison are as follows:

- *Tile*: the original implementation of the tiling database work [27]. This algorithm is originally proposed for a database. In order to adopt this algorithm for a data stream with sliding windows, we use *Tile* to recalculate the top-k largest tiles from scratch whenever the windows is sliding.

- *sTile*: adopts a sampling technique proposed in [7]. The *sTile* algorithm samples N itemsets from the window such that each itemset is sampled with probability proportional to the area of the itemset. If the number of samples is large the largest tiles will be present in the sampled set with a high probability. The top-k largest tiles are calculated by going through the window again and evaluate the area of each sample and extract the top-k samples with largest area.

The datasets, the video demo along with the source code of the *aTile* and the *cTile* algorithms written in C++ are available for download at our project website[3]. Evaluation

[3] `http://www.win.tue.nl/~lamthuy/tile.htm`

was done in a 4*2.0 GHz (i7-2630QM) GB of RAM, with Window 7 professional.

Accuracy

In this subsection, we perform experiments to demonstrate the effectiveness of the *aTile* algorithm in term of accuracy. In the first experiment we fix the window size to $w = 5000$ for the Kosarak dataset and $w = 10000$ for the AOL dataset. Figure 4-5 shows the average accuracy calculated as the precision of the top-k largest tiles returned by the *aTile* algorithm when L increases. The results show that even for very small L the accuracy is very high, e.g. 99% in the Kosarak dataset and 93% in the AOL dataset. The accuracy increases and reaches 100% accuracy when L is increased. In the same figure we can also see that the results are not significantly different with different values of k.

In order to compare the results with the approximation algorithm *sTile*, we plot the accuracy (y-axis) versus average update time (x-axis) in Figure 4-4. To obtain that plot we made both L for the *aTile* algorithm and the number of samples of the *sTile* algorithm vary. Recall that the *sTile* algorithm depends on the number of samples it collects from the window. The larger the number of samples the more accurate the results are but the more memory it consumes and the more update time is. This fact is illustrated in Figure 4-4 in which the *sTile* algorithm's accuracy is significantly lower than the accuracy of the *aTile* algorithm given about the same average update time.

The accuracy of *sTile* maybe negatively influenced by the fact that *sTile* returns any itemset instead of closed itemsets. However, when we calculated the average sum of the area of the top-*10* tiles in the Kosarak dataset when $w = 5000$, we observed that the average sum of the area was significantly lower than the average sum of the area of the *aTile* algorithm (Figure 4-6). The sum increased when the number of sample increased. This experiment shows that *sTile* needs a large number of samples to acquire large tiles.

Efficiency

Figure 4-7 compares the average update time of four algorithms when $k = 10$ and with an increasing window size. The number of samples in the *sTile* algorithm is set to $N = 80000$. The average accuracy is then about 5% in the Kosarak dataset and 73% in the AOL dataset (the accuracy increases when N is set to a larger number but the update time will be significantly slower). In subsection 4.6 we have already shown that the accuracy of the

sTile		aTile	
# samples	Sum of area	L	Sum of area
5000	2652	1000	20384
10000	2753	2000	20384
20000	2822	3000	20384
40000	2918	4000	20384
80000	2992	5000	20388

Figure 4-6: The average sum of the area of the top-10 tiles returned by the *aTile* and the *sTile* algorithms.

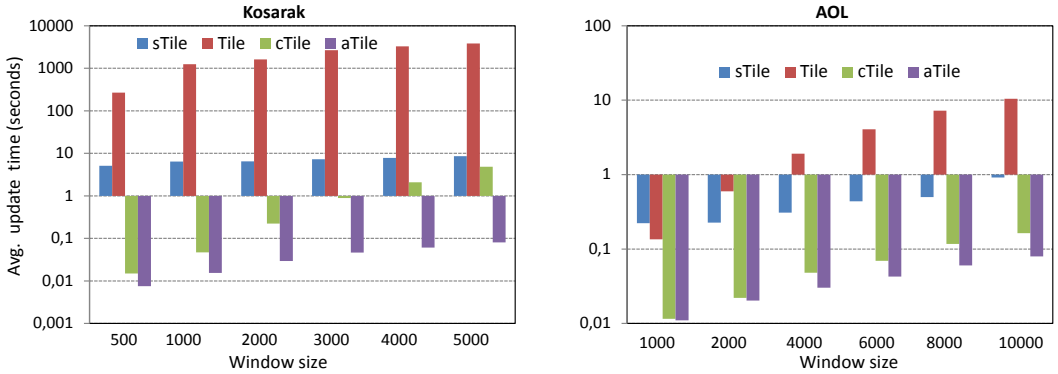


Figure 4-7: Average time per update according to the window size.

aTile algorithm is not sensitive to L so we set $L = \frac{w}{10}$. In term of update time, *aTile* is an order of magnitude (50x) faster than the *cTile* algorithm and about two to three (100x) order of magnitude faster than the *sTile* and the *Tile* (1000x) algorithms. The speed-up is larger when the window size is bigger.

In Figure 4-8, the last plot shows the number of candidates kept in the summary of the *aTile* and the *cTile* algorithm when the window size is varied (the result of the AOL dataset is omitted because it is very similar to the results of the Kosarak dataset in this experiment). The *aTile* is not only faster but it is more memory efficient than the *cTile* algorithm as the number of candidates it keeps in the summary is much lower.

Finally, in Figure 4-8 we show the running time when the *aTile*, *cTile* and *Tile* algorithms are used to find the top-*10* largest tiles in a static corpus with varying size. The purpose of the experiment is to see whether the *aTile* and the *cTile* algorithm can find the large tiles in a static corpus more efficiently than the *Tile* algorithm. For the AOL dataset, when most of transactions are small, the *Tile* algorithm is faster than both *aTile* and *cTile*. However, for the Kosarak dataset, when the average transaction size is larger, the *aTile*

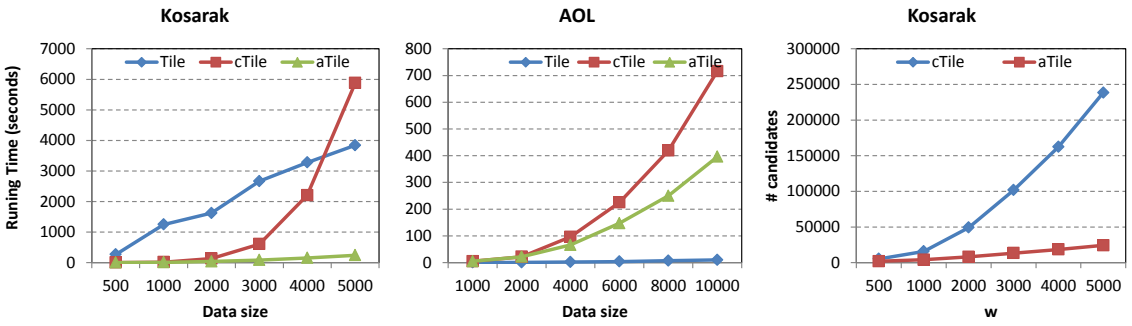


Figure 4-8: Running time according to the database size when the algorithm is used to mine large tiles in a database instead of a stream. The right most subplot shows the number of candidates in the summary when the window size increases.

Figure 4-9: Three snapshots (ordered by timestamps) of the demo show the evolution of the largest tiles of the AOL query stream overtime. Each snapshot is visualized such that larger words correspond to tiles with larger area. It shows important search keywords by the AOL's users and their evolution overtime.

algorithm is much more efficient. Therefore, the *atile* algorithm can not only be used for mining large tiles from a data stream but also to mine large tiles from a database efficiently.

A Use-case

In order to show a potential application of the work we created a demo video which can be watched at[4] to visualize the top-k largest tiles of the AOL query stream. Each snapshot of the video corresponds to a list of the largest tiles extracted from a sliding window.

Figure 4-9 shows example snapshots taken from the demo video. Three snapshots of the demo (ordered by timestamps) show the evolution of the largest tiles overtime. Each snapshot is visualized by the *wordcloud* tool in *R*. Larger words correspond to tiles with

[4] `http://www.youtube.com/watch?v=3UCjs9d91_g`

larger area. The demo shows important search keywords by the AOL's users and their evolution overtime. For example, in the first snapshot the keyword *"real estate"* is the most important tile. However, it becomes less important in the second and the third snapshot. Meanwhile, new important tiles such as *"high school"* or *"online casino"* take over the first position in the second and the third snapshot. With this demo, users can see the dynamic of keyword search in the AOL query log overtime.

Part II

Mining Non-redundant Sets of Patterns in Data Streams

Chapter 5

Mining Compressing Sequential Patterns in Sequence Databases

Pattern mining based on data compression has been successfully applied in solving the redundancy issue in frequent pattern mining for itemset data. However, for sequence data, the redundancy issue is not fully addressed in the literature. In this chapter we study MDL-based algorithms for mining non-redundant sets of sequential patterns from a sequence database[1]. First, we propose an encoding scheme for compressing sequence data with sequential patterns. Second, we formulate the problem of mining the most compressing sequential patterns from a sequence database. We show that this problem is intractable and belongs to the class of inapproximable problems. We propose two heuristic algorithms: the first one uses a two-phase approach similar to Krimp for itemset data. To overcome performance issues in candidate generation, we propose GoKrimp, an algorithm that directly mines compressing patterns by greedily extending a pattern until no additional compression benefit. Since checks for additional compression benefit of an extension are computationally expensive we propose a dependency test which only chooses related events for extending a given pattern. We conduct an empirical study on eight datasets to show the effectiveness of our approach in comparison to the state of the art algorithms in terms of interpretability of the extracted patterns, run time, compression ratio, and classification accuracy using the discovered patterns as features for different classifiers.

[1]This chapter was published as: Hoang Thanh Lam, Fabian Moerchen, Dmitriy Fradkin, Toon Calders: *Mining Compressing Sequential Patterns*. SDM 2012. The work is extended and published in the special issue of the best works at SDM 2012 published by the journal of Statistical Analysis and Data Mining, Wiley in May 2013.

Pattern	Support	Pattern	Support
algorithm algorithm	0.376	method method	0.250
learn learn	0.362	algorithm result	0.247
learn algorithm	0.356	Data set	0.244
algorithm learn	0.288	learn learn learn	0.241
data data	0.284	learn problem	0.239
learn data	0.263	learn method	0.229
model model	0.260	algorithm data	0.229
problem problem	0.258	learn set	0.228
learn result	0.255	problem learn	0.227
problem algorithm	0.251	algorithm algorithm algorithm	0.222

Figure 5-1: The 20 most frequent non-singleton closed sequential patterns from the JMLR abstracts datasets. This set, despite containing some meaningful patterns, is very redundant.

5.1 Introduction

Mining frequent sequential patterns from a sequence database is an important data mining problem which has been attracting researchers for more than a decade. Dozens of algorithms [51] to find sequential patterns effectively have been proposed. However, relatively few researchers have addressed the problem of reducing redundancy, ranking patterns by interestingness, or using the patterns for solving further data mining problems.

Redundancy is a well known problem in sequential pattern mining. Let us consider the JMLR dataset which contains a database of word sequences, each corresponding to an abstract of an article in the *Journals of Machine Learning Research*. Figure 6-1 shows the 20 most frequent closed sequential patterns ordered by decreasing frequency. This set of patterns is clearly very redundant, so many patterns with very similar meaning are shown to users.

Beside redundancy issues, the set of frequent patterns usually contain trivial and meaningless patterns. In fact, the set of frequent closed patterns in Figure 6-1 contains random combinations or repeats of frequent terms in the JMLR abstracts such as *algorithm, result, learn, data* and *problem*. These patterns are meaningless given our knowledge about the frequent terms.

In order to solve these issues, we have to find alternative interestingness measures rather than relying on frequency alone. For itemset data, an interesting approach has been proposed recently. The Krimp algorithm mines patterns that compress the data well [76] using

the Minimum Description Length principle [30]. This approach has been shown to reduce redundancy and generate patterns that are useful for classification [76], component identification [74] and change detection [73]. We extend these ideas to sequential data. The key issue in designing an MDL-based algorithm for sequence data is the encoding scheme that determines how a sequence is compressed given some patterns. In contrast to itemsets we need to consider the ordering of elements in a sequence and need to be able to deal with gaps, as well as overlapping and repeating patterns; all properties not present in itemset data.

In this work we study MDL-based algorithms for mining non-redundant and meaningful patterns from a sequence database. The key contributions of this work can be summarized as follows:

1. We propose a novel encoding for sequence data. Our encoding assigns shorter codewords for small gaps, thus penalizing pattern occurrences with longer gaps. It is shown to be more effective than the encoding proposed in our prior work [40]. Moreover, by using the Elias code for gaps [78], it allows to encode interleaved patterns which is prohibited in the encoding proposed recently in [71].

2. We discuss the complexity of mining compressing patterns from sequence database. The main result shows that this problem is NP-hard and belongs to the class of inapproximable problems.

3. We propose SeqKrimp, a two-phase candidate-based algorithm for mining compressing patterns inspired by the original Krimp algorithm.

4. We propose GoKrimp, an efficient algorithm that directly mines compressing patterns from the data by greedily extending patterns until no additional compression benefit is observed. In order to avoid exhaustive checks of all possible extensions a dependency test technique is proposed which considers only related events for extension. This technique helps the GoKrimp algorithm to be faster than SeqKrimp and the state of the art algorithms while being able to find patterns with similar quality.

5. We perform an empirical study with one synthetic and eight real-life datasets to compare different sets of patterns based on the interpretability of the patterns and on the classification accuracy when they are used as attributes for classification tasks.

5.2 Related work

Mining useful patterns is an active research topic of data mining. Recent approaches can be classified into three major categories: statistical approaches based on hypothesis tests, MDL-based approaches and information-theoretic approaches.

The first direction is concerned with statistical hypothesis testing. The data is assumed to follow a user-defined null hypothesis. Subsequently, standard statistical hypothesis testing is used to test the significance of patterns assuming that the data follows the null hypothesis. If a pattern passes the test it is considered significant and interesting. For example, Gionis et al. [29, 49] use swap randomization to generate random transactional data from the original data. The significance of a given pattern is estimated on randomized data. A similar method is proposed for graph data by Hanhijärvi et al. [33] and Milo et al. [50]. In those works, random graphs with prescribed degree distribution are generated, and significance of a subgraph is estimated on the set of random generated graphs. A similar approach has also been applied to find interesting motifs in time-series data by Castro et al. [13].

A drawback of such approaches is that the null hypothesis must be chosen explicitly by the users. This task is not trivial in different types of data. Frequently, the null hypothesis is too naive and does not fit the real-life data. As a result, all the patterns may pass the test and be considered as significant.

Other research tries to identify interesting sets of patterns without making any assumptions on the underlying data distribution. The approach is based on the MDL principle: it searches for patterns that compress the given data most. Examples of this direction include the Krimp algorithm [76] and direct mining descriptive patterns algorithm [67] for itemset data and the algorithms for graph data [35, 14]. The usefulness of compressing patterns was demonstrated in various applications such as classification [76], component identification [74] and change detection [73].

The idea of using data compression for data mining was first proposed by Cilibrasi et al. [17] for data clustering problem. This idea was also explored by Keogh et al. [37], who proposed to use compressibility as a measure of distance between two sequences. They empirically showed that by using this measure for classification, they were able to avoid setting complicated parameters, which is not trivial in many data mining tasks, while obtaining

promising classification results. Another related work by Faloutsos et al. [22] suggested that there is a connection between data mining and Kolmogorov complexity. While the connection was explained informally there, this notion quickly became the central idea for a lot of recent work on the same topic.

Our work is a continuation of this idea in the specific context of sequence data. In particular, it focuses on using the MDL principle to discover interesting sequential patterns. This work is an extended version of our previous work on the same topic [40]. That work used an encoding scheme which assumes that the cost of storing a number or a symbol is always a constant. Therefore, it does not punish the gaps between events of a pattern which results in using a window constraint parameter to limit a match with a pattern within the constraint window size. Following that work, Tatti and Vreeken [71] proposed the SQS-Search (SQS) approach that punishes gaps by using an encoding with zero cost for encoding non-gaps and higher cost for encoding events with larger gaps. The approach was shown to be very effective in mining meaningful descriptive patterns in text data. However, it does not handle the case of interleaving patterns. In practice, patterns generated by independent processes may frequently overlap. In this work, we propose an encoding that both punishes gaps and handles interleaving patterns.

5.3 Preliminaries

Sequential Pattern Mining

Let $S = (e_1, t(e_1)), (e_2, t(e_2)), \cdots, (e_n, t(e_n))$ denote a sequence of events, where $e_i \in \Sigma$ is an event symbol from an alphabet Σ and $t(e_i)$ is a timestamp of the event e_i. Given a sequence P, we say that S *matches* P if P is a subsequence of S.

Let $\mathfrak{S} = \{S_1, S_2, \cdots, S_N\}$ be a database of sequences. The number of sequences in the database matching P is the *support* f_P of the given sequence. The frequent sequential pattern mining problem is defined as follows:

Definition 5 (Frequent Pattern Mining)**.** *Given a sequence database* $\mathfrak{S}$ *a minimum support value minsup, find all sequences of events* P *such that* $f_P \geq minsup$*.*

A pattern P is called *closed* if it is frequent and there is no frequent pattern Q such that $f_P = f_Q$ and $P \subset Q$. The problem of mining all closed frequent patterns is formulated

Table 5.1: Notation

Notation	Meaning
$\mathfrak{S}$	a database
S	a sequence
D	a dictionary
$\mathcal{C}$	an encoding
$\sum$	an alphabet
e	an event represented by a symbol in $\sum$
$t(e)$	timestamp of the event e
$C(w)$	binary representation of w
$\|C(w)\|$	binary representation length
$L(\mathfrak{S})$	length of data before compression
$L^{\mathcal{C}}(D)$	length of the dictionary
$L^{\mathcal{C}}(\mathfrak{S}\|D)$	length of the data given it is encoded by D with the encoding $\mathcal{C}$
$L^{\mathcal{C}}_{D}(\mathfrak{S})$	total description length of the data in the encoding $\mathcal{C}$ with dictionary D

as follows:

Definition 6 (Closed Pattern Mining)**.** *Given a database of sequences $\mathfrak{S}$ and a minimum support value minsup, find all patterns P such that $f_P \geq minsup$ and P is closed.*

Minimum Description Length Principle

We briefly introduce the MDL principle and MDL-based pattern mining approaches in this subsection. A model M is a set of patterns $M = \{P_1, P_2, \cdots, P_m\}$ used to compress a database $\mathfrak{S}$. Let $L^{\mathcal{C}}(M)$ be the description length of the model M and $L^{\mathcal{C}}(\mathfrak{S}|M)$ be the description length of the database $\mathfrak{S}$ when it is encoded with the help of the model M in an encoding $\mathcal{C}$. Therefore the total description length of the data is $L^{\mathcal{C}}_{M}(\mathfrak{S}) = L^{\mathcal{C}}(M) + L^{\mathcal{C}}(\mathfrak{S}|M)$. Different models and encodings will lead to different description lengths of the database. Informally, the MDL principle states that the best model is the one that compresses the data the most. Therefore, the MDL principle [30] suggests that we should look for the model M and the encoding $\mathcal{C}$ such that $L^{\mathcal{C}}_{M}(\mathfrak{S}) = L^{\mathcal{C}}(M) + L^{\mathcal{C}}(\mathfrak{S}|M)$ is minimized.

The central question in designing an MDL-based algorithm is how to encode data given a model. In an encoding, the data description length is fully determined by an implicit probability distribution assumed to be the true distribution generating the data. Therefore, designing an encoding scheme is as important as choosing an explicit probability distribution

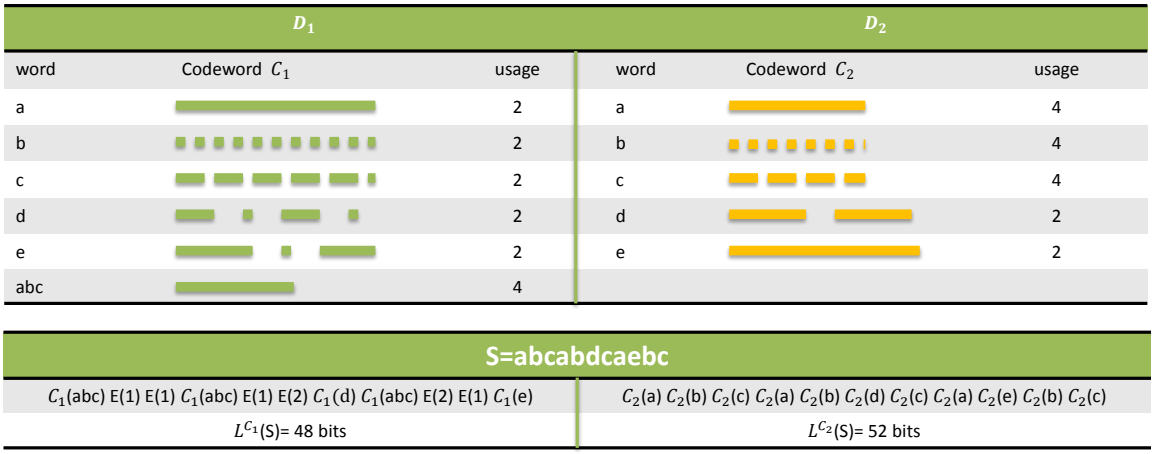


Figure 5-2: An example of two dictionaries and two encodings of the same sequence $S = abcadbcaebc$. In every dictionary, words are associated with codewords. Words with more usage are assigned with shorter codewords.

generating data in classical Bayesian statistics.

5.4 Data Encoding Scheme

In this section we explain how to encode the data given a set of sequential patterns.

Dictionary presentation

Let $\sum = \{a_1, a_2, \cdots, a_n\}$ be an alphabet containing a set of characters a_i. A dictionary D is a table with two columns: the first column contains a list of words $w_1, w_2, \cdots, w_m$ including also all the characters in the alphabet $\sum$, while the second column contains a list of codewords of every word w_i in the dictionary denoted as $C(w_i)$. Codewords are unique identifiers of the corresponding words and may have different length depending on the word usage, as defined in subsection 5.4.

The binary representation of a dictionary is given as follows: it starts with n codewords of all the characters in the alphabet followed by the binary representations of all non-singleton dictionary words. For any non-singleton word w, its binary representation contains a sequence of codewords of its characters followed by its codeword $C(w)$. For instance, the word $w = abc$ is represented in the dictionary as $C(a)C(b)C(c)C(w)$. This binary representation of the dictionary allows us to get any word from the dictionary given its codeword.

Number	Elias code E(n)	Number	Elias code E(n)
1	1	5	00101
2	010	6	00110
3	011	7	00111
4	00100	8	0001000

Figure 5-3: An example of Elias codes of the first eight natural numbers. Code length of $E(n)$ is equal to $2\lfloor log_2(n) \rfloor + 1$.

Example 9 (Dictionary)**.** *In Figure 5-2, two different dictionaries D_1 and D_2 are shown. The first dictionary contains both singleton and non-singleton words while the second one has only singletons. As an example, the binary representation of the first dictionary is* $C_1(a)C_1(b)C_1(c)C_1(d)C_1(e)C_1(a)C_1(b)C_1(c)C_1(abc)$.

Natural number encoding

In our sequence encoding, we need a binary representation of natural numbers used to indicate gaps between characters in an encoded word. For any natural number n when the upper-bound on n is undefined in advance, the *Elias code* is usually used [78].

The Elias code of any natural number n denoted as $E(n)$ starts with exactly $\lfloor log_2(n) \rfloor$ zero bits followed by the actual binary representation of the natural number n. In this way, the Elias code length is equal to $2\lfloor log_2(n) \rfloor + 1$ bits which makes the encoding universal in the sense that when the upper-bound of n is unknown in advance, the Elias code length is at most twice as long as the optimal code length. In the Elias coding, the larger the value of n is the longer the code length $|E(n)|$ is, therefore, short gaps is encoded more succinct than long gaps.

Example 10 (Elias encoding)**.** *An example of Elias codes is depicted in Figure 5-3 where the Elias codes of the first eight natural numbers are shown. The number 8 has the Elias code as* $E(8) = 0001000$ *starting with* $\lfloor log_2(n) \rfloor = 3$ *zeros and followed by the binary representation* 1000 *of the number n=8*

Decoding a binary string containing several Elias codes is simple. In fact, the decoder first reads the leading zero bits until it reaches a one bit. At this moment, it knows how many more bits it needs to read to reach the end of the current Elias code. This process is repeated for every block of Elias code to decode the binary string completely.

Example 11 (Elias decoding)**.** *The binary string* 000100000100 *can be decoded as follows: the decoder reads the first 3 zero bits, it knows that it needs to read 4 more bits to finish the current block. The obtained Elias code is decoded as the number 8. It continues to read the following 2 zero bits and reads another 3 bits to get the complete representation of the next number which is decoded as 4 in this case.*

Sequence encoding

Given a dictionary D, a sequence S is encoded by replacing instances of dictionary words in the sequence by pointers. A pointer p replacing an instance of a word w in a sequence S is a sequence of bits starting by the codeword $C(w)$ followed by a list of Elias codes of the gaps indicating the difference between positions of consecutive characters of the instances of word in S. In the case the word is a singleton, the pointer contains only the codeword of the corresponding singleton.

Example 12 (pointers)**.** *In the sequence* $S = abcabdcaebc$ *three instances of the word* $w = abc$ *at positions* $(1, 2, 3)$*,* $(4, 5, 7)$ *and* $(8, 10, 11)$ *are underlined. If the word abc already exists in the dictionary with the codeword* $C(w)$ *then the three occurrences can be replaced by three pointers* $p_1 = C(w)E(1)E(1)$*,* $p_2 = C(w)E(1)E(2)$ *and* $p_3 = C(w)E(2)E(1)$*.*

A sequence encoding can be defined as follows:

Definition 7 (Sequence Encoding)**.** *Given a dictionary, a sequence encoding of* S *is a replacement of instances of dictionary words by pointers.*

The encoding in Definition 7 is complete if all characters in the sequence S are encoded. In this work, we consider only complete encoding. In an encoding C of a sequence S, the usage of a word w denoted as $f_C(w)$ is defined as the number of times the word w is replaced by a pointer plus the number of times the word is present in the binary representation of the dictionary.

Example 13 (Sequence encoding)**.** *In Figure 5-2, two dictionaries* D_1 *and* D_2 *are created based upon two encodings* C_1 *and* C_2 *of the sequence* $S = abcabdcaebc$*. The first encoding* C_1 *replaces three occurrences of the word abc in the sequence* S *by pointers. Therefore, the usage of abc in that encoding is counted as the number of pointers replacing abc plus the number of the occurrences of abc in the dictionary, thus,* $f_{C_1}(abc) = 4$*. Meanwhile,*

although a is not replaced by any pointers it is present twice in the binary representation of the dictionary, so $f_{C_1}(a) = 2$. *Similarly, the usages of the other words are shown in the same figure.*

For every word w, the binary representation of the codeword $C(w)$ depends on its usage in the encoding. Denote $F_C = \sum_{w \in D} f_C(w)$ as the sum of the usages of all dictionary words in an encoding C. Relative usages of every word w defined as $\frac{f_C(w)}{F_C}$ which can be considered as a probability distribution defined on the space of all dictionary words because $\sum_{w \in D} \frac{f_C(w)}{F_C} = 1$.

According to [30], there exists a prefix-free encoding $C(w)$ such that the codeword length $|C(w)|$ is proportional to the entropy of the word, i.e. $|C(w)| \sim -\log \frac{f_C(w)}{F_C}$, i.e. shorter codewords are assigned to words with more usage. Such encoding is optimal over all encodings resulting in the same usage distribution of the dictionary words [30]. When the dictionary contains only singletons, the aforementioned encoding corresponds to the Huffman code [78]. In this work, we denote Huffman code as C_0 and consider the data in this encoding as the uncompressed representation of the data. In Figure 5-2 the second encoding corresponds to the Huffman code.

Sequence decoding

In this subsection, we discuss the decoding algorithm for an encoded sequence. First, we show how to read the content of a dictionary from its binary representation. A binary representation of a dictionary can be decoded as follows:

1. Read codewords of all singletons until encountering a duplicate of any singleton codeword.

2. Step by step read codewords of every non-singleton w by reading the contents of w (a sequence of familiar codewords of singletons) until reaching a completely unseen codeword $C(w)$ which is considered as the codeword of w in the dictionary.

Example 14 (Dictionary decoding)**.** *The dictionary* D_1 *in Figure 5-2 has the binary representation* $C_1(a)C_1(b)C_1(c)C_1(d)C_1(e)C_1(a)C_1(b)C_1(c)C_1(abc)$. *The decoder starts by reading codewords of all singletons* a, b, c, d *and* e. *It stops when a repeat of a codeword of a singleton is encountered, in this particular case, when it sees a repeat of* $C_1(a)$. *The de-*

coder knows that the codeword corresponds to the beginning of a non-singleton so it continuously reads the following codewords of singletons until reaching a never-seen-before codeword $C_1(abc)$. The latter codeword corresponds to the non-singleton abc in the dictionary.

Given the dictionary, a sequence can be decoded by reading every block of the binary string corresponding to a word replaced by a pointer. Each block is read as follows:

1. Read the codeword $C(w)$ and refer to the dictionary to get information about the word w.
2. If the word w is a singleton then it continues reading the next block. Otherwise, it uses the Elias decoder to get $|w| - 1$ gap numbers before continuing with the next block.

The following example shows how to decode the sequence $C_1(abc)$ $E(1)$ $E(1)$ $C_1(abc)$ $E(1)$ $E(2)$ $C_1(d)$ $C_1(abc)$ $E(2)$ $E(1)$ $C_1(e)$ with the help of the dictionary D_1:

Example 15 (Sequence decoding)**.** *The decoder first reads $C_1(abc)$ then it refers to the dictionary and knows that the word length is three, therefore, it reads two numbers by using the Elias decoder. The decoder continues reading the next block $C_1(abc)$ $E(1)$ $E(2)$ in the same way to decode another instance of abc. After that it reaches the codeword $C_1(d)$; a reference to the dictionary tells the decoder that there is no following gap number so the decoder continues to read the next blocks in a similar way to decode the last instance of abc and the singleton e.*

Data Description Length

Denote $g_C(w)$ as the cost of encoding the gaps by the Elias code of the word w in an encoding C. It is important to notice that the gap cost of a singleton is always equal to zero. The description length of the database $\mathfrak{S}$ encoded by the encoding C is calculated as follows:

$$L^C(\mathfrak{S}) = \sum_{w \in D} (|C(w)| * f_C(w) + g_C(w)) \tag{5.1}$$

$$= \sum_{w \in D} \left(\log \frac{F_C}{f_C(w)} * f_C(w) + g_C(w) \right) \tag{5.2}$$

5.5 Problem Definition

We denote $L_D^{C_D^*}(\mathfrak{S})$ as the length of the database $\mathfrak{S}$ in the optimal encoding C_D^* when the dictionary D is given. The problem of finding compressing patterns is formulated as follows:

Definition 8 (Compressing Sequences Problem)**.** *Given a sequence database $\mathfrak{S}$, find an optimal dictionary D^* and also optimal the encoding $C_{D^*}^*$ that use words in the dictionary D^* to encode the database $\mathfrak{S}$ such that $D^* = argmin_D L_D^{C_D^*}(\mathfrak{S})$.*

In order to solve the *Compressing Sequences Problem* we need to find at the same time the optimal dictionary D^* and the optimal encoding C_D^* that uses the dictionary D^* to encode the database $\mathfrak{S}$.

5.6 Complexity Analysis

This section discusses the complexity of the mining Compressing Sequences Problems. Finding a dictionary that compresses the database most is equivalent to finding a set of patterns that gives the most compression benefit defined as the difference between database description length before and after compression. The following theorem shows that even finding a dictionary containing all the singletons and one non-singleton pattern that gives the most compression benefit is inapproximable:

Theorem 7. *Finding the most compressing pattern is inapproximable.*

In order to prove Theorem 7, we reduce the most compressing pattern problem to the *Maximum Tile in Database problem* [27]. Given an itemset database $\mathfrak{D} = \{T_1, T_2, \cdots, T_n\}$, where every T_i is an itemset defined over an alphabet $\sum = \{a_1, a_2, \cdots, a_m\}$. The area of an itemset $I \subset \sum$ denoted as $A(I)$ is calculated as the size of I multiplying by the frequency of I in the database. The maximum tile problem looks for the itemset having the largest area. Mining the maximum tile is equivalent to finding the maximum clique in a bipartite graph known as an inapproximable problem in the literature [2].

From the itemset database $\mathfrak{D}$ we create a sequence database $\mathfrak{S} = \{S_1, S_2, \cdots, S_n\}$ as follows. First, distinct symbols $b_1, b_2, \cdots, b_M$ are added to $\sum$ to obtain a new alphabet $\sum^+$. Each transaction $T_i \in D$ is sorted increasingly according to any lexicographical order defined over $\sum^+$. Assume that T_i has the form $a_{i_1}, a_{i_2}, \cdots, a_{i_k}$ after sorting, therefore, a sequence

S_i is created as such $S_i = (a_{i_1}, 1), (a_{i_2}, 2), \cdots, (a_{i_k}, k)$. Besides, in the database $\mathfrak{S}$ we add an additional sequence S_{n+1} such that it contains all the symbols in $\{b_1, b_2, \cdots, b_M\}$ sorted increasing according to the lexicographical order. Let $N > 1$ be the sum of the lengths of all sequences except the last sequence in $\mathfrak{S}$.

In the Huffman encoding C_0 of $\mathfrak{S}$ using only singletons the description length of $\mathfrak{S}$ is:

$$\begin{aligned} L^{C_0}(\mathfrak{S}) &= \sum_{i=1}^{m} f_{C_0}(a_i) \log \frac{F_{C_0}}{f_{C_0}(a_i)} + \\ &\quad \sum_{i=1}^{M} f_{C_0}(b_i) \log \frac{F_{C_0}}{f_{C_0}(b_i)} \\ &= F_{C_0} \log F_{C_0} - \sum_{i=1}^{m} f_{C_0}(a_i) \log f_{C_0}(a_i) \\ &\quad -2M \end{aligned}$$

Let $P = a_{i_1} a_{i_2} \cdots a_{i_{|P|}}$ be any non-singleton word with $|P|$ characters and let C_P be an encoding that use a dictionary D_P containing only one non-singleton P to encode the data $\mathfrak{S}$ by replacing $f_{C_P}(P) - 1$ occurrences of P in the database. The description length of the database $\mathfrak{S}$ is:

$$\begin{aligned} L^{C_P}(\mathfrak{S}) &= \sum_{i=1}^{m} f_{C_P}(a_i) \log \frac{F_{C_P}}{f_{C_P}(a_i)} \\ &\quad + \sum_{i=1}^{M} f_{C_P}(b_i) \log \frac{F_{C_P}}{f_{C_P}(b_i)} \\ &\quad + f_{C_P}(P) \log \frac{F_{C_P}}{f_{C_P}(P)} + g_{C_P}(P) \\ &= F_{C_P} \log F_{C_P} \\ &\quad - \sum_{i=1}^{m} f_{C_P}(a_i) \log f_{C_P}(a_i) - 2M \\ &\quad - f_{C_P}(P) \log f_{C_P}(P) + g_{C_P}(P) \end{aligned}$$

We first prove two supporting lemmas from which Theorem 7 is a direct consequence.

Lemma 16. *If M is chosen such that $F_{C_0} > N^8 + N$ then:*

$$0.5 \log \frac{F_{C_0} - N}{N^8} \leq \frac{L^{C_0}(\mathfrak{S}) - L^{C_P}(\mathfrak{S})}{|P| f_{C_P}(P)} \leq 3 \log F_{C_0}$$

Proof. First since function $\frac{x}{\log x}$ is increasing for any $x > 2$ so we have a support inequality $\frac{x}{\log x} > \frac{y}{\log y}$ for any $x > y > 2$.

Since $F_{C_0} = F_{C_P} + |P|f_{C_P}(P) - |P| - f_{C_P}(P)$ we have $F_{C_0} > F_{C_P}$ and $|P|f_{C_P}(P) > F_{C_0} - F_{C_P} > \frac{|P|f_{C_P}(P)}{2}$. From which we first imply that:

$$\begin{aligned} F_{C_0}\log F_{C_0} - F_{C_P}\log F_{C_P} &\geq \frac{|P|f_{C_P}(P)\log F_{C_P}}{2} \\ &\geq \frac{|P|f_{C_P}(P)\log\left(F_{C_0} - N\right)}{2} \end{aligned}$$

Moreover, since $F_{C_0} > F_{C_P}$ we have $\frac{F_{C_0}}{\log F_{C_0}} > \frac{F_{C_P}}{\log F_{C_P}}$ from which we further imply that:

$$\begin{aligned} (F_{C_0} - F_{C_P})(\log F_{C_0} + \log F_{C_P}) \geq \\ F_{C_0}\log F_{C_0} - F_{C_P}\log F_{C_P} \\ 2|P|f_{C_P}(P)\log F_{C_0} \geq F_{C_0}\log F_{C_0} - F_{C_P}\log F_{C_P} \end{aligned}$$

Besides, $f_{C_0}(a_i) = f_{C_P}(a_i)\ \forall a_i \notin P$, $f_{C_P}(P) > f_{C_0}(a_i) - f_{C_P}(a_i) > 0\ \forall a_i \in P$ and $f_{C_0}(a_i) < N$. Therefore, we have:

$$\begin{aligned} 0 \geq \sum_{i=1}^{m} f_{C_P}(a_i)\log f_{C_P}(a_i) - \sum_{i=1}^{m} f_{C_0}(a_i)\log f_{C_0}(a_i) \geq \\ \sum_{a_i \in P}(f_{C_P}(a_i) - f_{C_0}(a_i))(\log f_{C_P}(a_i) + \log f_{C_0}(a_i)) \geq \\ -2|P|f_{C_P}(P)\log N \end{aligned}$$

Moreover, since the gaps value always less than N, we have $0 > -g(P) > -2|P|f_{C_P}(P)\log N$ and $|P|f_{C_P}(P)\log F_{C_0} > f_{C_P}(P)\log f_{C_P}(P) > 0$. Sum up all the last obtained inequalities, we have:

$$0.5\log\frac{F_{C_0} - N}{N^8} \leq \frac{L^{C_0}(\mathfrak{S}) - L^{C_P}(\mathfrak{S})}{|P|f_{C_P}(P)} \leq 3\log F_{C_0}$$

from which the lemma is proved □. □

Lemma 17. *If there is an algorithm approximating the best compressing pattern of $\mathfrak{S}$ within a constant factor α in polynomial time then there exists a constant factor β such that we can approximate the maximum tile of the database $\mathfrak{D}$ within a constant factor β.*

Proof. Let denote P^* as the maximum tile of the database $\mathfrak{D}$. Let P be the pattern that

approximates the best compressing pattern of $\mathfrak{S}$ within the constant factor α. We have:

$$L^{C_0}(\mathfrak{S}) - L^{C_P}(\mathfrak{S}) \geq \alpha(L^{C_0}(\mathfrak{S}) - L^{C_{P^*}}(\mathfrak{S}))$$

Based upon the results in Lemma 16, we can imply that:

$$3|P|f_{C_P}(P)\log F_{C_0} \geq 0.5\alpha|P^*|f_{C_{P^*}}(P^*)\log\frac{F_{C_0}-N}{N^8}$$

If M is chosen such that $F_{C_0} > N^{16} + 2N$, we have $\log\frac{F_{C_0}-N}{N^8} > \frac{\log F_{C_0}}{2}$, from which we further imply that:

$$|P|f_{C_P}(P) \geq \beta|P^*|f_{P^*}(P^*)$$

where $\beta = \frac{\alpha}{12}$ from which the lemma is proved. □ □

It is obvious that Theorem 7 is a direct corollary of Lemma 17 because the reduction can be done in polynomial time of the size of the database (M is chosen such that F_{C_0} is a polynomial of the size of the data, in this case $F_{C_0} > N^{16} + 2N$). A direct corollary of Theorem 7 is that the *Compressing Sequences Problem* is NP-Hard:

Theorem 8. *The* compressing pattern problem *is NP-hard.*

5.7 Algorithms

This section discusses two heuristic algorithms inspired by the idea of the Krimp algorithm to solve the compressing pattern mining problem. Before explaining these algorithms we first explain how to compress a sequence database using a single pattern as this procedure is used in both algorithms as a subtask.

Compress a database by a pattern

Since mining compressing patterns is NP-hard, the heuristic solution greedily chooses the next pattern that gives the best compression benefit when added to the dictionary. Thus as a subtask of the greedy selection we need to evaluate the compression benefit of adding a given non-singleton pattern. This step can be performed by considering the following greedy encoding of the database $\mathfrak{S}$ using a pattern P.

Algorithm 7 Compress($\mathfrak{S}|P$)

1: **Input**: A sequence database $\mathfrak{S} = \{S_1, S_2, \cdots, S_n\}$ and a pattern P
2: **Output**: the compress benefit of adding P to the dictionary D
3: $L^{C_0}(\mathfrak{S}) \longleftarrow$ the original length of data
4: **for** $S_i \in \mathfrak{S}$ **do**
5: **while** S_i has an instance of P **do**
6: $s \longleftarrow$ minGapMatch(S_i, P)
7: Replace s by a pointer to P in the dictionary
8: **end while**
9: **end for**
10: $L(\mathfrak{S}|D \bigcup \{P\}) \longleftarrow$ length of the data after adding P
11: Return $L^{C_0} - L(\mathfrak{S}|D \bigcup \{P\})$

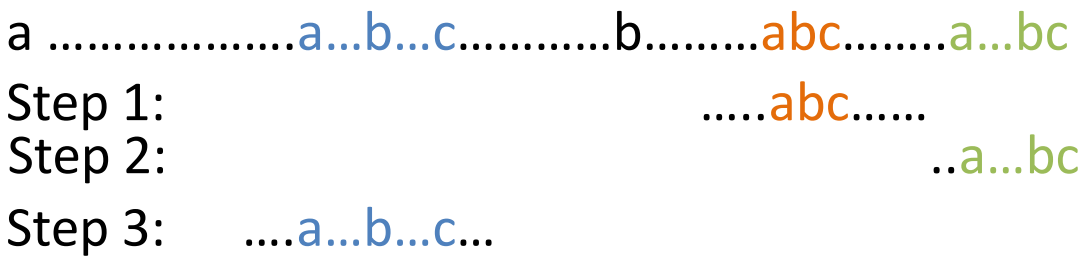


Figure 5-4: An example of the greedy encoding of the sequence S by the pattern $P = abc$. In every step it picks the match of P in S that has the minimum gap cost in the sequence S and replaces it with a pointer.

Algorithm 7 looks for instances of P in S such that the positions of the characters in the match are close to each other. Intuitively, those matches give shorter encodings. Therefore, for every individual sequence S in the database it first looks for a match of P in S having the minimum cost to encode the gaps between consecutive characters of the match (line 6). Subsequently, this match is replaced with a pointer and is removed from the sequence (line 7). This step is repeated to find any other matches of P in S. The same procedure is applied for encoding the other sequences in the database. The algorithm returns the compression benefit of adding the pattern P to the dictionary and encoding the database by the greedy encoding procedure.

Example 16. *As an example, Figure 5-4 shows every step of Algorithm 7 with a sequence S and a pattern $P = abc$. In the first step, the match with smallest gap cost is chosen and it is removed from the the sequence. The following two matches are chosen by the same procedure that looks for the match with minimum gap cost.*

An important task of the greedy encoding is to find the instance of $P = a_1 a_2 \cdots a_k$

Algorithm 8 minGapMatch(S_t, P)

Input: a sequence S_t and word $P = a_1 a_2 \cdots a_k$ and lists of positions $l_{a_1}, l_{a_2}, \cdots, l_{a_k}$
Output: the match with minimum gap cost
for $i = 1$ **to** k **do**
  for $j = 1$ **to** $|l_{a_i}|$ **do**
    $l^2_{a_i}[j] = \min_p \{l^2_{a_{i-1}}[p] + E(l^1_{a_i}[j] - l^1_{a_{i-1}}[p])\}$
  end for
end for
Return match with minimum cost

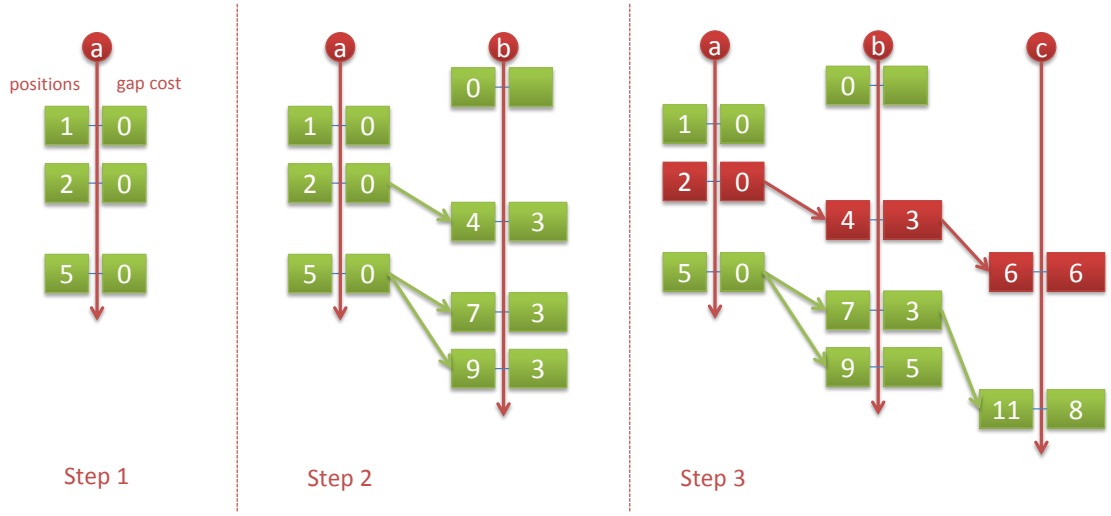


Figure 5-5: An example of dynamic programming algorithm to find the match of a pattern $P = abc$ with minimum gap cost in the sequence S

having the minimum gap cost. This task can be done by using a dynamic programming method as follows. Let $l_{a_1}, l_{a_2}, \cdots, l_{a_k}$ be the lists associated with the characters of the pattern P: The $j-th$ element of a list l_{a_i} contains two fields denoted as $l^1_{a_i}[j]$ and $l^2_{a_i}[j]$. The first field $l^1_{a_i}[j]$ contains a position of a_i in the sequence S and the second field $l^2_{a_i}[j]$ contains the gap cost of the match of the word $a_1 a_2 \cdots a_i$ with minimum gap cost given that the match must end at the position $l^1_{a_i}[j]$. Algorithm 8 finds the match of the word P with minimum gap cost by scanning through all the lists l_{a_i} for $i = 1, 2, \cdots, k$ and for the j-th element of the list l_i it calculates $l^2_{a_i}[j]$ by using the following formula:

$$l^2_{a_i}[j] = \min_p \{l^2_{a_{i-1}}[p] + E(l^1_{a_i}[j] - l^1_{a_{i-1}}[p])\} \tag{5.3}$$

, where $l^2_{a_1}[j] = 0$ for $j = \overline{1, 2, \cdots, |l_{a_1}|}$.

Example 17 (match with minimum gap cost)**.** *Figure 6-5 illustrates the basic steps of Algorithm 8 finding the match with minimum gap cost of the word* $w = abc$ *in the sequence* $S = (b,0)(a,1)(a,2)(b,4)(a,5)(c,6)(b,7)(b,9)(c,11)$.

Step 1: l_a *contains 3 elements with* $l_a^1[1] = 1$, $l_a^1[2] = 2$ *and* $l_a^1[3] = 5$ *indicating the positions of a in the sequence S. First, we can initialize the second field of every element of the list* l_a *to zero.*

Step 2: l_b *contains 4 elements with* $l_b^1[1] = 0$, $l_b^1[2] = 4$, $l_b^1[3] = 7$ *and* $l_b^1[4] = 9$ *indicating the positions of b in the sequence S. According to formula 5.3 we can calculate the second field of every element of the list* l_b *as follows, for instance for* $l_b^2[2]$*:*

$$\begin{aligned} l_b^2[2] &= \min_p \{l_a^2[p] + E(l_b^1[2] - l_a^1[p])\} \\ &= l_a^2[2] + E(l_b^1[2] - l_a^1[2]) \\ &= 3 \end{aligned}$$

We draw an arrow connecting $l_a[2]$ *and* $l_b[2]$ *in order to keep track of the best match so far. The value of* $l_a^2[3]$ *and* $l_a^2[4]$ *can be calculated in a similar way.*

Step 3: l_c *contains 2 elements with* $l_c^1[1] = 6$ *and* $l_c^1[2] = 11$ *indicating the positions of c in the sequence S. The values of* $l_c^2[1]$ *and* $l_c^2[2]$ *can be obtained in the same way as in step 2. Among them* $l_c^2[1] = 6$ *bits is smallest so the match of abc in S with minimum gap cost corresponds to the instance of abc at positions* $(2,4,6)$.

SeqKrimp, a Krimp-based algorithm for sequence Database

In this subsection, we introduce an algorithm for mining compressing patterns from a sequence database similar to Krimp for itemset data. The SeqKrimp described in Algorithm 9 consists of two phases. In the first phase, a set of candidate patterns is generated by using a frequent closed sequential patterns mining algorithm (line 3).

In the second phase, the SeqKrimp algorithm chooses a good set of patterns from the set of candidates based upon a greedy procedure. It first calculates the compression benefit of adding a pattern $P \in \mathbf{C}$ to the current dictionary. The compression benefit is calculated with the help of Algorithm 7. The pattern P^* with the most additional compression benefit is included in the dictionary. Additionally, once P^* has been chosen, Algorithm 7 is used to replace all the instances of P^* in the data D by pointers to P^* in the dictionary. These

Database	Dictionary D	Candidate set C	Benefit (P)
(a,1)(b,2)(c,3) (a,4)(b,5)(c,6) (a,1)(b,2)(c,3)(a,4)(b,5)(c,6) (a,1)(b,2)(c,3)(a,4)(b,5) (a,1)(b,2)(c,3)(a,4)(b,5) (a,1)(b,2)(c,3) (a,4)(b,5)(c,6) (a,1)(b,2)(c,3) (a,4)(b,5)(c,6)	$w_1 = a\ w_2 = b$ $w_3 = c$	ab abc bc	-2 11 -14

Step 1: abc is chosen

Database	Dictionary D	Candidate set C	Benefit (P)
……. ……. ……(a,4)(b,5) ……(a,4)(b,5) ……. …….	$w_1 = a\ w_2 = b$ $w_3 = c$ $w_4 = abc$	ab cb	-4 -∞

Step 2: stop as long as no positive additional benefit of adding a new pattern

Figure 5-6: An example illustrates how the SeqKrimp algorithm works

actions are repeated as long as the candidate set **C** is not empty and there is still additional positive compression benefit to add a pattern.

Example 18 (SeqKrimp)**.** *As an example, Figure 5-6 shows each step of the SeqKrimp algorithm for a database and a candidate set. In the first step, the compression benefit of adding every candidate is calculated. The word abc is chosen because it gives the best additional compression benefit among the candidates. When abc is chosen the database is updated by replacing every instance of abc by a pointer. Subsequently, the compression benefit of the remaining candidates is recalculated accordingly. Finally, in step 2, since there is no additional compression benefit of adding a new pattern, the algorithm stops.*

SeqKrimp suffers from the dependency on the candidate generation step that is very expensive for low minimum support thresholds. Even for moderate-size datasets state of the art algorithms for extracting frequent or closed patterns from sequence database such as PrefixSpan [53] or BIDE algorithm [77, 26] are very time-consuming.

Direct Mining Of Compressing Patterns

This subsection discusses a direct algorithm for mining compressing patterns. In particular, GoKrimp depicted in Algorithm 10 directly looks for the next most compressing pattern P^*. When a pattern has been obtained, the Compress procedure in Algorithm 7 is used to replace every instance of this pattern in the database by a pointer. These actions are

Algorithm 9 SeqKrimp($\mathfrak{S}$)

Input: Database $\mathfrak{S}$
Output: compressing patterns
C $\longleftarrow$ **GetCandidate**()
$D \longleftarrow \sum$
while $C \neq \emptyset$ and Benefit(P^*) > 0 **do**
  for $P \in$ **C do**
    Benefit(P) $\longleftarrow$ **Compress**($\mathfrak{S}|P$)
  end for
  $P^* \longleftarrow argmax_P$Benefit($P$)
  if $Benefit(P^*) \leq 0$ **then**
    break
  end if
  $D \longleftarrow D \cup \{P^*\}$
  C $\longleftarrow$ **C** $\setminus \{P^*\}$
  Using Algorithm 7 to replace all instances of P^* in $\mathfrak{S}$ by pointers
end while
Return D

Algorithm 10 GoKrimp($\mathfrak{S}$)

Input: Database $\mathfrak{S} = \{S_1, S_2, \cdots, S_n\}$
Output: the set of compressing patterns
$D \longleftarrow \sum$
while Benefit(P^*)> 0 **do**
  $P^* \longleftarrow$ **GetNextPattern**($\mathfrak{S}$))
  if $Benefit(P^*) \leq 0$ **then**
    break
  end if
  $D \longleftarrow D \cup \{P^*\}$
  Using Algorithm 7 to replace all instances of P^* in $\mathfrak{S}$ by pointers
end while
return D

repeated until there is no more additional compression benefit of adding a new pattern.

The most important subtask of the GoKrimp algorithm is a greedy procedure to obtain the next good compressing pattern from the data. *GetNextPattern*($\mathfrak{S}$) depicted in Algorithm 11 step by step extends every frequent event until no more additional compression benefit can be obtained. When all the extensions have been obtained the algorithm chooses among them the one with highest compression benefit to return as an output pattern.

The evaluation of each extension is very time consuming because it involves multiple searches for minimum gap matches of the extension in the database. Therefore the set of events chosen to extend a pattern is limited to the set of events being related to the occur-

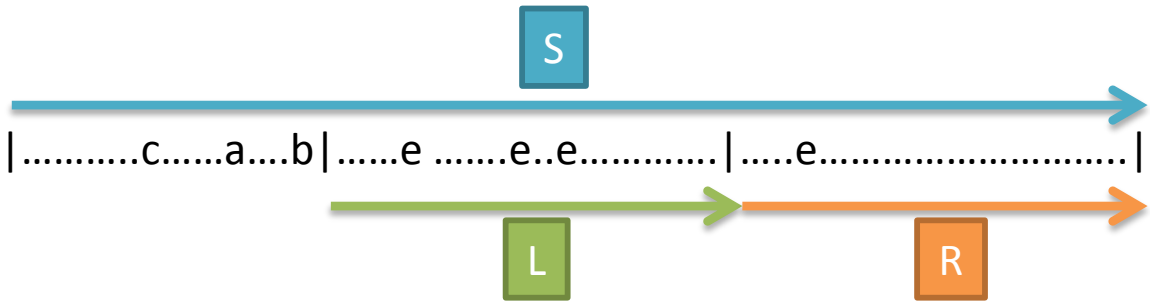


Figure 5-7: An example of how dependency test is carried out. If the event e is independent from the pattern $P = cab$ then it must occur in two equal-length subintervals L and R with the same chance.

Algorithm 11 GetNextPattern($\mathfrak{S}$)

Input: Database $\mathfrak{S} = \{S_1, S_2, \cdots, S_n\}$
Output: P approximation of the best compressing pattern
$P \longleftarrow \{\emptyset\}$
$F \longleftarrow$ frequent events
while (true) **do**
  for $e \in F$ **do**
    Benefit(e) $\longleftarrow$ **Compress**($D|P.e$)
  end for
  $e^* \longleftarrow argmax_e$ Benefit(e)
  if Benefit(e^*) < 0 **then**
    Break
  end if
  $P \longleftarrow P.e^*$
  $\mathfrak{S} \longleftarrow \mathfrak{S}$ projected to e^*
  $F \longleftarrow$ related frequent events
end while
return P

rences of the given pattern. Indeed, the *GetNextPattern* algorithm adopts a dependency test to collect all the related events. Subsequently, the event when added to the given pattern giving the most compression benefit is chosen to extend that pattern. When an event has been chosen the database is projected to the event and the algorithm keeps extending the pattern as long as the extensions still adds more compression benefit.

In order to test the dependency between a pattern P and an event e we use the *Statistical Sign Test* [18]. Given m pairs of numbers $(X_1, Y_1)(X_2, Y_2), \cdots, (X_m, Y_m)$, denote N^+ as the number of pairs such that $X_i > Y_i$ for $i = 1, 2, \cdots, m$. If two sequences $X_1, X_2, \cdots, X_m$ and $Y_1, Y_2, \cdots, Y_m$ are generated by the same probability distribution then the test statistics N^+ follows a binomial distribution $B(0.5, m)$.

The sign test is applied to test the dependency between a pattern P and an event e as follows. For every sequence $S \in \mathfrak{S}$ and an event $c \in P$ denote $S(c)$ as the leftmost instance of c in S. Consider the interval right after the last position of $S(c)$ as illustrated in Figure 5-7. This interval is divided into two equal-length subintervals L and R. Denote the frequency of the event e in the two subintervals as L_e and R_e respectively. If the event e is independent from the occurrence of $S(c)$, we would expect that the chance e occurring in left and the right intervals is the same. Therefore, the number of sequences in which we observe $L_e > R_e$ can be used as a test statistics in the sign test for testing the dependency between the event e and the pattern c. The test is done for every event $c \in P$, an event e is considered as related to pattern P if it passes all the dependency tests regarding all the event belong to P. When a test has been done we keep log of the dependency results for reusing next time. In the next section, we empirically show that the dependency test speeds up the GoKrimp algorithm significantly while preserving the quality of the compressing patterns.

5.8 Experiments and Results

This section discusses results of experiments carried out several real-life and one synthetic dataset. We will compare the set of patterns produced by SeqKrimp and GoKrimp algorithms to the following baseline algorithms:

- BIDE: BIDE was chosen because it is a state of the art approach for closed sequential pattern mining. BIDE is also used to generate the set of candidates for SeqKrimp, i.e. an implementation of the **GetCandidate**(.) function in line 3 Algorithm 9.
- SQS: proposed recently by Tatti and Vreeken [71] for mining compressing patterns in sequence database
- pGOKRIMP : the prior version of the GoKrimp algorithm (denoted as pGoKrimp) published in our previous work [40]. We include pGOKRIMP in the comparison to demonstrate the effectiveness of the revised encoding adopted by the GOKRIMP algorithm.

We use seven different real-life datasets introduced in [52] to evaluate the proposed approaches in term of classification accuracy. Each dataset is a database of symbolic interval sequences with class labels. For our experiments the interval sequences are converted to

Method	Patterns			
SEQKRIMP	support vector machin real world machin learn data set bayesian network	state art high dimension larg scale futur selection experiment result	compon analysi sampl size supervis learn support vector loss function	solv problem kernel kernel kernel kernel model select train set loss loss
GOKRIMP	support vector machin real world machin learn data set bayesian network	state art high dimension reproduc hilbert space larg scale independ compon analysi	neural network experiment result sampl size supervis learn support vector	well known special case solv problem signific improv object function
SEARCH_SQS	support vector machin machin learn state art data set bayesian network	larg scale nearest neighbor decis tree neural network cross valid	featur select graphic model real world high dimension mutual inform	sampl size learn algorithm princip compon analysi logist regress model select
pGOKRIMP	machin lean learn learn algorithm algorithm algorithm algorithm data data data data set data set method method method	result show paper data problem problem set set model model gener	result result machine learn Perform perform paper propose machin kernel kernel	present algorithm such data learn learn show show Function function function

Figure 5-8: Patterns discovered by the SeqKrimp, GoKrimp, SQS and the pGoKrimp algorithm.

Table 5.2: Summary of Datatsets

Datasets	**Events**	**Sequences**	**Classes**
jmlr	787	75,646	NA
parallel	1,000,000	10,000	NA
aslbu	36,500	441	7
aslgt	178,494	3,493	40
auslan2	1800	200	10
pioneer	9766	160	3
context	25,832	240	5
skating	37,186	530	7
unix	295,008	11,133	10

event sequences by considering the start and end points of every interval as different events. A brief summary of the datasets is given in Table 5.2. All the benchmark datasets are available for download at the website [2].

Besides, other two datasets are also used for evaluating the proposed approaches in term of pattern interpretability. The first dataset JMLR contains 787 abstracts of the Journal of Machine Learning research. JMLR is chosen because the potential important patterns are easily interpreted. The second dataset is a synthetic one with known patterns. For this dataset we evaluate the proposed algorithms based on the accuracy of the set of patterns returned by each algorithms. These datasets along with the source code of the GoKrimp and the SeqKrimp algorithms written in Java are available for download at our project

[2] `http://www.mybytes.de/sipo.zip`

website[3]. Evaluation was done in a 4 x 2.4 Ghz, 4 GB of RAM, Fedora 10 / 64-bit station.

In summary, the proposed approaches are evaluated according to the following criteria:

1. *Interpretability* - to informally assess the meaningfulness and redundancy of the patterns.

2. *Run time* - to measure the efficiency of the approaches.

3. *Compression* - to measure how well the data is compressed.

4. *Classification accuracy* - to measure the usefulness of a set of patterns.

Pattern Interpretability

JMLR

Since descriptive pattern mining is unsupervised, it is very hard to compare different sets of patterns in the general case. However, for text data it is possible to interpret the extracted patterns. In this work, we compare different algorithms on the JMLR dataset.

For the GoKrimp algorithm, the significance level used in the sign test is set to 0.01 and the minimum number of pairs needed to perform a sign test is set to 25 as recommended in [18]. For the SeqKrimp algorithm the minimum support was set to 0.1 at which the top 20 patterns returned by each of these algorithm does not change when the minimum support is set smaller. Figure 5-8 shows the top 20 patterns from the JMLR dataset extracted by the SeqKrimp, the GoKrimp, the SQS and the pGoKrimp algorithm.

Comparing to the top 20 most frequent closed patterns depicted in Figure 6-1, these sets of patterns are obviously less redundant. The results of the GoKrimp, SeqKrimp and SQS are quite similar. Most of the patterns corresponds to well-known research topics in machine learning.

The pGoKrimp algorithm, i.e. a prior version of the GoKrimp algorithm returns a lots of uninteresting patterns being combinations of frequent events. A possible reason is that in contrast to the SQS and the GoKrimp algorithm, the pGoKrimp algorithm uses an encoding that does not punish gaps and it does not consider the usage of a pattern when assigning codeword to the patterns.

[3] `http://www.win.tue.nl/~lamthuy/gokrimp.htm`

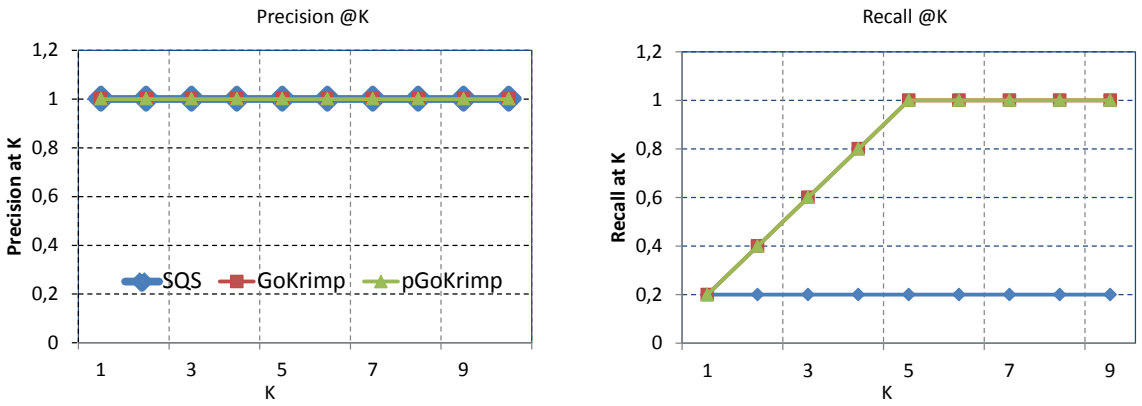


Figure 5-9: Precision and recall at K of the patterns discovered by the GoKrimp, SQS and the pGoKrimp algorithm in the *Parallel* dataset.

Parallel

Parallel is a synthetic dataset which mimics a typical situation in practice where the data stream is generated by five independent parallel processes. Each process P_i generates one event from the set of events $\{A_i, B_i, C_i, D_i, E_i\}$ in that order. In each step, the generator chooses one of five process uniformly at random and generates an event by using that process until the stream length is 1000000. For this dataset, we know the ground truth since all the sequences containing a mixture of events from different parallel processes are not the right patterns.

We get the first 10 patterns extracted by each algorithm and calculate the precision and recall at K. Precision at K is calculated as the fraction of the number of right patterns in the first K patterns selected by each algorithm. While the recall is measured as the fraction of the number of types of true patterns in the the first K patterns selected be each algorithm. For instance, if the set of the first 10 patterns contains only events from the set $\{A_i, B_i, C_i, D_i, E_i\}$ for a given i then the precision at $K = 10$ is 100% while the recall at $K = 10$ is 20%. The precision measures the accuracy of the set of patterns and the recall measures the diversity of the set of patterns.

For this dataset, the BIDE algorithm was not able to finish its running after a week even if the minimum support was set to 1.0. A reason is that all possible combination of the 25 events are frequent patterns. Therefore, the results of the BIDE and the SeqKrimp algorithm for this dataset are missing. Figure 5-9 shows the precision and the recall of the set of K patterns returned by the three algorithms SQS, GoKrimp and pGoKrimp when K

Run time in seconds

Datasets	Bide	SeqKrimp	SQS	GoKrimp
auslan2	0.85	1.0	1.0	0.40
aslbu	74.3	972	277	28
aslgt	73.7	1344	58501	1842
pioneer	11.4	65	15	9
skating	67.3	183	123	85
context	309	402	86	44
unix	1055	47111	84869	1824
jmlr	10	232	890	93
parallel	U/N	U/N	2066	342

The number of patterns

Datasets	Bide	SeqKrimp	SQS	GoKrimp
auslan2	128	4	13	4
aslbu	14620	52	195	67
aslgt	3472	56	1095	68
pioneer	5475	21	143	49
skating	3767	24	140	49
context	6760	15	138	33
unix	28477	75	1070	165
jmlr	4240	23	580	30
parallel	U/N	U/N	17	23

Figure 5-10: Running time in seconds and the number of patterns returned by each algorithm on nine datasets

(x-axis) is varied.

In term of precision all the algorithms are good because the top patterns selected by each of them are all correct ones. However, in term of recall the SQS algorithm is worse than the other two algorithms. A possible explanation is that the SQS algorithm uses an encoding that does not allow encoding interleaving patterns. For this particular dataset where interleaved patterns are observed frequently the SQS algorithm misses patterns that are interleaved with the chosen patterns.

Running Time

We perform experiments to compare running time of different algorithms. For the SeqKrimp algorithm and the BIDE algorithm, we first fix the minimum support parameter to the smallest values used in the experiment where patterns are used as features for classification tasks in subsection 5.8. The SQS algorithm are parameter-free while the GoKrimp algorithm uses standard parameter setting recommended for *Sign Test* so their running time only depends upon the size of the data.

The experimental result is illustrated in Figure 6-8. As we can see in this figure, the SeqKrimp algorithm is always slower than the BIDE algorithm because it needs an extra procedure to select compressing patterns from the set of candidates returned by the BIDE algorithm. The GoKrimp algorithm is 1-2 orders of magnitude faster than SeqKrimp or the BIDE algorithms, giving results "to go" when in a hurry. The SQS algorithm is very fast on small datasets (though still slower than GoKrimp), however, it is several times slower than the other algorithms on larger datasets such as the unix and the aslgt.

Data	Algorithms	Naïve Bayes	Random Forest	J48	VFI	Linear SVM	RBF SVM	Kstar	IB1	Best
auslan2	BIDE	22.50	29.00	25.50	22.50	26.50	23.50	25.50	25.50	29.00
	SEQKRIMP	22.00	30.50	26.50	24.50	28.00	22.50	24.00	27.00	30.50
	GOKRIMP	20.50	29.00	26.00	24.00	29.50	23.50	26.00	26.00	29.50
	SINGLETONS	22.00	29.00	27.00	23.50	29.00	22.00	25.00	26.00	29.00
aslbu	BIDE	48.07	58.27	50.56	31.06	59.18	50.34	59.41	59.18	59.41
	SEQKRIMP	52.15	60.31	51.02	26.98	59.86	52.07	59.18	57.59	60.31
	GOKRIMP	52.38	54.87	50.34	24.26	59.86	53.28	59.18	58.27	59.86
	SINGLETONS	51.24	54.89	50.79	25.17	58.50	51.24	59.64	57.14	59.64
aslgt	BIDE	70.25	75.06	69.91	54.33	81.82	79.38	74.03	73.08	81.82
	SEQKRIMP	72.23	73.35	69.96	56.94	82.27	79.64	75.23	73.71	82.27
	GOKRIMP	72.17	76.35	70.59	56.65	81.90	80.36	76.00	74.83	81.90
	SINGLETONS	71.68	76.92	69.82	57.05	81.04	79.38	75.63	73.94	81.04
pioneer	BIDE	96.87	95.625	94.37	93.75	99.37	95.62	98.12	98.75	99.37
	SEQKRIMP	100.0	98.75	99.37	93.12	100.0	100.0	90.37	93.37	100.0
	GOKRIMP	100.0	99.37	99.37	95.12	100.0	100.0	99.37	99.37	100.0
	SINGLETONS	100.0	96.67	99.37	93.75	100.0	100.0	98.75	99.37	100.0
skating	BIDE	60.75	57.73	54.33	50.37	63.77	57.33	48.49	47.16	63.77
	SEQKRIMP	73.58	73.58	72.45	66.03	74.15	74.33	64.52	61.69	74.33
	GOKRIMP	67.54	59.81	62.45	57.92	67.54	66.98	53.58	52.64	67.54
	SINGLETONS	61.88	58.67	55.09	51.69	64.71	58.67	49.24	61.25	64.71
context	BIDE	77.50	70.83	75.00	71.25	74.56	70.41	70.41	61.66	77.50
	SEQKRIMP	79.58	72.91	77.91	74.58	76.25	73.75	72.08	65.00	79.58
	GOKRIMP	80.83	75.41	80.00	77.91	82.08	78.75	74.58	72.18	82.08
	SINGLETONS	78.75	68.33	75.41	74.16	76.66	74.16	67.50	61.25	78.75
unix	BIDE	42.15	72.15	71.25	29.08	74.05	44.43	67.71	63.36	74.05
	SEQKRIMP	54.80	73.81	72.09	37.48	74.26	45.90	70.25	65.85	74.26
	GOKRIMP	54.09	73.88	72.05	37.70	74.33	45.87	70.39	65.87	74.52
	SINGLETONS	57.77	73.90	72.05	38.06	74.52	44.43	70.77	66.35	74.52

Figure 5-11: Classification results with patterns used as binary attributes. The number of patterns used in each algorithm were balanced.

Figure 6-8 also reports the number of patterns returned by each of algorithms. The BIDE algorithm as usual returns a lot of patterns depending on the minimum support parameter. When this parameter is set low the number of patterns returned by the BIDE algorithm is even larger than the size of the datasets. On the other hand, the SeqKrimp, the SQS and the GoKrimp algorithm returned just a few patterns. The total number of patterns seems to be dependent only on the size of the datasets.

Classification Accuracy

Classification is one of the most important applications of pattern mining algorithms. In this subsection, we discuss results of using the extracted patterns, together with all singletons, as binary attributes for classification tasks. We'll refer to the approach of using only singletons as features as *Singletons*. This algorithm together with the BIDE algorithm are considered as baseline approaches in our comparison.

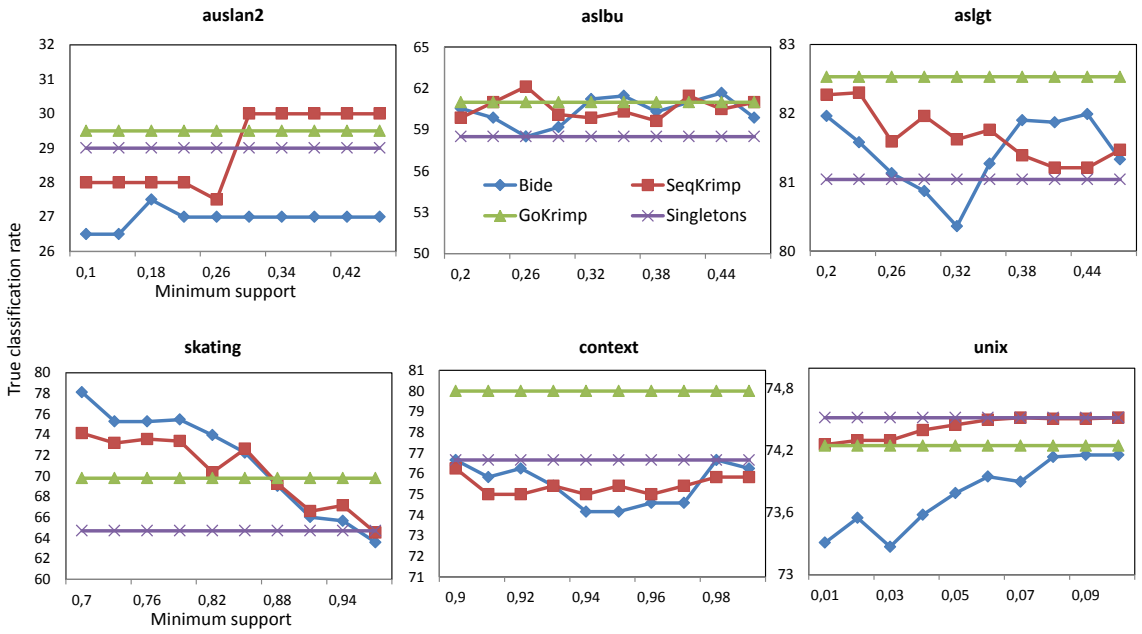


Figure 5-12: Classification results with linear SVM when using the full set of patterns and varying minimum support.

We use the implementations of classification algorithms available in the Weka package[4]. All the parameters are set to default values. The classification results were obtained by averaging the classification accuracy over 10 folds cross-validations. In the experiments, there are two important parameters: the minimum support value for the BIDE and the SeqKrimp algorithm, and the classification algorithm used to build the classifiers.

Therefore, we perform two different experiments to evaluate the proposed approaches when these parameters are varied. In the first experiment, the minimum supports were set to the smallest values reported in Figure 5-12. At first, the parameter K is set to infinite to get as many patterns as possible. In doing so, we obtain sets of patterns with different size and the patterns are ordered decreasingly according to the ranks defined by every algorithm. In order to make the comparison fair enough, the patterns at the end of each pattern set are removed such that all the sets have the same number of patterns being equal to the minimum number of patterns discovered by every algorithm. Moreover, different classifiers are used to evaluate the classification accuracy. This helps us to choose the best classifier for the next experiment.

Figure 5-11 shows the results of the first experiment. Eight different popular classifiers

[4] `http://www.cs.waikato.ac.nz/ml/weka/`

were chosen for classification. The numbers in each cell show the percentage of correctly classified instances. The last column in this figure summarizes the best result, i.e the highest number in each row. Besides, in each cell of this column, the highest value corresponding to the best classification result in a dataset is also highlighted.

The highlighted numbers in the last column show that the top patterns returned by the SeqKrimp and the GoKrimp algorithm are more predictive than the top patterns returned by the BIDE algorithms. On each dataset either SeqKrimp or GoKrimp achieved the best results. Besides, the highlighted numbers in each row show that the *Linear SVM classifier* is the most appropriate classifier for this type of data because it gives the best results in most of the cases.

In the next experiment, the minimum support parameter was varied to see how classification results change. Because the linear SVM classifier gave the best results in most of the datasets, we choose this classifier for this experiment. Figure 5-12 shows the results. Because the GoKrimp and Singletons features do not depend on minimum support settings, the results of these algorithms do not change across different minimum support settings and are shown as straight lines.

The results show that, in most of the datasets, adding more patterns to the singleton set gives better classification results. However, the benefit of adding more patterns is very sensitive to the minimum support settings. Especially, it varies significantly from one dataset to another.

Behavior of the BIDE algorithm is very unstable. For example, in the aslgt and the skating datasets adding more patterns, i.e lowering the minimum support, actually improves the classification results of the BIDE algorithm. However, in the auslan2, aslbu, context and unix datasets the effect of adding more patterns is very ambiguous. The behavior of the SeqKrimp algorithm is also very unstable as it uses patterns extracted by BIDE as candidate patterns. Therefore, in these cases, extra effort on parameter tuning is needed.

On the other hand, the classification results of the GoKrimp algorithm do not depend on minimum support. It is better than the singleton approach in most of the cases. It is also much better than the BIDE algorithm in dense datasets such as the context, the aslgt, and the unix data.

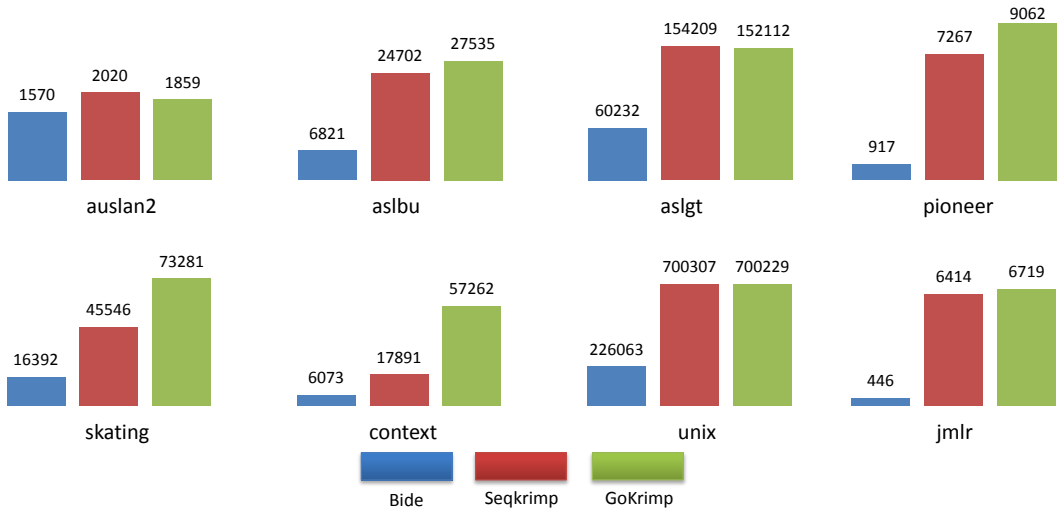


Figure 5-13: Compression benefit (in number of bits) when using the top patterns selected by each algorithm to compress the data.

Compressibility

We calculate the compression benefit of the set of patterns returned by every algorithm. In order to make the comparison fair, all sets of patterns have the same size, being equal to the minimum of the number of patterns returned by all algorithms. For the SeqKrimp and the GoKrimp algorithms the compression benefits were calculated as the sum of the compression benefit returned after each greedy step. For Closed patterns, compression benefit was calculated according to the greedy encoding procedure used in the SeqKrimp algorithm. For the SeqKrimp and the BIDE algorithm, the minimum support is fixed to the smallest values in the corresponding experiment shown in Figure 5-12. Compression benefit is measured as the number of bits saved when encoding the original data using the pattern set as the dictionary. Because the SQS algorithm uses different encoding for data before compression so we cannot compare the compressibility of that algorithm to ours in terms of bits (see below for a comparison by ratios).

Figure 5-13 shows the obtained results in eight different datasets (the result of the algorithm on the parallel dataset is omitted because both SeqKrimp and BIDE did not scale to the size of this dataset). As we expect, in most of the datasets, SeqKrimp and GoKrimp are able to find better compressing patterns than BIDE. Especially, in most of the large datasets such as *aslgt, aslbu, unix, context* and *skating* the differences between SeqKrimp, GoKrimp and BIDE are very significant. The GoKrimp algorithm is able to

	SQS	GoKrimp	GoKrimp*
auslan2	1.571	1.428	1.907
aslbu	1.155	1.123	1.284
aslgt	1.308	1.156	1.450
pioneer	1.302	1.171	1.243
skating	1.880	1.629	2.095
context	2.700	1.706	2.698
unix	2.230	1.638	1.880
jmlr	1.039	1.008	1.008
parallel	1.070	1.135	2.042

Figure 5-14: Compression ratio comparison of different algorithms

find compressing patterns with similar quality as the SeqKrimp algorithm in most of the datasets and is even better than the SeqKrimp algorithm in several cases such as in the pioneer, skating and context.

Finally we perform another experiment to compare the GoKrimp algorithm with the SQS algorithm based on compression ratio calculated by dividing the size of the data before compression with the size after compression. It is important to note that the compression ratio is highly dependent how we calculate the size of uncompressed data and how we choose the encoding for gaps. Therefore, in order to make the comparison fair the compression ratios were calculated when using the same uncompressed data representation. However, there is another practical issue of the comparison as follows.

The implementation of SQS uses an ideal code length for gaps. It assigns code length to a gap and a non-gap by considering the entropy of the gap or the non-gap. When the number of non-gaps dominates, which is actual the case in the experiments with our datasets, a non-gap can be assigned a codelength close to zero. This is an ideal case because in practice one cannot assign a codeword with length close to zero. In contrast, GoKrimp uses actual Elias codewords for gaps. Therefore there is a practical issue of comparing two algorithms one use the ideal code length and another use the actual code length. Therefore, for GoKrimp we calculate the ideal code length of a gap n as $\log n$, the result of this ideal case will be reported as GoKrimp* in the experiments.

Figure 5-14 shows the compression ratio of three algorithms on nine datasets. The

	GoKrimp with Sign Test			GoKrimp without Sign Test		
	Time (Seconds)	Compression ratio	# patterns	Time (seconds)	Compression ratio	# patterns
auslan2	0.40	1.428	4	1	1.420	3
aslbu	28	1.123	67	7414	1.169	117
aslgt	1842	1.156	68	10293	1.158	79
pioneer	9	1.171	49	822	1.214	88
skating	85	1.629	49	348	1.662	59
context	44	1.706	33	251	1.802	33
unix	1824	1.638	165	U/N	U/N	U/N
jmlr	93	1.008	30	537895	1.018	182
parallel	342	1.135	23	2296	1.135	23

Figure 5-15: The compression ratios of patterns by GoKrimp with and without sign test are almost the same but with the sign test the GoKrimp algorithm is much more efficient.

SQS algorithms show a better compression ratio in most of the case except for the parallel dataset when non-gap is not popular. For that dataset the effect of using ideal codelength is not visible. However, a version of GoKrimp with ideal code length for gaps gives better compression ratios than SQS in most of the cases. These results shows that variation of codeword length calculation can influence the compression ratio significantly. Therefore, interpretation of the results with compression ratios is quite hard in such cases.

Effectiveness of dependencies test:

In this subsection, we perform experiments to demonstrate the effectiveness of the dependency test proposed for speeding up the GoKrimp algorithm. We recall that the dependency test is proposed to avoid exhaustive evaluation of all possible extensions of a pattern. Once a test is done, the results of the test is kept for the next time so in the worst case the maximum number of tests is at most equal to the size of the alphabet. Besides, the set of related events to a given event is quite small compared to the size of the alphabet so the dependency test also helps to reduce the number of extension evaluations.

Figure 5-15 shows the running time of the GoKrimp algorithm with and without dependency test. It is obvious that the GoKrimp algorithm is much more efficient when dependency testing is used. More importantly, the compression ratio is almost the same in both cases. Therefore the dependency test helps speed up the GoKrimp algorithm significantly while preserving the quality of the pattern set in all the datasets. This result is consistent with an intuition that using patterns with unrelated events for compression does not result in good compression ratios.

Chapter 6

Mining Compressing Sequential Patterns in a Data Stream

Mining patterns that compress the data well was shown to be an effective approach for extracting meaningful patterns and solving the redundancy issue in frequent pattern mining. Most of the existing work in the literature however, consider mining compressing patterns from a static database of itemsets or sequences. These approaches require multiple passes through the data and do not scale up with the size of data streams. In this chapter, we study the problem of mining compressing sequential patterns from a data stream. We propose an approximate algorithm that needs only a single pass through the data and efficiently extracts a meaningful and non-redundant set of sequential patterns[1]. Experiments on three synthetic and three real-world large-scale datasets show that our approach extracts meaningful compressing patterns with similar quality to the state-of-the-art multi-pass algorithms proposed for static databases of sequences. Moreover, our approach scales linearly with the size of data streams while all the existing algorithms do not.

[1]This chapter was published as Hoang Thanh Lam, Toon Calders, Jie Yang, Fabian Moerchen and Dmitriy Fradkin *Zips: Mining Compressing Sequential Patterns in Streams.* IDEA workshop at ACM KDD 2013

6.1 Introduction

Descriptive pattern mining aims at finding important structures hidden in data and providing a concise summary of important patterns or events in data. It answers questions posed by users during a data exploration process such as "what are the interesting patterns in the data?" and "what do these interesting patterns look like and how are they related?". The answers to these questions help people gain insights into the properties of the data, which in turn enables them to make business decisions. For example, given a stream of tweets on Twitter[2], people may be interested in what hot topics Twitter users are talking about at a given moment. Or given a stream of queries issued by users of a search engine, one may be interested in what kind of information people are looking for at a given time.

Mining frequent patterns is an important research topic in data mining. It has been shown that frequent pattern mining helps finding interesting association rules, or can be useful for classification and clustering tasks when the extracted patterns are used as features [15, 8]. However, in descriptive data mining the pattern frequency is not a reliable measure. In fact, it is often the case that highly frequent patterns are just a combination of very frequent yet independent items. For example, in [40, 34] we showed that the set of frequent sequential patterns extracted from a set of 787 abstracts of the Journal of Machine Learning Research articles contained many sequences that were multiple repetitions of the same frequent word such as "algorithm", "machine", "learn" etc. (see Figure 6-1 for an example). These uninteresting patterns do not provide any further insight into the important structures of the data given prior knowledge about individual item frequencies. Moreover, due to the anti-monotonicity of the frequency measure, if an itemset is frequent, all of its subsets are frequent as well which leads to the redundancy issue in the set of frequent patterns [61, 29, 75].

There are many approaches that address the aforementioned issues in the literature. One of the most successful approaches is based on data compression which looks for the set of patterns that compresses the data most. The main idea is based on the *Minimum Description Length Principle* (MDL) [30] stating that the best model describing data is the one that together with the description of the model, it compresses the data most. The MDL principle has been successfully applied to solve the redundancy issue in pattern mining and

[2] www.twitter.com

Pattern	Support	Pattern	Support
algorithm algorithm	0.376	method method	0.250
learn learn	0.362	algorithm result	0.247
learn algorithm	0.356	Data set	0.244
algorithm learn	0.288	learn learn learn	0.241
data data	0.284	learn problem	0.239
learn data	0.263	learn method	0.229
model model	0.260	algorithm data	0.229
problem problem	0.258	learn set	0.228
learn result	0.255	problem learn	0.227
problem algorithm	0.251	algorithm algorithm algorithm	0.222

Figure 6-1: The set of the most frequent closed patterns in the abstracts of the Journal of Machine Learning Research articles. The set is very redundant and contains a lot of uninteresting patterns.

to return meaningful patterns [75, 35].

So far most of the work focussed on mining compressing patterns from static datasets or from modestly-sized data. In practice, however databases are often very large. In some applications, data instances arrive continuously with high speed in a streaming fashion. In both cases, the algorithm must ideally scale linearly with the size of the data and be able to quickly handle fast data stream updates. In the streaming case, the main challenge is that whole data cannot be kept in memory and hence the algorithm has to be single-pass. None of the approaches described in the literature scales up to the arbitrarily large data or obey the single-pass constraint.

In this work, we study the problem of mining compressing patterns in a data stream where events arrive in batches for instance like stream of tweets. We first introduce a novel encoding scheme that encodes sequence data with the help of patterns. Different from the encodings being used in recent work [40, 34, 71], the new encoding is online which enables us to design online algorithms for efficiently mining compressing patterns from a data stream. We prove that there is a simple algorithm using the proposed online encoding scheme and achieving a near optimal compression ratio for data streams generated by an independent and identical distributed source, i.e. the same assumption that guarantees the optimality of the *Huffman* encoding in the offline case [20].

Subsequently, we formulate the problem of mining compressing patterns from a data stream. Generally, the data compression problem is NP-complete [68]. Under the streaming context with the additional single pass constraint, we propose a heuristic algorithm to

solve the problem. The proposed algorithm scales linearly with the size of data. In the experiments with three synthetic and three real-life large-scale datasets, the proposed algorithm was able to extract meaningful patterns from the streams while being much more scalable than the-state-of-the-art algorithms.

6.2 Related work

We classify the existing work into three main categories and discuss each of them in the following subsections.

Concise summary of the set of frequent patterns:

Mining closed patterns is one of the first approaches [61] addressing the redundancy issue in pattern mining. A frequent pattern is closed if there is no frequent super-pattern having the same frequency. The set of frequent closed patterns typically has much lower cardinality than the set of all frequent patterns.

Alternatively, another approach finding a concise representation of the set of frequent patterns is proposed by mining all non-derivable patterns [11]. It finds a concise set of patterns such that together with the support frequency information the entire set of frequent patterns can be derived.

The aforementioned methods are lossless approaches in which the set of all frequent patterns can be reconstructed from its concise representation. Alternative approaches were proposed to find a lossy concise representation of the set of frequent patterns by mining maximal frequent patterns [3]. Belonging to this category also include other lossy methods using clustering [79] or condense the pattern set in post-processing [63].

Approaches of this type were shown to be very effective in reducing the size of the pattern set. However, in the worst case, the pattern set can still be exponential in the number of items. Moreover, the set of frequent closed patterns may still contain many uninteresting patterns that are combinations of frequent items that are independent of each other [75].

Hypothesis testing based approaches

Hypothesis testing based approaches first assume that data are generated by a null model. The observed pattern frequency in the data is compared to the expected frequency of the

pattern given the assumption that the data are generated by the null model. If the observed frequency significantly deviates from the expectation of the pattern frequency in the null model, the pattern is considered statistically significant. This approach helps to remove uninteresting patterns that are explained by the null model alone.

Swap randomization [29] was one of the first hypothesis testing based approaches proposed for testing the significance of data mining results. Similar approaches are also proposed for sequence data in which Markov models are usually chosen as a null model [31, 13]. In the field of significant subgraph mining, the idea of using hypothesis testing to filter out insignificant frequent subgraphs were proposed in which the null model is similar to swap randomization for generating random graphs preserving the degree distribution [50].

Depending on the expectation of data miners about the type of patterns, different null models are chosen. In the case when there is no particular preference about a specific null model, the maximum entropy model with constraints on pattern frequency can be used as a null model [69, 5, 6]. Beside assuming a fixed null model, in [45], the authors proposed an approach iteratively updating the null model and succinctly building the set of patterns to summarize the pattern sets effectively.

The hypothesis testing based approaches were shown to be very effective in filtering out uninteresting patterns. Especially, the iterative mining approach was also able to solve the redundancy issue. These approaches on the one hand provide us with a flexibility through explicit choice of the null models to only retain the patterns that we consider as unexpected. However, on the other hand, the hypothesis testing based approaches have a disadvantage that the significance of patterns is a subjective score with respect to a given null model. Therefore, the mining results are highly dependent on the choice of the null model. In many cases, choosing the right null model is not a trivial task.

Minimum description length approaches

In the MDL-based approaches, an encoding scheme is defined to compress data by a set of patterns. The choice of encodings implicitly defines a probability distribution on the data. This is in contrast to hypothesis testing based approaches in which the null models are chosen explicitly. According to the MDL principle [30] the set of most compacted patterns that compresses the data most is considered as the best set of patterns. This approach was shown to be very effective in solving the redundancy issue in pattern mining [75].

The SubDue system [35] is the first work exploiting the MDL principle for mining a non-redundant set of frequent subgraphs. In the field of frequent itemset mining, the well-known Krimp algorithm [75] was shown to be very good at solving the redundancy issue and at finding meaningful patterns.

The MDL principle was first applied for mining compressing patterns in sequence data in [40, 34] and in [71]. The GoKrimp algorithm in the former work solved the redundancy issue effectively. However, the first version of the GoKrimp algorithm [40] used an encoding scheme that does not punish large gaps between events in a pattern. In an extended version of the GoKrimp algorithm [34] this issue is solved by introducing gaps into the encoding scheme based on Elias codes. Besides, a dependency test technique is proposed to filter out meaningless patterns. Meanwhile, in the latter work the SQS algorithm proposed a clever encoding scheme punishing large gaps. In doing so, the SQS was able to solve the redundancy issue effectively. At the same time it was able to return meaningful patterns based solely on the MDL principle.

However, a disadvantage of the encoding defined by the SQS algorithm is that it does not allow encoding of overlapping patterns. Situations where patterns in sequences overlap are common in practice, e.g. message logs produced by different independent components of a machine, network logs through a router etc. Moreover, neither the GoKrimp algorithm nor the SQS algorithm were intended for mining compressing patterns in data streams. The encodings proposed for these algorithms are *offline encodings*. Under the streaming context, an offline encoding does not work because of the following reasons:

1. Complete usage information is not available at the moment of encoding because we don't know the incoming part of the stream

2. When the data size becomes large, the dictionary size usually grows indefinitely beyond the memory limit. Temporally, part of the dictionary must be evicted. In the latter steps, when an evicted word enters the dictionary again we loose the historical usage of the word completely.

3. Handling updates for the offline encoding is expensive. In fact, whenever the usage of the word is updated, all the words in the dictionary must be updated accordingly. On one hand, this operation is expensive, on the other hand, it is impossible to update the compression size correctly for the case that part of the dictionary has been evicted.

In contrast to these approaches, the Zips algorithm proposed in this work inherits the advantages of both state-of-the-art algorithms. It defines a new *online encoding* scheme that allows to encode overlapping patterns. It does not need any dependency test to filter out meaningless patterns and more importantly, under reasonable assumptions, it provably scales linearly with the size of the stream making it the first work in this topic being able to work efficiently on very large datasets.

Our work is tightly related to the *Lempel-Ziv*'s data compression algorithm [20]. However, since our goal is to mine interesting patterns instead of compression, the main differences between our algorithm and data compression algorithms are:

1. Data compression algorithms do not aim to a set of patterns because they only focus on data compression.

2. Encodings of data compression algorithms do not consider important patterns with gaps. The *Lempel-Ziv* compression algorithms only exploit repeated strings (consecutive subsequences) to compress the data while in descriptive data mining we are mostly interested in patterns interleaved with noise events and other patterns.

6.3 Data stream encoding

In this work, we assume that events in a data stream arrive in batches. This assumption covers broad types of data streams such as tweets, web-access sequences, search engine query logs, etc. This section discusses online encodings that compress a data stream by a set of patterns. For education reasons, we start with the simplest case where only singletons are used to encode the data. The generalized encoding for the case with non-singletons are described in the subsequent subsections.

Online encoding using singletons:

We discuss an online encoding that uses only singletons to compress the data. Since this encoding does not exploit any pattern for compress the data, we consider the representation of the data in this encoding as an uncompressed form of that data. Let $\sum = \{a_1, a_2, \cdots, a_n\}$ be an alphabet containing a set of characters a_i, the online data encoding problem can be formulated as follows:

Definition 9 (Online Data Encoding Problem)**.** *Let A denote a sender and let B denote a receiver. A and B communicate over some network, where sending information is expensive. A observes a data stream $S_t = b_1 b_2 \cdots b_t$. Upon observing a character b_t, A needs to compress the character and transmit it to the receiver B, who must be able to uncompress it. Since sending information on the network is expensive, the goal of A and B is to compress the stream as much as possible to save up the network bandwidth.*

In the offline scenario, i.e. when S_t is finite and given in advance, one possible solution is to first calculate the frequency of every item a (denoted as $f(a)$) of the alphabet in the sequence S_t. Then assign each item a a codeword with length proportional to its entropy, i.e. $-\log f(a)$. It has been shown that when the stream is independent and identically distributed (i.i.d) this encoding, known as the *Huffman* code in the literature, is optimal [20]. However, in the streaming scenario, the frequency of every item a is unknown and the codeword of a must be assigned at the time a arrives and B must know that codeword to decode the compressed item.

We propose a simple solution for Problem 9 as follows. First, in our proposed encoding we need codewords for natural numbers. This work uses the *Elias Delta* code [78] denoted $E(n)$ to encode the number n. The Elias was chosen because it was provable as near optimal when the upper bound on n is unknown in advance. The length of the codeword $E(n)$ is $\lfloor \log_2 n \rfloor + 2\lfloor \log_2 (\lfloor \log_2 n \rfloor + 1) \rfloor + 1$ bits.

A first notifies B of the size of the alphabet by sending $E(|\sum|)$ to B. Then it sends B the dictionary containing all characters of the alphabet $\sum$ in the lexicographical order. Every character in the dictionary is encoded by a binary string with length $\lceil \log_2 |\sum| \rceil$. Finally, when a new character in the stream arrives A sends the codeword of the gap between the current and the most recent occurrence of the character. When B receives the codeword of the gap it decodes the gap and uses that information to refer to the most recent occurrence of the character which has been already decoded in the previous step. Since the given encoding uses a reference to the most recent occurrence of a word to encode its current occurrence we call this encoding the *reference encoding* scheme. We call the sequence of encoded gaps sent by A to B the *reference stream.*

Example 19. *Figure 6-2 shows an example of a reference encoding scheme. A first sends B the alphabet in lexicographical order. When each item of the stream arrives A sends B*

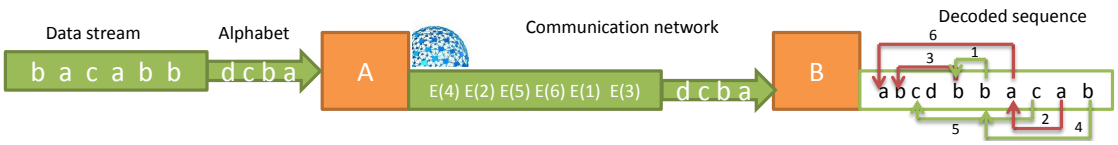


Figure 6-2: A first sends B the alphabet *abcd* then it sends the codewords of gaps between consecutive occurrences of a character. B decodes the gaps and uses them to refer to the characters in part of the stream having been already decoded. The reference stream is $E(3)E(1)E(6)E(5)E(2)E(4)$.

the codeword of the gap to its most recent occurrence. For instance, A sends $E(3)$ *to encode the first occurrence of b and sends* $E(1)$ *to encode the next occurrence of b. The complete reference stream that A sends B is* $E(3)E(1)E(6)E(5)E(2)E(4)$.

Let O be a *reference encoding*; denote $L^O(S_t)$ as the length of the data including the length of the alphabet. The average number of bits per character is calculated as $\frac{L^O(S_t)}{t}$. The following theorem shows that when the data stream is generated by an *i.i.d* source, i.e. the same assumption guaranteeing the optimality of the *Huffman* code, the *reference encoding* scheme approximates the optimal solution by a constant factor with probability 1.

Theorem 9 (Near Optimality)**.** *Given an i.i.d data stream* S_t*, let* $H(P)$ *denote the entropy of the distribution of the characters in the stream. If the Elias Delta code is used to encode natural numbers then:*

$$Pr\left(\lim_{t\mapsto\infty}\frac{L^O(S_t)}{t} \leq H(P) + \log_2(H(P)+1) + 1\right) = 1$$

Proof. For every character $a_i \in \sum$, let $f_i(t)$ be the frequency of a_i in the stream S_t at time point t. Denote $C_i(t)$ as the total cost (in the number of bits) for encoding the occurrences of a_i. Therefore, the description length of the stream can be represented as:

$$L^O(S_t) = \sum_{i=1}^{n} C_i(t)$$

$$\Rightarrow \frac{L^O(S_t)}{t} = \frac{\sum_{i=1}^{n} C_i(t)}{t}$$

$$\Rightarrow \frac{L^O(S_t)}{t} = \sum_{i=1}^{n} \frac{C_i(t)}{f_i(t)} * \frac{f_i(t)}{t}$$

Given a character a_i denote $p_i > 0$ as the probability that a_i occurs in the stream. Denote G_i as the gap between two consecutive occurrences of a_i in the stream. Since the stream is i.i.d, G_i is distributed according to the *geometric distribution* with parameter p_i, i.e $Pr(G_i = k) = (1-p_i)^{k-1}p_i$. Recall that the expectation of G_i is $E[G_i] = \frac{1}{p_i}$.

According to the law of large numbers $Pr\left(\lim_{t\mapsto\infty} \frac{f_i(t)}{t} = p_i\right) = 1$. Moreover, recall that $C_i(t)$ is the sum of the gaps' codewords lengths and the initial cost for encoding the character a_i in the alphabet. When $t \mapsto \infty$, the frequency of a_i also goes to infinity because $p_i > 0$, therefore, by *the law of large number*: $Pr\left(\lim_{t\mapsto\infty} \frac{C_i(t)}{f_i(t)} = E[C(G_i)]\right) = 1$.

When the Elias Delta code is used to encode the gap, we have:

$$C(G_i) = \lfloor \log_2 G_i \rfloor + 2\lfloor \log_2 (\lfloor \log_2 G_i \rfloor + 1) \rfloor + 1$$

$$\Rightarrow C(G_i) \leq \log_2 G_i + 2\log_2 (\log_2 G_i + 1) + 1$$

$$\Rightarrow E[C(G_i)] \leq E[\log_2 G_i] + 2E[\log_2 (\log_2 G_i + 1)] + 1$$

Since log is a concave function, by the *Jenssen's inequality* :

$$E[C(G_i)] \leq \log_2 E[G_i] + 2\log_2 (\log_2 E[G_i] + 1) + 1$$

We further imply that:

$$Pr\left(\lim_{t\mapsto\infty} \frac{L^O(S_t)}{t} \leq \sum_i p_i \log_2 E[G_i] + 2p_i \log_2 (\log_2 E[G_i] + 1) + p_i\right) = 1$$

$$\Rightarrow Pr\left(\lim_{t\mapsto\infty} \frac{L^O(S_t)}{t} \leq \sum_{i=1}^{n} p_i \log_2 E[G_i] + 2\log_2 (\sum_{i=1}^{n} p_i \log_2 E[G_i] + 1) + 1\right) = 1$$

$$\Rightarrow Pr\left(\lim_{t\mapsto\infty} \frac{L^O(S_t)}{t} \leq H(P) + \log_2(H(P) + 1) + 1\right) = 1$$

from which the lemma is proved.□ □

It has been shown that in expectation the lower bound of the average number of bits per character of any encoding scheme is $H(P)$ [20]. Therefore, a corollary of Theorem 9 is

that the *reference encoding* approximates the optimal solution by a constant factor $\alpha = 2$ plus one extra bit.

In the proof of Theorem 9 we can also notice that the gaps between two consecutive occurrences of a character represent the usage of the character in the offline encoding because in expectation the gap is proportional to the entropy of the character, i.e. $-\log p_i$. This property is very important because it provides us with a lot of conveniences in designing an effective algorithm to find compressing patterns in a data stream. In particular, since gaps can be calculated instantly without the knowledge about the whole data stream, using reference encoding we can solve all the aforementioned issues of the offline encodings discussed earlier in this section:

1. Using the reference encoding we don't need complete information about the stream to encode the observed part of the stream. The near optimality of the reference encoding in the i.i.d case is ensured by Theorem 9.

2. Under the presence of word evictions (to be discussed in the algorithm part), when an evicted word enters the dictionary again, we just need to follow that word for a while to get the recent gaps information with no need to care about the historical usage of the word before the eviction.

3. Updates can be handled more efficiently even under the presence of eviction because part of the stream that has been compressed remains unchanged.

Online encoding with non-singletons

The *reference encoding* can be easily extended for the case that uses singleton together with non-singleton patterns to encode a data stream. Let $\mathfrak{S} = S_1 S_2 \cdots S_t$ denote a stream of sequences where each S_i is a sequence of events. Let D be a dictionary containing all characters of the alphabet and some non-singletons. *Reference encodings* use D to compress a data stream $\mathfrak{S}$ by replacing instances of words in the dictionary by references to the most recent occurrences of the words. If the words are non-singletons, beside the references, gaps between characters of the encoded words must be stored together with the references. Therefore, in a reference encoding, beside the reference stream we also have a *gap stream*.

Similar to the case with non-singleton, first we need to encode the dictionary D (now contains both singleton and non-singleton). We add a special symbol $\sharp$ to the alphabet.

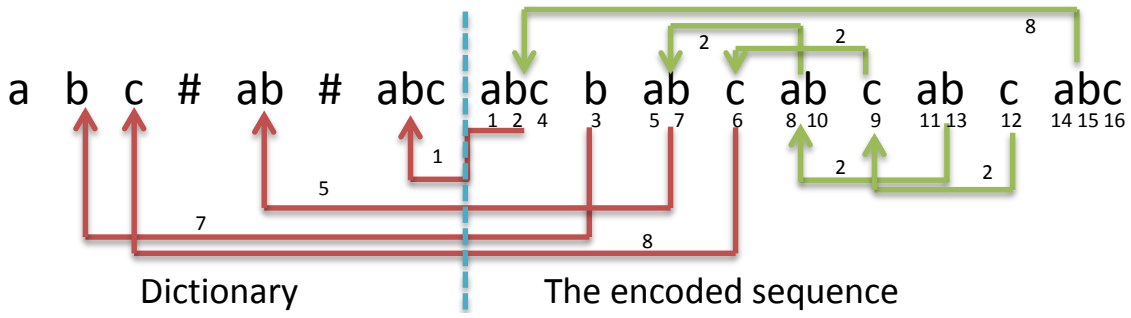


Figure 6-3: An example of encoding the sequence $S = abbcacbacbacbabc$ with a reference encoding. The arrows represent the references between two consecutive encoded occurrences of a word which represent as a reference stream $E(1)E(7)E(5)E(8)E(2)E(2)E(2)E(2)E(8)$. Having the reference stream we can reconstruct the stream but not in the same order of event occurrence. Therefore, we need a gap stream indicating the gaps between consecutive characters in each encoded non-singletons. For example, the gap stream is $E(1)E(2)E(2)E(2)E(2)E(1)E(1)$.

The binary representation of the dictionary starts with the codeword of the size of the dictionary. It is followed by the codewords of all the characters in the alphabet each with length $\lceil \log_2 |D| \rceil$. The representations of every non-singleton follow right after that. The binary representation of a non-singleton contains codewords of the characters of the non-singleton. Non-singletons are separated from each other by the special character $\sharp$.

Example 20 (Dictionary representation)**.** *The dictionary $D = \{a, b, c, \sharp, ab, abc\}$ can be represented as follows $E(6)C(a)C(b)C(c)C(\sharp)C(a)C(b)C(\sharp)C(a)C(b)C(c)$. The representation starts with $E(6)$ indicating the size of D. It follows by the codewords of all the characters and the binary representation of the non-singletons separated by $\sharp$.*

Having the binary representation of the dictionary, A first sends that representation to B. After that A send the encoding of the actual stream with the reference stream and the gap stream. The following example show how to encode a sequence with a reference encoding.

Example 21 (Reference encoding)**.** *Given the dictionary $D = \{a, b, c, \sharp, ab, abc\}$ and a sequence $S = abbcacbacbacbabc$. S can be encoded by a reference encoding using the dictionary D as shown in Figure 6-3 (the numbers below the character stream denote the positions of the character in the original stream). The reference stream is $E(1)E(7)E(5)E(8)E(2)E(2)E(2)E(2)E(8)$, where $E(1)$ is the reference of the first abc to the position of abc in the dictionary, $E(7)$ is the reference of the following b to the position of b in the dictionary and so on.*

The gap stream is $E(1)E(2)E(2)E(2)E(2)E(1)E(1)$, where for instance, the first codewords $E(1)E(2)$ indicate the gaps between a and b, b and c in the first occurrence of abc at position $(1, 2, 4)$ in the stream. For non-singleton, there is no gap information representing in the gap stream.

Example 22 (Decoding)**.** *In Figure 6-3, the sequence can be decoded as follows. Reading the first codeword of the reference stream, i.e. $E(1)$, the decoder refers one step back to get abc. There will be two gaps (between a, b and b, c) so the decoder reads the next two codewords from the gap stream, i.e. $E(1)$, and $E(2)$. Knowing the gaps it can infer the positions of abc. In this case, the positions are 1,2 and 4. Subsequently, the decoder reads the next codeword from the reference stream, i.e. $E(7)$, it refers seven steps back and decode the current reference as b. There is no gap because the word is a singleton, the position of b in the stream corresponds to the earliest position that has not been occupied by any decoded character, i.e. 3. The decoder continues decode the other references of the stream in the same way.*

Different from the singleton case, there might be a lot of different reference encodings for a stream given a dictionary. Each reference encoding incurs different description lengths. Finding an optimal dictionary and an optimal reference encoding is the main problem we solve in this work.

6.4 Problem definition

Given a data stream $\mathfrak{S}$ and a dictionary D denote $L_D^C(\mathfrak{S})$ as the description length of the data (including the cost to store the dictionary) in the encoding C. The problem of mining compressing sequential patterns in data stream can be formulated as follows:

Definition 10 (Compressing patterns mining)**.** *Given a stream of sequences $\mathfrak{S}$, find a dictionary D and an encoding C such that $L_D^C(\mathfrak{S})$ is minimized.*

Generally, the problem of finding the optimal lossless compressed form of a sequence is NP-complete [68]. In this work, Problem 10 is similar to the data compression problem but with additional constraint on the number of passes through data. Therefore, in next section we discuss a heuristic algorithm inspired by the idea of the *Lempel-Ziv*'s data compression algorithm [20] to solve this problem.

Algorithm 12 Zips(S)

Input: Event stream $\mathfrak{S} = S_1 S_2 \cdots$
Output: Dictionary D
$D \longleftarrow \emptyset$
for $t = 1$ **to** ∞ **do**
  while $S_t \neq \epsilon$ **do**
    $w = encode(S_t)$
    $w^* = extend(w)$
    $update(w^*)$
  end while
end for
Return D

6.5 Algorithms

In this section, we discuss an algorithm for finding a good set of compressing patterns from a data stream. Our algorithm has the following essential properties for a streaming application:

1. Single pass: only one pass through the data.

2. Memory-efficient: since the data stream grows indefinitely, the streaming algorithms create a memory efficient summary of the stream.

3. Fast and scalable: summary update operation is fast to catch up with high-speed stream and scales up to the size of the stream.

We call our algorithm Zips as for *Zip a stream*. There are three important subproblems that Zips will solve. The first problem concerns how to grow a dictionary of promising candidate patterns for encoding the stream. Since the memory is limited, the second problem is how to keep a small set of important candidates and evict from the dictionary unpromising candidates. The last problem is that having a dictionary how to encode the next sequence effectively with existing words in the dictionary.

The pseudo-code depicted in Algorithm 12 shows how Zips work. It has three subroutines each of them solves one of the three subproblems:

1. Compress a sequence given a dictionary: for every new sequence S_t in the stream, Zips first uses the subroutine $encode(S_t)$ to find the word w in the dictionary which gives the most compression benefit when it is used to encode the uncompressed part

of the sequence. Subsequently, the instance corresponds to the best chosen word is removed from S_t. Detail about how to choose w is discussed in subsection 6.5.

2. Grow a dictionary: Zips uses the subroutine $extend(w)$ to extend the word w returned by the subroutine $encode(S_t)$. Word extensions are discussed in subsection 6.5.

3. Dictionary update: the new extension is added to the dictionary. When the dictionary size exceeds the memory limit, a space-saving algorithm is used to evict unpromising words from the dictionary (subsection 6.5).

These steps are iteratively repeated as long as S_t is not encoded completely. When compression of the sequence finishes, Zips continues to compress the next in a similar way.

Compress a sequence:

Let S be a sequence, consider a dictionary word $w = a_1 a_2 \cdots a_k$, let $S(w)$ denote an instance of w in S. Let $g_2, g_3, \cdots, g_k$ be the gaps between the consecutive characters in $S(w)$. We denote the gap between the current occurrence and the most recent encoded occurrence of w by g. Let $\overline{g_i}$ $i = 1, \cdots, k$ be the gap between the current and the most recent occurrence of a_i. Therefore, to calculate the compression benefit we subtract the size of encoding $S(w)$ and the cost of encoding the gap to the last encoded occurrence of $S(w)$ from the size of encoding each singleton:

$$B(S(w)) = \sum_{i=1}^{k} |E(\overline{g_i})| - |E(g)| - \sum_{i=2}^{k} |E(g_i)| \tag{6.1}$$

Example 23 (Compression benefit)**.** *Figure 6-4.a shows a sequence S in the uncompressed form and Figure 6-4.b shows the current form of S. Assume that the instance of $w = abc$ at positions $1, 2, 4$ is already compressed. Consider two instances of abc in the uncompressed part of S:*

1. *$S^1(w) = (a, 3)(b, 5)(c, 7)$: the cost to replace this instance by a pointer is equal to the sum of the cost to encode the reference to the previous encoded instance of abc $|E(1)|$ plus the cost of gaps $|E(2)| + |E(2)|$. The cost of representing this instance in an uncompressed form is $|E(2)| + |E(3)| + |E(3)|$. Therefore the compression benefit of*

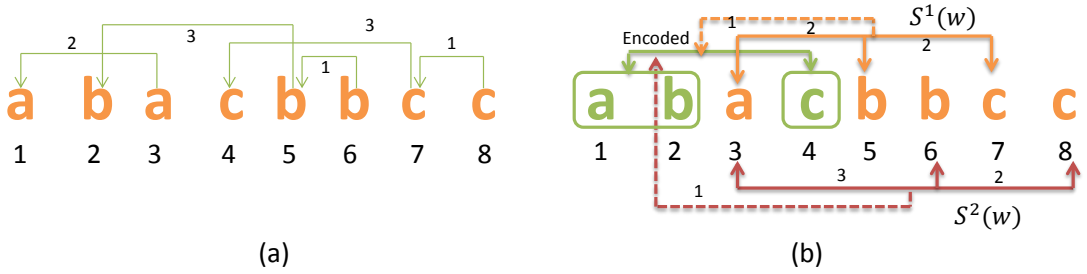


Figure 6-4: An illustration of how compression benefit is calculated: (a) The sequence S in the uncompressed form. (b) two instances of $w = abc$: $S^1(w)$ and $S^2(w)$ and their references to the most recent encoded instance of w highlighted by the green color.

using this instance to encode the sequence is $B(S^1(w)) = |E(2)| + |E(3)| + |E(3)| - |E(1)| - |E(2)| - |E(2)| = 3$ *bits.*

2. $S^2(w) = (a, 3)(b, 6)(c, 8)$*: the compression benefit of using* $S^2(w)$ *to encode the sequence is calculated in a similar way:* $B(S^2(w)) = |E(2)| + |E(1)| + |E(1)| - |E(1)| - |E(3)| - |E(2)| = -3$ *bits.*

In order to ensure that every symbol of the sequence is encoded, the next instance considered for encoding has to start at the first non-encoded symbol of the sequence. There maybe many instances of w in S that start with the first non-encoded symbol of S, denote $S^*(w) = argmax_{S(w)} B(S(w))$ as the one that results in the maximum compression benefit. We call $S^*(w)$ the *best match* of w in S. Given a dictionary, the encoding function depicted in Algorithm 13 first goes through the dictionary and finds the best match starting at the next uncompressed character of every dictionary word in the sequence S (line 4). Among all the best matches, it greedily chooses the one that results in the maximum compression benefit (line 6).

For any given dictionary word $w = a_1 a_2 \cdots a_k$, the most important subroutine of Algo-

Algorithm 13 encode(S)

1: **Input**: a sequence S and dictionary $D = w_1 w_2 \cdots w_N$
2: **Output**: the word w that starts at the first non-encoded symbol gives the most additional compression benefit
3: **for** $i = 1$ **to** N **do**
4: $S^*(w_i) = bestmatch(S, w_i)$
5: **end for**
6: $max = argmax_i B(S^*(w_i))$
7: Return w_{max}

rithm 13 is to find the best match $S^*(w)$. This problem can be solved by creating a directed acyclic graph $G(V, E)$ as follows:

1. Initially, V contains a start node s and an end node e
2. For every occurrence of a_i at position p in S, add a vertex (a_i, p) to V
3. Connect s with the node (a_1, p) by a directed edge and add to that edge a weight value equal to $|E(\overline{g}_1)| - |E(g)|$.
4. Connect every vertex (a_k, p) with e by a directed edge with weight 0.
5. For all $q > p$ connect (a_i, p) to (a_{i+1}, q) by a directed edge with weight value $|E(\overline{g}_{i+1})| - |E(q-p)|$

Theorem 10 (The best match and the maximum path)**.** *The best match $S^*(w)$ corresponds to the directed path from s to e with the maximum sum of the weight values along the path.*

The proof of theorem 10 is trivial since any instance of w in S corresponds to a directed path in the directed acyclic graph and vice versa. The sum of the weights along a directed path is equal to the benefit of using the corresponding instance of w to encode the sequence. Finding the directed path with maximum weight sum in a directed graph is a well-known problem in graph theory. That problem can be solved by a simple dynamic programming algorithm in linear time of the size of the graph, i.e. $O(|S|^2)$[19].

Example 24 (Find the best match in a graph)**.** *Figure 6-5 shows the directed acyclic graph created from the instances of abc in the uncompressed part of S shown in Figure 6-4.b. The best match of abc corresponding to the path $s(a,3)(b,5)(c,7)e$ with the maximum sum of weights equal to 3 bits.*

It is important to notice that in order to evaluate the compression benefit of a dictionary word, Equation 6.1 only requires bookkeeping the position of the most recent encoded instance of the word. This is in contrast to the offline encodings used in recent work [40, 34, 71] in which the bookkeeping of the word usage and the gaps cost is a must. When a new instance of a word is replaced by a pointer, the relative usage and the codewords of all dictionary words also change. As a result, the compression benefit needs to be recalculated

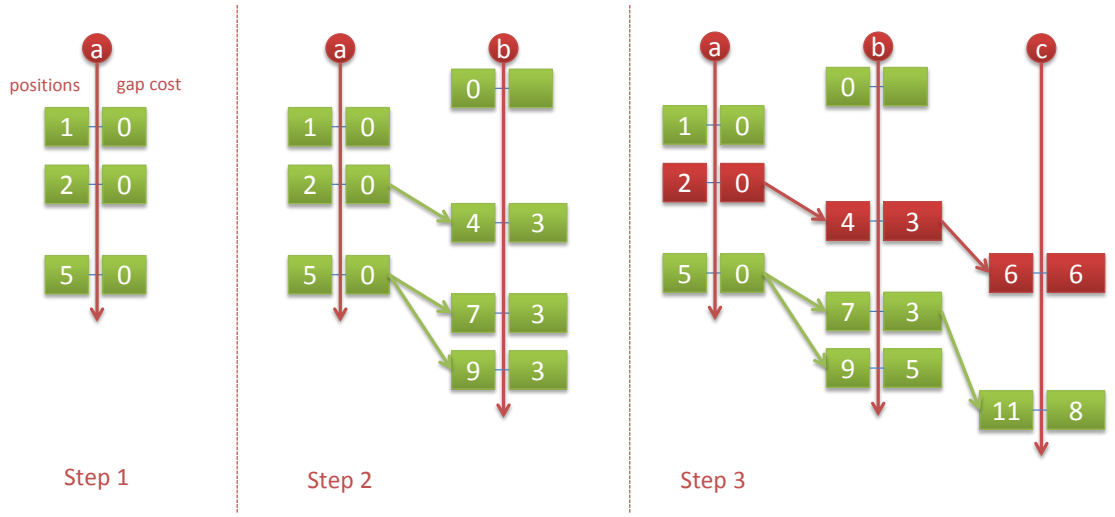


Figure 6-5: A directed acyclic graph created from the instances of *abc* in the uncompressed part of S shown in Figure 6-4.b

by a pass through the dictionary. This operation is an expensive task when the dictionary size is unbounded.

Dictionary extension:

Initially, the dictionary contains all singletons; it is iteratively expanded with the locally best words. In each step, when the best match of a dictionary word has been found, Zips extends the best match with one extra character and adds this extension to the dictionary. There are different options to choose the character for extension. In this work, Zips chooses the next uncompressed character right after the word. The choice is inspired by the same extension method suggested by the *Lempel-Ziv* compression algorithms.

Moreover, there is another reason behind our choice. When w is encoded for the first time, the reference to the previous encoded instance of w is undefined although the word has been already added to the dictionary. Under that circumstance, we have to differentiate between two cases: either a reference to an extension or to an encoded word. To achieve this goal one extra flag bit is added to every reference. When the flag bit is equal to 1, the reference refers to an extension of an encoded word. Otherwise, it refers to an encoded word. When the former case happens by extending the word with the character right after it, the decoder always knows where to find the last character of the extension. When a word is added to a dictionary all of its prefixes have been already added to the dictionary.

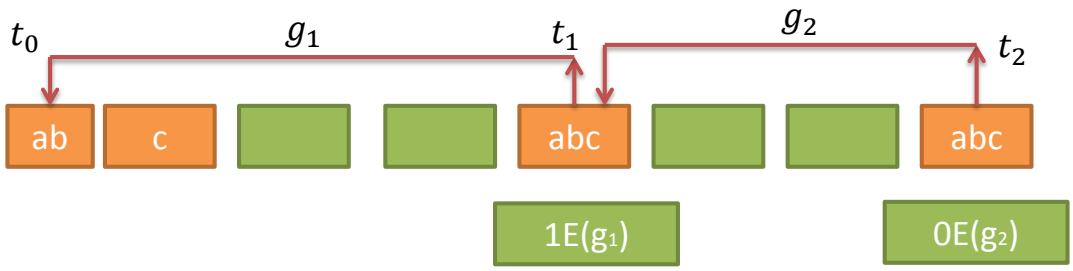


Figure 6-6: An example of word is extended and encoded the first time and the second time. One extra flag bit is added to every reference to differentiate two cases.

This property enables us to store the dictionary by using a prefix tree.

Example 25. *Figure 6-6 shows the first moment t_0 when the word $w = abc$ is added to the dictionary and two other moments t_1 and t_2 when it is used to encode the stream. At t_1, the flag bit 1 is used to notify the decoder that the gap g_1 is a reference to an extension of an encoded word, while at t_2 the decoder understands that the gap g_2 is a reference to the previous encoded instance of w.*

Due to dictionary extensions, ambiguous reference may happen. For instance, a potential case of ambiguous reference called *reference loop* is discussed in the following example:

Example 26 (reference loops)**.** *At time point t_0 the word $w = ab$ at positions p_1p_2 is extended by c at position p_3. Later on at another time point $t_1 > t_0$, another instance of abc at $q_1q_2q_3$ ($q_1 > p_1$) refers back to the instance abc at $p_1p_2p_3$. At time point $t_2 > t_1$, another instance of abc at $r_1r_2p_3$ ($r_1 > q_1$) refers back to the instance abc at $q_1q_2q_3$. In this case, c at p_3 and c at q_3 refers to each other forming a reference loop.*

Reference loops result in ambiguity when we decode the sequence. Therefore, when we look for the next best matches, in order to avoid reference loops, we always check if the new match incurs a loop. The match that incurs a loop is not considered for encoding the sequence. Checks for ambiguity can be done efficiently by creating paths of references. Vertices of a reference path are events in the sequence. Edge between two consecutive vertices of the path corresponds to a reference between the associated events. Since the total sizes of all the paths is at most the sequence length, its is cheap to store the paths for a bounded size sequence. Moreover, updating and checking if a path is a loop can be done in $O(1)$ if vertices of the path are stored in a hashmap.

Example 27 (reference paths)**.** *The references paths of the encoding in Example 26 are:* $(a, r_1) \mapsto (a, q_1) \mapsto (a, p_1)$*,* $(b, r_2) \mapsto (b, q_2) \mapsto (b, p_2)$ *and* $(c, p_3) \mapsto (c, q_3) \mapsto (c, p_3)$*. The last path is a loop.*

Dictionary update:

The new extension is added to the dictionary. When the dictionary exceeds the memory limit, the *space-saving* algorithm is used to evict unpromising words from the dictionary. The space-saving algorithm [48] is a well-known method proposed for finding the most frequent items in a stream of items given a budget on the maximum number of counters it can keep in the stream summary. In this work, we propose a similar space-saving algorithm to keep the number of non-singleton words in the dictionary at below a predefined number M while it can be able to return the set of compressing patterns with high accuracy.

The algorithm works as follows, for every non-singleton word w it maintains a counter with two fields. The first field denoted as $w[1]$ contains an over-estimate of the compression benefit of w. The second field denoted as $w[2]$ contains the compression benefit of the word with least compression benefit in the dictionary at the moment that w is inserted into the dictionary.

Every time when a word w is chosen by Algorithm 13 to encode its best match in the sequence S_t, the compression benefit of the word is updated. The word w is then extended to w^* with an extra character by the extension subroutine. In its turn, Algorithm 14 checks if the dictionary already contains w^*. If the dictionary does not contains w^* and it is full with M non-singleton words, the least compressing word v resident at a leaf of the dictionary prefix-tree is removed from the tree. Subsequently, the word w^* is inserted into the tree and its compression benefit can be over-estimated as $w[1] = w[2] = v[1]$. The first counter of every word is always greater than the true compression benefit of the words. This property ensures that the new emerging word is not removed very quickly because its accumulated compression benefit is dominated by long lasting words in the dictionary. For any word w, the difference between $w[2]$ and $w[1]$ is the actual compression benefit of w since the moment that w is inserted into the dictionary. At anytime point when we need to find the most compressing patterns, we compare the value of $w[2] - w[1]$ and select those with highest $w[2] - w[1]$. In section 7.6 we show empirical results with different datasets that this algorithm is very effective in finding the most compressing patterns with high accuracy

Algorithm 14 update(w^*)

Input: a word w^* and dictionary $D = \{w_1, w_2, \cdots, w_N\}$
Output: the dictionary D
$m \leftarrow |i : w_i$ is a non-singleton$|$
$v = argmin_i w_i[1]$ and v is non-singleton at a leaf of the prefix-tree
if $m > M$ and $w^* \notin D$ **then**
    $D = D \setminus \{v\}$
    $w^*[1] = w^*[2] = v[1]$
    $D = D \bigcup \{w^*\}$
else if $w^* \notin D$ **then**
    $w^*[1] = w^*[2] = 0$
    $D = D \bigcup \{w^*\}$
else
    add additional compression benefit to $w^*[1]$
end if
Return D

even with limited memory.

Algorithm analysis

Memory consumption: the algorithm needs to store the whole dictionary. The size of the dictionary is proportional to $O(|\sum| + M)$ where M is the maximum number of non-singletons in the dictionary. When the size of the alphabet is bounded and M is chosen as a constant, the memory consumption of Zips is constant too.

Computation complexity: the complexity of the dynamic programming algorithm for calculating the best match in a sequence S is $O(|S|^2)$. The maximum number of iterations to encode a sequence is $O(|S|)$. Therefore, the encoding function takes $O(M|S|^3)$ in the worst case. If M and $|S|$ are upper-bounded by a constant, the cost of encoding takes O(1). The extension also takes $O(1)$ with the assumption that M and $|S|$ are upper-bounded by a constant. Therefore, the complexity of the Zips algorithm is linear in the size of the stream. This fact is empirically verified in the section 7.6.

6.6 Experiments

We perform experiments with three synthetic datasets with ground truth and three large-scale real-world datasets. Our implementation of the Zips algorithm in C++ together

Datasets	# Sequences	# Events	Alphabet size	Ground-truth
Parallel	10000	1000000	25	Yes
Noise	10000	1000000	1025	Yes
Plant	1000	100000	1050	Yes
JMLR	787	75646	3846	No
Tweets	900417	8008552	452264	No
AOL	10122004	21080479	616145	No

Figure 6-7: Datasets.

with the datasets are available for download at our project website[3]. All the experiments were carried out on a machine with 16 processor cores, 2 Ghz, 12 GB memory, 194 GB local disk, Fedora 14 / 64-bit. As baseline algorithms, we choose the GoKrimp algorithm [40, 34] and the SQS algorithm [71] for comparison in terms of running time, scalability, and interpretability of the set of patterns.

Data

We use six different datasets to evaluate the performance of the Zips algorithm. A summary of five datasets is presented in Figure 6-7. Details about the creation of these datasets from raw data are as follows:

1. **Parallel** [40, 34]: is a synthetic dataset which mimics a typical situation in practice where the data stream is generated by five independent parallel processes. Each process P_i $(i = 1, 2, \cdots, 5)$ generates one event from the set of events $\{A_i, B_i, C_i, D_i, E_i\}$ in that order. In each step, the generator chooses one of five processes uniformly at random and generates an event by using that process until the stream length is 1000000. For this dataset, we know the ground truth since all the sequences containing a mixture of events from different parallel processes are not the right patterns.

2. **Noise** [40, 34]: is a synthetic dataset generated in the same way as the generation of the parallel dataset but with additional noise. A noise source generates independent events from a noise alphabet with 1000 distinct noise events and randomly mixes noise events with parallel data. The amount of noise is 20% of the data. All the subsequences containing mixtures of the events from different sources are considered wrong patterns.

[3] `www.win.tue.nl/~lamthuy/zips.html`

3. **Plant**: is a synthetic dataset generated in the same way as the generation of the plant10 and plant50 dataset used in [71]. The plant10 and plant50 are small so we generate a larger one with ten patterns each with 5 events occurs 100 times at random positions in a sequence with length 100000 generated by 1000 independent noise event types.

4. **JMLR**: contains 787 abstracts of articles in the Journal of Machine Learning Research. English words are stemmed and stop words are removed. JMLR is small but it is considered as a benchmark dataset in the recent work [40, 34, 71]. The dataset is chosen also because the set of extracted patterns can be easily interpreted.

5. **Tweets**: contains over 1270000 tweets from 1250 different twitter accounts[4]. All tweets are ordered ascending by timestamp, English words are stemmed and stop words are removed. After preprocessing, the dataset contains over 900000 tweets. Similar to the JMLR dataset, this dataset is chosen because the set of extracted patterns can be easily interpreted.

6. **AOL**: contains over 25 million queries given by users of the AOL search engine[5]. All queries are ordered ascending by timestamp, English words are stemmed and stop words are removed. Duplicate queries by the same users in a session are removed. The final dataset after preprocessing contains more than 10 million queries.

Running time and Scalability

Figure 6-8 plots the running time and the average update time per sequence of the Zips algorithm in two dataset Tweets and AOL when the data stream size (the number of sequences) increases. Three different lines in each subplot correspond to different maximum dictionary size settings $M = 1000$, $M = 5000$ and $M = 10000$ respectively. The results show that the Zips algorithm scales linearly with the size of the stream. The average update time per sequence is constant given a maximum dictionary size setting. For example, when $M = 10000$, Zips can handle one update in about 20-100 milliseconds.

Figure 6-9 shows the running time in y-axis of the Zips algorithm against the stream size in x-axis in two datasets Tweets and AOL when the maximum dictionary size is set to 1000.

[4] `http://user.informatik.uni-goettingen.de/~txu/cuckoo/dataset.html`
[5] http://gregsadetsky.com/aol-data/

Run time in seconds

Datasets	Bide	SeqKrimp	SQS	GoKrimp
auslan2	0.85	1.0	1.0	0.40
aslbu	74.3	972	277	28
aslgt	73.7	1344	58501	1842
pioneer	11.4	65	15	9
skating	67.3	183	123	85
context	309	402	86	44
unix	1055	47111	84869	1824
jmlr	10	232	890	93
parallel	U/N	U/N	2066	342

The number of patterns

Datasets	Bide	SeqKrimp	SQS	GoKrimp
auslan2	128	4	13	4
aslbu	14620	52	195	67
aslgt	3472	56	1095	68
pioneer	5475	21	143	49
skating	3767	24	140	49
context	6760	15	138	33
unix	28477	75	1070	165
jmlr	4240	23	580	30
parallel	U/N	U/N	17	23

Figure 6-8: The running time and the average update time per sequence of the Zips algorithm in two datasets Tweets and AOL when the stream size increases. Zips scales linearly with the size of the stream.

In the same figure, the running time of the baseline algorithms GoKrimp and SQS are also shown. There are some missing points in the results corresponding to the SQS algorithm because we set a deadline of ten hours for an algorithm to get the results corresponding to a point. The missing points corresponding to the cases when the SQS program did not finish in time.

In the log-log scale, three running time lines resemble a straight line. This result shows that the running time of Zips, GoKrimp and SQS are the power of data size, i.e. $T \sim \alpha|S|^{\beta}$. Using linear fitting functions in log-log scale we found that with the Zips algorithm $\beta = 1.02$ and $\beta = 1.01$ for the AOL and the Tweets datasets respectively, i.e. Zips scales linearly with the data size. Meanwhile, for the SQS algorithm the corresponding exponents are $\beta = 1.91$ and $\beta = 2.2$ and for the GoKrimp algorithm the exponents are $\beta = 2.28$ and $\beta = 2.01$. Therefore, both GoKrimp and SQS do not scale linearly with the data size and hence they are not suitable for data stream applications.

Real-world dataset

In this subsection, we discuss the interpretability of the patterns with three real-world datasets. All three datasets are text so it is easy to interpret the meanings of the set of patterns.

JMLR

In Figure 6-10, we show the first 20 patterns extracted by two baseline algorithms GoKrimp and SQS and the Zips algorithm from the JMLR dataset. Three lists are slightly different

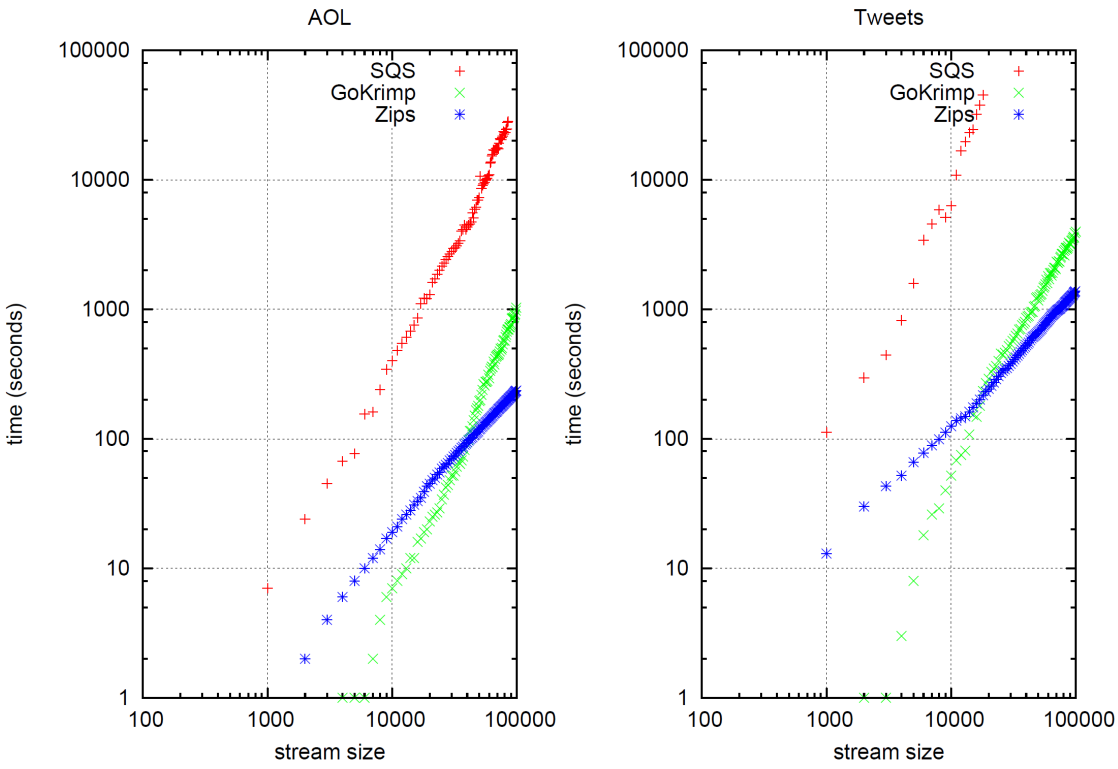


Figure 6-9: Running time (x-axis) against the data size (y-axis) of three algorithms in log-log scales. The Zips algorithm scales linearly with the data size while the GoKrimp and the SQS algorithm scales quadratically with the data size.

but the important patterns such as "support vector machine", "data set", "machine learn", "bayesian network" or "state art" were discovered by all of the three algorithms. This experiment confirms that the Zips algorithm was able to find important patterns that are consistent with the results of state-of-the-art algorithms.

Tweets

Since the tweet dataset is large, we schedule the programs so that they terminate their running after two weeks. The SQS algorithm was not able to finish its running before the deadline while GoKrimp finished running after three days and Zips finished running after 35 hours. The set of patterns extracted by the Zips algorithm and the GoKrimp algorithm are shown in Figure 6-11. Patterns are visualized by the wordcloud tool in R such that more important patterns are represented as larger words. In both algorithms, the sets of patterns are very similar. The result shows the daily discussions of the 1250 twitter accounts about the topics regarding "social media", "Blog post', about "youtube video", about "iphone

Method	Patterns			
SQS	**support vector machin**	**larg scale**	featur select	sampl size
	machin learn	nearest neighbor	graphic model	learn algorithm
	state art	decis tree	**real world**	princip compon analysi
	data set	**neural network**	**high dimension**	logist regress
	bayesian network	cross valid	mutual inform	model select
GOKRIMP	**support vector machin**	**state art**	**neural network**	well known
	real world	**high dimension**	experiment result	special case
	machin learn	reproduc hilbert space	sampl size	solv problem
	data set	**larg scale**	supervis learn	signific improv
	bayesian network	independ compon analysi	support vector	object function
Zips	**support vector machin**	featur select	**high dimension**	cross valid
	data set	**machine learn**	paper propose	decis tree
	real world	**bayesian network**	graphic model	**neutral network**
	learn algorithm	model select	**larg scale**	well known
	state art	optim problem	result show	hilbert space

Figure 6-10: The first 20 patterns extracted from the JMLR dataset by two baseline algorithms GoKrimp and SQS and the Zips algorithm. Common patterns discovered by all the three algorithms are bold.

apps", about greetings such as "happy birthday", "good morning" and "good night", about custom service complaint etc.

AOL

The AOL is a very large dataset. We scheduled the programs so that they terminate their running after two weeks. Both the SQS algorithm and the GoKrimp algorithm did not finish running before the deadline so we don't know the set of patterns extracted by these algorithms. Zips finished running after 58 hours. The result shows how people in the US used the AOL search engine in 2006. Figure 6-12 shows that the users of the AOL search engine (mostly from the US) were mostly interested in looking for information about "real estate", "high school", "community college", "credit union", "credit card" and "cell phone". Especially, the queries regarding locations in the US such as "los angel", "south and north Carolina", "York city" are also popular.

Synthetic dataset with known ground truths

In this subsection, we show the results with three synthetic datasets, i.e. the parallel dataset, the noise dataset and the plant dataset. For the parallel and the noise dataset, all sequences containing a mixture of events generated by different processes or containing at least one noise event are considered as wrong patterns. True patterns are sequences containing events generated by only one process. For the plant dataset, true patterns are ten sequences with exactly 5 events.

Figure 6-11: The top 20 most compressing patterns extracted by Zips (left) and by GoKrimp (right) from the Tweets dataset.

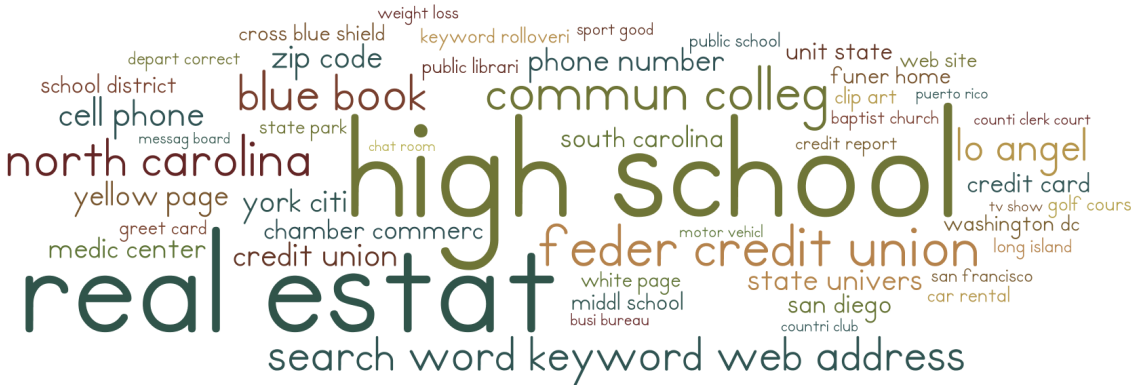


Figure 6-12: Top 50 most compressing patterns extracted by Zips from the AOL dataset.

We get the first ten patterns extracted by each algorithm and calculate the precision and recall at K. Precision at K is calculated as the fraction of the number of right patterns in the first K patterns selected be each algorithm. While the recall is measured as the faction of the number of types of true patterns in the the first K patterns selected be each algorithms. For instance in the parallel dataset, if the set of the first 10 patterns contains only events from the set $\{A_i, B_i, C_i, D_i, E_i\}$ for a given i then the precision at $K = 10$ is 100% while the recall at $K = 10$ is 20%. The precision measures the accuracy of the set of patterns and the recall measures the diversity of the set of patterns.

Figure 6-13 shows the precision and recall at K for $K = 1, 2, \cdots, 10$. For the parallel

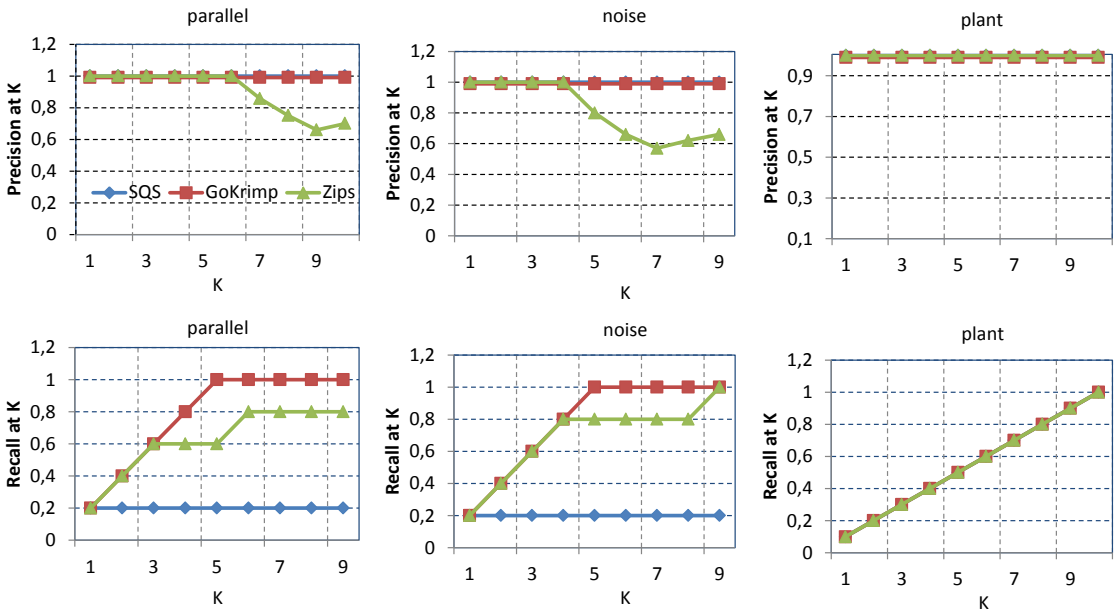


Figure 6-13: The precision and recall at K of three algorithms SQS, GoKrimp and Zips in three synthetic datasets.

and the noise dataset, an interesting result is that all the three algorithms are good at dealing with noise events, none of them return patterns that contain noise events. In term of precision GoKrimp and SQS were able to return all true patterns while the precision of the Zips algorithm is high with small K and the precision starts decreasing as K increases. At $K = 10$ the precision of the Zips algorithm is about about 65% in both datasets.

The SQS algorithm uses an encoding scheme that does not allow interleaving patterns so it returns only one among 5 different pattern types of patterns. Therefore, the recall of the SQS algorithm is low in contrast to the Gokrimp and the Zips algorithms where the recall is very high. The plant dataset is an ideal case when gaps between events of patterns are rare and patterns are not overlapping. In such case, three algorithms return a perfect result.

On the effects of the space-saving technique

The space saving technique requires the Zips algorithm to set the parameter M in advance, i.e. the maximum number of non-singleton dictionary words. As we have discussed in subsection 6.6, the update time is proportional to the value of M. In this subsection, we empirically show that when M is set to a reasonable value the top patterns extracted by

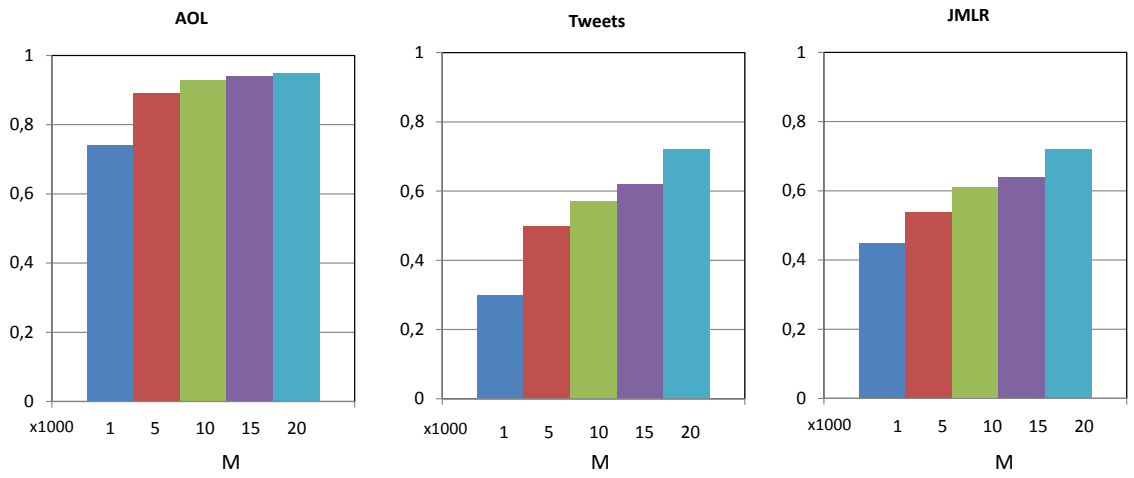


Figure 6-14: Similarity between the lists of the first 100 patterns extracted by the Zips algorithm when M was set to $1000 - 20000$ and M was set to ∞.

the Zips algorithm is very similar to the top patterns when M is set to infinity.

It is important to notice that the case when M is set to ∞ is equivalent to the case when space saving algorithm is not applied. In this subsection, we will compare the results of the cases when $M = \infty$ and the results of the cases when M is increased from 1000 to 20000. When $M = \infty$, for the JMLR dataset, the Zips algorithm scaled up to the size of the entire dataset. However, since the Tweets and the AOL datasets are very large, the Zips algorithm was not able to scale up the size of these datasets. Therefore, for these datasets we report the results for only the first 100000 sequences.

First, we used the Zips algorithm to extract the first 100 patterns from three datasets Tweets, AOL and JMLR when M was set from 1000 to 20000. Then the similarity between two lists L_1 and L_2 with 100 elements each is calculated as $\frac{L_1 \bigcap L_2}{100}$. Figure 6-14 shows the similarity between the top-100 lists when M was set from 1000 to 20000 and the top-100 lists when $M = \infty$.

In the figure, we can see that the similarity of the top-K lists increases when the M increases. When $M = 20000$ the similarity reaches very high value and being close to 1.0. In the AOL dataset, the sequences are shorter so the similarity reaches high values even for small value of M.

Chapter 7

Independent Component Decomposition of a Stream

Given a sequence of events generated by a random mixture of unknown independent processes, we study the problem to split the sequence into the subsequences corresponding to the process that generated it. The sets of events emitted by the processes are pairwise disjoint, but we do not know which event belongs to what process, nor how many different processes there are. In this chapter we propose a compression-based method to solve this decomposition problem. We prove that under the encoding we develop, the decomposition which in expectation results in the optimal compression length, corresponds to an independent decomposition. This theoretical result encourages us to look for the decomposition that incurs the minimum description length to solve the independent decomposition problem. A parameter-free, bottom-up hierarchical clustering algorithm, called Dzip, is proposed to find that decomposition. Both the decision which clusters to merge first in the hierarchical clustering, and when to stop clustering, are based on the minimal description length principle. In experiments with synthetic data with known ground truths, our approach is able to more accurately identify independent subsequences than the existing state-of-the-art algorithm. Moreover, for two real-world datasets, it is shown with randomization tests that the decompositions found by our algorithm are statistically more significant than the decompositions found by the former state-of-the-art algorithm.

7.1 Introduction

The problem

Many processes produce extensive sequences of events, such as: alarm messages from different components of industrial machines or telecommunication networks, web-access logs, click stream data, and records of geographical events. In many cases, a sequence consists of a random mixture of independent or only loosely connected processes in which each process produces a specific disjoint set of events that are independent from the events of the other processes. Given such a sequence, we study the problem of decomposing it into its independent subsequences.

Example 28 (Independent sequence decomposition)**.** *Figure 7-1 shows a sequence and its decompositions into subsequences. In the first decomposition, events in each subsequence are strongly related to each other. For instance, subsequence* $X = ABCABCABC$ *is very regular since every* A *is followed by a* B*, and later a* C*. In the second decomposition, however, the events in the subsequences are at best only loosely connected. For instance, the occurrences of* A*,* E *and* F *in subsequence* U *seem to be independent.*

Motivations

Decompose a sequence into a number of independent subsequences is a very useful data preprocessing step. It facilitates further analysis by allowing for an individual treatment for each independent process separately. For instance, independent sequence decomposition has been reported to improve the accuracy of predictive models [64, 38]. This improvement was achieved by building local predictive model for each of the independent process separately instead of building one global model for the entire sequence. Furthermore, in descriptive data mining, people are usually interested in summarizing data. They are eager to discover the dependency structure between events in the data and also want to know how strong the dependency is in each of the independent components [46]. If the dependency in a component is strong, it may be associated with an explainable context that can help people understand the data better.

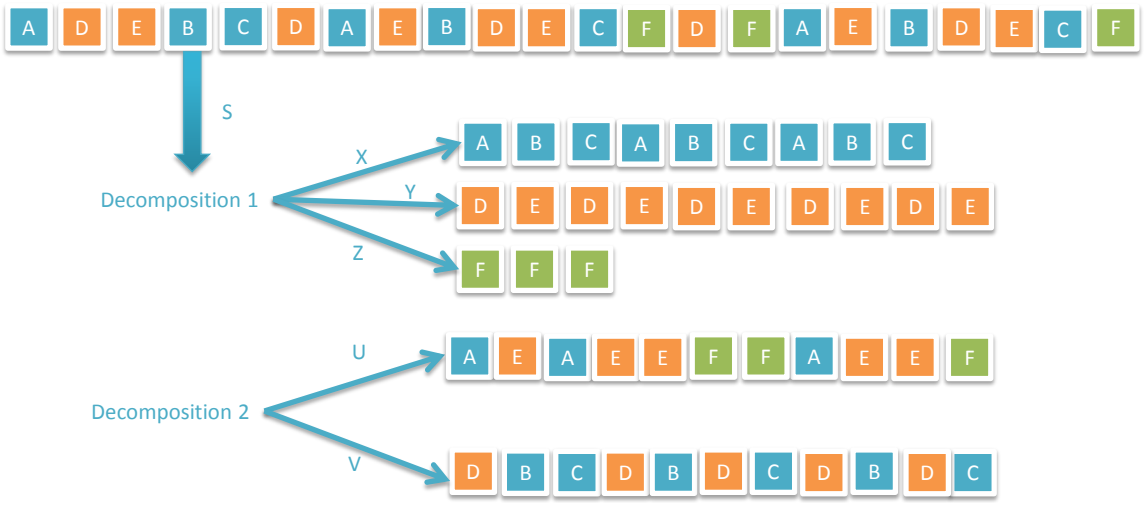


Figure 7-1: A long sequence is decomposed into subsequences. In the first decomposition, the subsequences are very regular, e.g. $X = ABCABCABC$, and events within the same subsequence are tightly related to each other. On the other hand, in the second decomposition, the subsequences show less structure and it seems unlikely all events within a subsequence were generated by the same process.

Existing approach and drawbacks

The sequence independent decomposition problem was first introduced by *Mannila et al.* [47]. They proposed a method based upon statistical hypothesis testing for dependency. This method, however, has a couple of drawbacks:

- The p-value derived from the test is a score subjective to the null hypothesis. Furthermore, the p-value does not convey any information about how strong the dependencies between the events is.
- The method introduces two parameters which require to be tweaked manually for different applications.
- As we discuss in the next section, the method is vulnerable to the discovery of false connections.

Why a data-compression based approach?

In order to illustrate the connection between data compression algorithms and the independent decomposition problem let us revisit the example of Figure 7-1. In the first decomposition, the sequence S is decomposed into three subsequences X, Y, and Z. One way

to compress the sequence S is by encoding every subsequence separately in combination with an encoding of the positions of the subsequences in the original sequence S. All three subsequences X, Y and Z are very regular and a compression algorithm can exploit these regularities to compress the subsequences well.

On the other hand, in the second decomposition, U and V are not very regular, but rather mixtures of independent events. In this case compression algorithms cannot take the advantage of structures to compress them. For this reason, the compression of S using the second decomposition is less efficient than the compression based upon the first decomposition.

This example seems to imply that the independent decomposition is the one that incurs the shortest data description length. Considering a decomposition as a model, this intuition is inline with the general idea of the Minimum Description Length principle [30] which always suggests that the best model is the one that describes the data in the shortest way.

Our contributions

The major contributions can be summarized as follows:

- We propose an MDL-based approach for solving the independent sequence decomposition problem. Instead of blindly accepting MDL as a general principle, we prove first that the best model according to the MDL principle indeed corresponds to an independent decomposition.

- Beside being parameter free, the proposed MDL based approach provides us with a ranking score based on compression ratios showing how strong the dependency between events in an independent component is. In contrast, the state of the art solution highly depends on two parameters and does not provide any ranking score for the strength of the dependencies.

- Experiments with synthetic datasets with known ground truths show that our approach was able to decompose sequences into independent subsequences much more accurately than the state of the art algorithm. For two real-world datasets, with randomization tests it was shown that the decompositions found by our algorithm were statistically more significant than the decompositions returned by baseline algorithm.

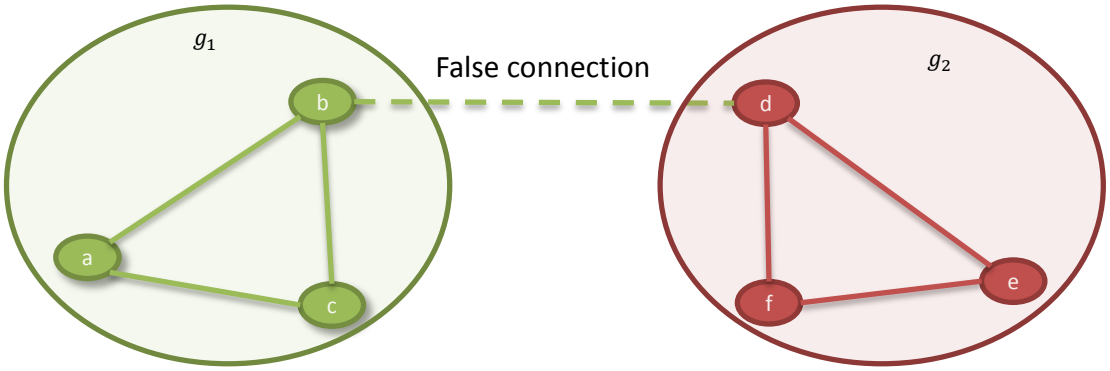


Figure 7-2: An example of two strongly connected and independent components. Due to chance or noise, a connection between b and d was falsely reported. This is problematic, as the Dtest algorithm merges two components even if there is only one connection.

7.2 Related work

The independent decomposition problem for sequences was first studied by Mannila et al. in [47]. The authors proposed a method, which we will denote Dtest (for *Dependency Test*), that is based on statistical hypothesis testing for dependency between events. The algorithm first performs a dependency tests for every pair of events. Subsequently, it builds a dependency graph in which the vertices correspond to events and every pair of dependent events is connected with an edge.

An independent component of the output decomposition corresponds to a connected component of the graph. This approach has one strong drawback: Dtest will merge two components together even if there is only one false connection. Even if only one pair of events is wrongfully recognized as a connection it may result in two independent components being joined together. For instance, Figure 7-2 shows two strongly connected components g_1 and g_2 of the dependency graph. If the dependency test between $b \in g_1$ and $d \in g_2$ wrongly concludes the existence of a connection result, the two independent components g_1 and g_2 will incorrectly be merged into a single component.

In the experiment, we show that unfortunately false connections are common because of the following two reasons. First, the Dtest algorithm has two parameters. Setting these parameters in such a way that false connections are avoided without compromising the true connections, is highly non-trivial. Second, the dependency between b and d can be incorrectly detected due to noise. In contrast to the Dtest algorithm, our compression-based method Dzip does not require parameter tuning and is less vulnerable to noise.

Independent component analysis for other types of data is a well studied problem in the literature [36]. For example, the ICA method was proposed to decompose a multivariate time series into independent components. This method, however, works on continuous data and cannot easily be adapted for univariate categorical data.

Another closely related work concerns the item clustering problem studied in the context of itemset data [46]. The authors [46] propose a method to find clusters of strongly related items for data summarization. The work relies on the MDL principle which clusters items together such that the description length of the data is minimized when the cluster structure is exploited for compressing the data. Our work proposes a different encoding which is suitable for handling sequence data, which is not handled by the encoding of [46]. Additionally, we show a theoretical connection between the MDL principle and the independent component analysis, providing theoretical justification for selecting the MDL approach.

Finally, the idea of using data compression algorithms in data mining is not new. In fact, data compression algorithms were successfully used for many data mining tasks including data clustering [17] and data classification [37]. It was also used for mining non-redundant sets of patterns in itemset data [75] and in sequence data [34]. Our work, however, is the first one to proposes the use of data compression for solving the independent sequence decomposition problem.

7.3 Problem definition

Let $\sum = \{a_1, a_2, \cdots, a_N\}$ be the alphabet of events; we will consider sequences $S_t = x_1 x_2 \cdots x_t$, where each $x_i \in \sum$ is generated by a random variable X_i at time i. $|S|$ denotes the length of sequence S.

We assume that S is generated by a stochastic process $\mathfrak{P}$. For any natural number n, denote $P_t(X_t = x_t, X_{t+1} = x_{t+1}, \cdots, X_{t+n-1} = x_{t+n-1})$ as the joint probability of $X_{t+1}, X_{t+2}, \cdots, X_{t+n-1}$ governed by the stochastic process $\mathfrak{P}$, i.e., the probability of observing the subsequence $x_t x_{t+1} \cdots x_{t+n-1}$ at the time points $t, \ldots, t+n-1$.

A stochastic process is called *stationary* [20] if for any n the joint probability $P_t(X_t = x_t, X_{t+1} = x_{t+1}, \cdots, X_{t+n-1} = x_{t+n-1})$ does not depend on t, i.e., $P_t(X_t = x_t, X_{t+1} = x_{t+1}, \cdots, X_{t+n-1} = x_{t+n-1}) = P_1(X_1 = x_1, X_2 = x_2, \cdots, X_n = x_n)$ for any $t \geq 1$. In our theoretical analysis, we will assume stationary processes only. This is not a strong

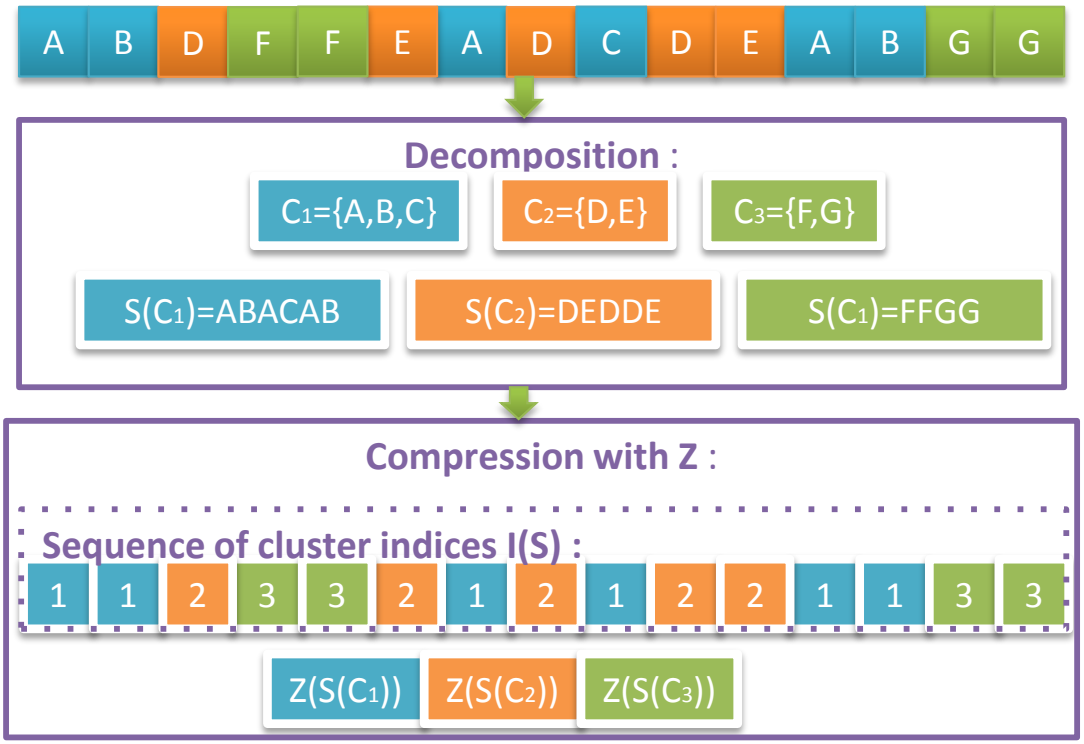


Figure 7-3: Sequence S is decomposed into three subsequences by using the decomposition C_1, C_2 and C_3. The sequence is compressed by encoding every subsequences with the algorithm Z and bookkeeping the cluster indices showing the positions of the subsequences in S.

restriction, as in practice a lot of datasets are generated by a stationary process [20], and, it is not a requirement for our algorithms to operate properly. For a stationary process the t-independent joint probability $P_t(X_t, X_{t+2}, \cdots, X_{t+n-1})$ is simply denoted $P(X_1, X_2, \cdots, X_n)$. For a given sequence S the probability of observing the sequence is denoted $P(S)$.

Let $C = \{C_1, C_2, \cdots, C_k\}$ be a partitioning of the alphabet $\sum$ into k pairwise disjoint subsets. Given a sequence S, the partitioning C decomposes S into k disjoint subsequences denoted $S(C_i)$, for $i = 1, 2, \cdots, k$. Let $P_i(S(C_i) = s)$ denote the marginal distribution defined on the set of subsequences with fixed size $|s| < |S|$.

Example 29 (Sequence decomposition)**.** *Figure 7-3 shows a partitions:* $C = \{C_1, C_2, C_3\}$ *where* $C_1 = \{A, B, C\}$, $C_2 = \{D, E\}$ *and* $C_3 = \{F, G\}$ *of the sequence* $S = ABDFFEADC\ DEABGG$ *into three subsequences* $S(C_1) = ABACAB$, $S(C_2) = DEDDE$ *and* $S(C_3) = FFGG$.

Denote α_i as the probability of observing an event belonging to the cluster C_i. Assume

that $\mathfrak{P}_\mathfrak{i}$ is the stochastic process that generates $S(C_i)$.

Definition 11 (Independent decomposition)**.** *$C = \{C_1, C_2, \cdots, C_k\}$ is an independent decomposition of the alphabet if S is a random mixture of independent subsequences $S(C_i)$:*

$$P\left(S\left(\bigcup_{i=1}^{k} C_i\right)\right) = \prod_{i=1}^{k} \alpha_i^{|S(C_i)|} P_i\left(S(C_i)\right)$$

There are many independent decompositions; we are interested in the decomposition with maximum k. Denote that decomposition as C^*. The independent sequence decomposition problem can be formulated as follows:

Definition 12 (Sequence independent decomposition)**.** *Given a sequence S and an alphabet $\sum$, find the maximum independent decomposition C^* of S.*

Theorem 11 (Unsolvable)**.** *There is no deterministic algorithm that can solve the sequence independent decomposition problem exactly based upon observing a sequence with bounded size M generated by a stochastic (stationary) process.*

Proof. Assume that there is a deterministic algorithm A that can return the maximum independent decomposition exactly when up to $2 * M$ events of a sequence are observed. Consider the following alphabet $\sum = \{a, b\}$ and two different stationary processes:

- The events a and b are drawn independently at random with probability 0.5
- The events a and b are drawn from a simple Markov chain with two states a and b and $P(a \mapsto b) = P(b \mapsto a) = 1.0$

The sequence $S = (ab)^M$ with length $2M$ can be generated by both stationary processes with non-zero probability. Therefore, by observing S, the algorithm A cannot decide the maximum independent decomposition C^* because for the latter process $C^* = \{\{a, b\}\}$ while for the former process $C^* = \{\{a\}, \{b\}\}$. □

Sequence compression

Given an observed sequence with bounded size, Theorem 11 shows that the problem in Definition 12 is unsolvable. However, in this work we show that it can be solved asymptotically

by using data compression algorithms. We first define the encodings that we will use to compress sequence S given a decomposition $C = \{C_1, C_2, \cdots, C_k\}$.

Given an event a_i, $I(a_i) = j$ denotes the identifier of the cluster (partition) C_j which contains a_i. Let $S = x_1 x_2 \cdots x_n$ be a sequence. $I(S)$ denotes the cluster identifier sequence, i.e. $I(S) = I(x_1)I(x_2)\cdots I(x_n)$.

Example 30 (cluster identifier sequence)**.** *The decomposition depicted in Figure 7-3 has the following cluster identifier sequence* $I(ABDFFEADCDEABGG) = 112332121221133$. *Knowing the cluster identifier sequence we can reconstruct the positions of the subsequences in the original sequence.*

Definition 13 (Sequence encoding)**.** *Let Z be a compression algorithm, the sequence encoding of sequence S contains two parts: the cluster identifier sequence $I(S)$ and the compression of individual subsequences using algorithm Z.*

Example 31 (Sequence encoding)**.** *Figure 7-3 shows a sequence encoding with the cluster identifier sequence and three compressed subsequences $Z(S(C_1))$, $Z(S(C_2))$ and $Z(S(C_3))$. If Z is a lossless compression algorithm we can reconstruct the sequence S by first decoding the individual subsequences and using the cluster identifier sequence to get the positions of the subsequences in the original sequence.*

For encoding the cluster identifier sequence, if the distribution of the cluster identifiers $\alpha = (\alpha_1, \alpha_2, \cdots, \alpha_k)$, where $\sum_{j=1}^{k} \alpha_j = 1$, is known, the *Huffman code* [20] can be used to encode each cluster identifier j in the sequence $I(S)$ with a codeword with length proportional to $-\log \alpha_j$. When the identifiers are independent to each other that encoding incurs, in expectation, the minimum compression length for $I(S)$[20]. $E^*(I(S))$ denotes the encoded form of $I(S)$ under that *ideal encoding*. In practice, we don't know the exact distribution $(\alpha_1, \alpha_2, \cdots, \alpha_k)$. However, the distribution can be estimated from data. An encoding is called *asymptotically optimal* if $\lim_{S \mapsto \infty} \frac{|E^+(I(S))|}{|S|} = H(\alpha)$, where H denotes the entropy of the distribution α. An example of an asymptotically optimal encoding E^+, is the one that uses the empirical value $\frac{|S(C_i)|}{|S|}$ as an estimate for α_i.

Z^* is an *ideal compression algorithm* if $|Z^*(S)| = -\log P(S)$. In expectation, Z^* results in the minimum compression length for the data [20]. In practice, however, we don't know the distribution $P(S)$, yet we can use an asymptotic approximation of the ideal compression

algorithm, for instance, the *Lempel-Ziv algorithms* [20]. An algorithm Z^+ is asymptotically optimal if $\lim_{S \mapsto \infty} \frac{|Z^+(S)|}{|S|} = H(\mathfrak{P})$, where $\mathfrak{P}$ is the stationary process that generates S.

In this work, we use the term *ideal encoding* to refer to the encoding that uses E^* and Z^* and *asymptotic encoding* to refer to the encoding that uses E^+ and Z^+.

7.4 Theoretical results

The description length of the sequence S using the decomposition C can be calculated as follows:

$$L^C(S) = |E(|S|)| + |E(I(S))| + \sum_{i=1}^{k} |Z(S(C_i))| \ .$$

In this encoding, the term $|E(|S|)|$ is the cost to encode the length of S which is invariant when S is given. The decomposition C can be considered as a model and the cost to describe that model is equal to $|E(I(S))|$ while the term $\sum_{i=1}^{k} |Z(S(C_i))|$ corresponds to the cost of describing the data given the model C. According to the minimum description length principle we should find a decomposition resulting in the minimum description length in expectation, which is believed to be the best model for describing the data.

This section introduces two key theoretical results (proofs are given in the Appendix): subsection 7.4 shows the analysis for an ideal case for data with a bounded size: under the ideal encoding, the model describing the data best according to the MDL principle, corresponds to an independent decomposition and vice versa. Subsection 7.4 discusses an asymptotic result showing that under an asymptotic encoding, any independent decomposition corresponds to a best model by the definition of the MDL principle.

Analysis under the ideal encoding

We recall some definitions from information theory. Given a discrete probability distribution $P = \{\alpha_1, \alpha_2, \cdots, \alpha_k\}$ where $\sum_{i=1}^{k} \alpha_i = 1$, the *entropy* of the distribution P, denoted $H(P)$, is defined as $-\sum_{i=1}^{k} \alpha_i \log \alpha_i$.

Given a stochastic process $\mathfrak{P}$ which generates sequence S, $H(\mathfrak{P})$ denotes the *entropy rate* (or entropy in short) of the stochastic process $\mathfrak{P}$. Recall that $H(\mathfrak{P})$ is defined as $\lim_{n \mapsto \infty} \frac{1}{n} H(X_1, X_2, \cdots, X_n)$, where $H(X_1, X_2, \cdots, X_n)$ stands for the joint entropy of the random variables $X_1, X_2, \cdots, X_n$. It has been shown that whenever the process $\mathfrak{P}$ is sta-

tionary, the joint entropy $\lim_{n\mapsto\infty}\frac{1}{n}H(X_1, X_2, \cdots, X_n)$ exists [20].

Theorem 12 (MDL versus independent decomposition)**.** *Under an ideal encoding, given data with bounded size, the best decomposition which results in the minimum data description length in expectation is an independent decomposition and vice versa.*

Analysis under the asymptotic encoding

The complete analysis requires an ideal encoding, which is not a realistic assumption. However, we can still prove a similar result under the more relaxed condition of an asymptotically optimal encoding.

Theorem 13 (Asymptotic result)**.** *Under an asymptotical encoding, the data description length in an independent decomposition is asymptotically optimal with probability 1.*

The ideal analysis shows the one-to-one correspondence between the optimal encoding and an independent decomposition. The asymptotic result only shows that an independent decomposition asymptotically corresponds to an optimal encoding. The theorem does not include the reverse correspondence; however, in experiments we empirically show that the correspondence is one-to-one.

7.5 Algorithms

The theoretical analysis in Section 7.4 encourages us to design an algorithm that searches the best decomposition in order to find an independent decomposition. When an independent decomposition is found, the algorithm can then recursively be applied to each independent component to find the maximum independent decomposition. Given data S with alphabet $\sum$ this section discusses a hierarchical clustering algorithm called Dzip to find the desired decomposition.

Algorithm 15 shows the main steps of the Dzip algorithm. It starts from N clusters where each cluster contains only one character of the alphabet. Subsequently, it evaluates the compression benefit of merging any pair of clusters. The best pair of clusters, that is, the pair that delivers the highest compression benefit, is chosen to be merged. These steps are repeated until none of the remaining pairs give any additional compression benefit anymore.

Algorithm 15 Dzip(S)

Input: a sequence S, an alphabet $\sum = \{a_1 a_2 \cdots a_N\}$
Output: a decomposition C
$C \leftarrow \{C_1 = \{a_1\}, C_2 = \{a_2\}, \cdots, C_n = \{a_n\}\}$
while true **do**
  $max \leftarrow 0$
  $C^* \leftarrow C$
  for $i = 1$ **to** $|C|$ **do**
    for $j = i + 1$ **to** $|C|$ **do**
      $C^+ \leftarrow C$ with merged C_i and C_j
      if $L^C(S) - L^{C^+}(S) > max$ **then**
        $max \leftarrow L^C(S) - L^{C^+}(S)$
        $C^* \leftarrow C^+$
      end if
    end for
  end for
  if $|C^*| = 1$ or $max = 0$ **then**
    Return C^*
  end if
end while

In theory, Dzip could recursively be applied again on each cluster to get the maximum decomposition. However, in our experiments we observed that in most of the cases the cluster found by Dzip cannot be decomposed further because of the bottom-up process which already checked for the benefits of splitting the cluster.

Dzip uses the *Lempel-Ziv 78* (LZ78) implementation [20] which has linear complexity in the size of the data. It utilizes an inverted list as data structure to store the list of positions of each character in the sequence. Moreover, Dzip also caches the compression

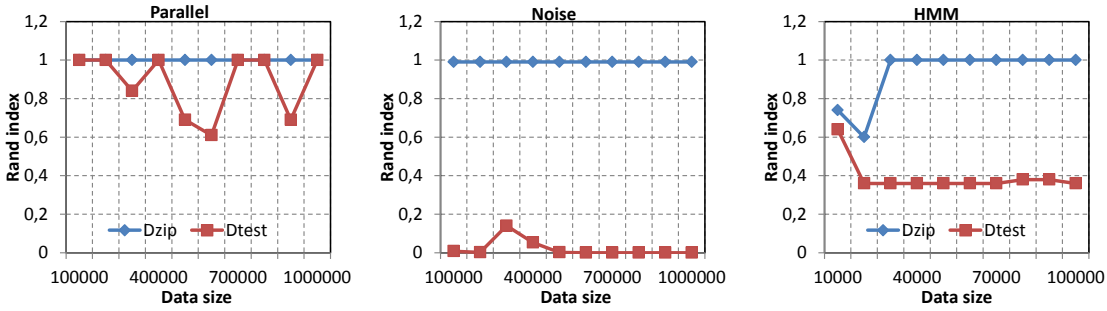


Figure 7-4: Plots of Rand index versus sequence size. Rand index measures the similarity (higher is better) between the decompositions produced by the algorithms and the ground-truth. The Rand index is equal to 1 if the decomposition optimally corresponds to the ground truth.

size of merged clusters. In doing so, in the worst case the computational complexity of Dzip is bounded by $O(|S|N^2)$. This complexity is the same as the amortized complexity of the Dtest algorithm [47].

7.6 Experiments

We consider the current state-of-the-art technique, Dtest [47], as a baseline approach. All experiments were carried out on a 16 processor cores, 2 Ghz, 12 GB memory, 194 GB local disk, Fedora 14 / 64-bit machine. The source code of out implementation in Java of Dtest and Dzip, and the datasets are available for download from our project website[1].

Dtest has two parameters: the significance value α and the gap number G. We choose $\alpha = 0.01$ and $G = 300$ as is recommended in the work [47]. We also tried to vary α and G and observed that this resulted in quite different results. The running time of the algorithm increases with increasing G, and smaller values of α results in lower false positive yet higher false negative rate and vice versa. Despite the high variability in the results produced by Dtest for different parameter settings, the overall conclusions of the comparison with Dzip in our experiments does not change significantly. That is, there is no "optimal" choice of parameter setting that improves the performance of Dtest in the comparison.

Synthetic data with known ground truth

We experimented with three datasets of which the ground truth is known:

- *Parallel:* is a synthetic dataset which mimics a typical situation in practice where a data stream is generated by mixing five independent parallel processes. Each process P_i $(i = 1, 2, \cdots, 5)$ repeatedly generates the sequence of events A_i, B_i, C_i, D_i, E_i. In each step, the generator chooses one of five processes uniformly at random and generates the next event from that process, until the total stream length reaches 1 million events.

- *Noise:* is generated in the same way as the parallel dataset but with additional noise. A noise source generates independent events from a noise alphabet with size 1000. Noise events are randomly mixed with the parallel dataset. The amount of noise is

[1] `http://www.win.tue.nl/~lamthuy/dzip.htm`

20% of the parallel data; i.e., in the end 1 event out of 6 is a noise event. This dataset is considered to measure the sensitivity to noise of Dtest and Dzip.

- *HMM:* is generated by a random mixture of two hidden Markov models. Both hidden Markov models have 10 hidden and 5 observed states. The transition matrix and the emission matrix are generated randomly according to a standard normal distribution with mean in the diagonal of the matrix. Each Markov model generates 50000 events and the mixture of them contains 100000 events. This dataset is considered to see the performance of the methods in a dataset with weak dependencies.

Since the ground-truth for these three datasets is known, we can use the *Rand index* [65] to compare the partitionings of the alphabet set generated by the algorithms. The rand index calculates the number of pairs of events that are either in the same cluster in both partitionings or in different clusters in both partitionings. This number is normalized to the interval $[0, 1]$ by dividing by the total number of different pairs. The rand index measures the degree of agreement between the two partitionings, where a value of 1 indicates a perfect match, and 0 completely disagreement.

Figure 7-4 shows the rand indices (y-axis) of the two algorithms when the sequence size (x-axis) is varied. It is clear from the figure that Dzip finds the correct decompositions with much higher accuracy than the Dtest algorithm. In the Parallel and Noise datasets, Dzip is able to return perfect decompositions when the data size is large. When the data size is smaller the performance degrades somewhat although the rand index remains at an respectable level.

Dtest is good in the Parallel dataset although it is not stable when the data size is varied. Dtest, however, does not work well in the Noise and the HMM dataset, especially when there is a lot of noise. In both datasets, after a visual inspection, it was observed that Dtest has clustered all events in one big cluster; this experiment confirms our claim in section 7.2 that Dtest is vulnerable to noise.

Real-life data

Masters portal is a click-through log of user behaviors in the *Masters Portal*[2] website. It contains about 1.7M events of 16 different activity types. The decomposition results are

[2] `www.mastersportal.eu`

shown in Figures 7-5. There are two major clusters, each containing three activities. The first cluster can be interpreted as a set of actions that can be performed on search results. The second cluster mainly contains activities that users perform when they input a search query, refine the search and click on some links in the search results. The Dtest algorithm returns only one cluster which groups all these events together.

Since the ground truth is unknown, we performed a randomization test to see how statistically significant the decompositions are. First, we sampled 1000 decompositions uniformly at random and calculated the compression ratio of the compression using these decompositions. The distribution of compression ratios of these decompositions is plotted in Figure 7-6.A., where the x-axis shows compression ratios and the y-axis shows the number of decompositions. The highlighted points with compression ratios 0.9742 and 1.033 correspond to the decompositions of the Dtest and the Dzip algorithm respectively. It is clear from the figure that Dzip can find the decomposition with much more statistical significant compression ratio with p-value $3 * 10^{-6}$ for $Dzip$ and 0.05 for $Dtest$.

Figure 7-6.A. also shows an interesting fact that the distribution of compression ratio of random decomposition resembles a normal distribution with a large part of the probability mass falling into the interval less than 1.0. Only a few number of decompositions has compression ratio greater than 1.0. This result shows that the Masters Portal dataset has very loose dependency structure.

Machine log is a message log containing about 2.7 million messages (more than 1700 distinct message types) produced by different components of a photolithography machine. We removed infrequent messages which occur less than in 0.1 percent of the data. After preprocessing the data, the size is reduced to 2.6 million messages of 50 different message types.

For this dataset, the Dzip algorithm produced 2 clusters: one major cluster with 47 messages and another smaller one with 3 messages. The first cluster is difficult to interpret because it contains a lot of messages. However, the messages in the second cluster are:

- WH-8093,"Difference between Robot Inner Axis absolute and incremental encoder"
- WH-8094,"Error in communication between Robot Inner Axis and absolute encoder"
- WH-8099,"Difference between Robot Outer Axis absolute and incremental encoder"

Clusters	Compression ratios
No search result Program in search results Program in related programs	1.05
Basic search Refine search Click on program link	1.02
Quick search	1.0
Click on banner	1.0
Click on university link click	1.0
Click on country link	1.0
Submit inquiry	1.0
Question submit	1.0
File view	1.0
Program in landing page	1.0
Impression of Universities in spotlight	1.0
University on nearby universities	1.0

Figure 7-5: Decomposition returned by Dzip in the Master Portal dataset.

These messages seems to be related to each other because they are all concerning the critical errors regarding the differences between the coordinates of the robot and the encoder of the machine. On the other hand, the Dtest algorithm produced only one cluster which grouped all the messages together.

Since the ground truth is unknown, we performed similar randomization tests as for the masters portal dataset. The results are depicted in Figure 7-6.B. It is clear from the figure that both decompositions by Dtest and Dzip are more statistical significant than random decompositions. However, the decomposition by Dzip is more significant than the decomposition by Dtest with p-values 10^{-11} and $4 * 10^{-11}$ respectively.

Figure 7-6.B. shows another interesting fact that the distribution of compression ratio of random decomposition resembles two normal distributions with a large part of the probability mass falling into the interval greater than 1.0. This result shows that this dataset has very strong dependency structure. This is in contrast to the Master Portal dataset where the dependency between events is very loose.

Running time

In section 7.5, we have shown that the amortized time complexity of the Dtest algorithm and the worst case complexity of the Dzip algorithm are the same. The result promises

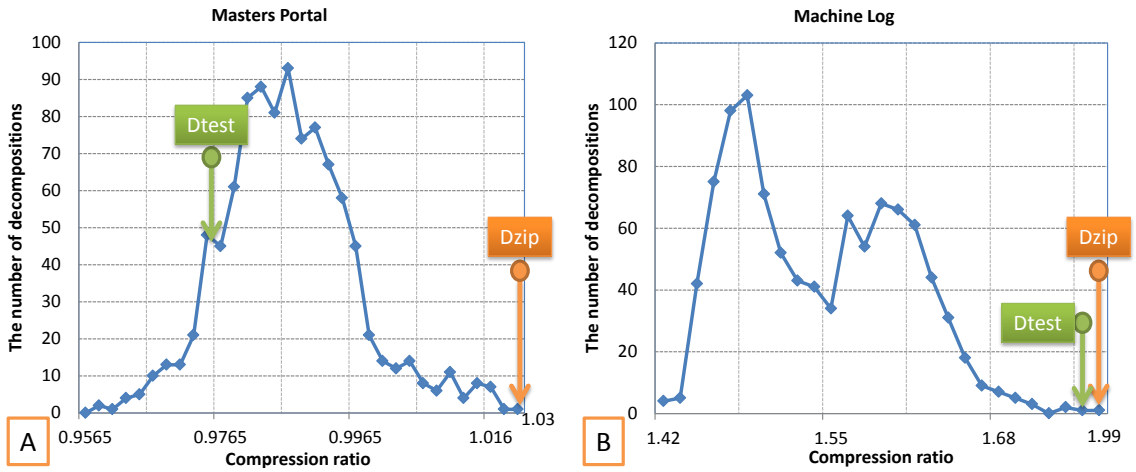


Figure 7-6: Distribution of compression ratio of random decompositions. The highlighted points correspond to the decompositions returned by the Dzip and the Dtest algorithm. The results by Dzip is much more statistical significant than the results by Dtest with much smaller p-value.

	Masters Portal	Machine log	Parallel	Noise	HMM
Dzip	28	1312	12	328	1
Dtest	159	1402	138	25130	1

Figure 7-7: Running time in seconds of two algorithms. Dzip is about an order of magnitude faster than Dtest.

that the Dzip algorithm will be faster than the Dtest algorithm. Indeed, this fact holds for the collection of datasets we use in this work. In Figure 7-7 we compare Dzip and Dtest in terms of running time. In most datasets, Dzip is about an order of magnitude faster than the Dtest algorithm.

Appendix

Proof of Theorem 12. Given a decomposition $C = \{C_1, C_2, \cdots, C_k\}$, for a given n assume that S is a sequence of length n. Under an ideal encoding, the description length of the cluster identifier sequence of S is $|E^*(I(S))| = -\sum_{i=1}^{k} |S(C_i)| \log \alpha_i$. In the ideal encoding, since the length of the compressed subsequence $Z^*(S(C_i))$ is $|Z^*(S(C_i))| = -\log P_i(S(C_i))$

the total description length is:

$$
\begin{aligned}
L^C(S) &= |E(n)| - \sum_{i=1}^{k} |S(C_i)| \log \alpha_i && (7.1)\\
&\quad - \sum_{i=1}^{k} \log P_i(S(C_i)) && (7.2)\\
E(L^C(S)) &= \sum_{S:|S|=n} P(S)(|E(n)| - && (7.3)\\
&\quad \sum_{i=1}^{k} |S(C_i)| \log \alpha_i - \sum_{i=1}^{k} \log P_i(S(C_i))) && (7.4)\\
E(L^C(S)) &= |E(n)| - && (7.5)\\
&\quad \sum_{S:|S|=n} P(S) \log \prod_{i=1}^{k} \alpha_i^{|S(C_i)|} P_i(S(C_i)) && (7.6)\\
E(L^C(S)) &= |E(n)| + H_P(X_1, X_2, \cdots, X_n) && (7.7)\\
&\quad + D(P|Q) && (7.8)
\end{aligned}
$$

where Q is the random mixture of the distributions P_i defined on the space of all sequence $S : |S| = n$, i.e. $Q(S) = \prod_{i=1}^{k} \alpha_i^{|S(C_i)|} P_i(S(C_i))$ and $D(P|Q)$ is the relative entropy or the *Kullback-Leibler* distance between P and Q. Since $D(P|Q) \geq 0$ [20] we have $E(L^C(S)) \geq |E(n)| + H_P(X_1, X_2, \cdots, X_n)$. The equality happens if and only if $D(P|Q) = 0$, i.e. $P \equiv Q$ which proves the theorem. □

In order to prove Theorem 13, we first prove a basic supporting lemma. The lemma is a generalized result of the *Cesàro mean*.

Lemma 18. *Given a sequence* (a_n)*, a sequence* (c_n) *is defined as :* $c_n = \sum_{i=1}^{n} b_i(n) a_i$ *where* $\sum_{i=1}^{n} b_i(n) = 1$ *and* $b_i(n) > 0 \ \forall n > 0$*. If* $\lim_{n \mapsto \infty} a_n = A$ *and* $\lim_{n \mapsto \infty} b_i(n) = 0 \ \forall i > 0$ *then we also have* $\lim_{n \mapsto \infty} c_n = A$*.*

Proof. Since $\lim_{n \mapsto \infty} a_n = A$ given any number $\epsilon > 0$ there exists N such that $|a_n - A| < \frac{\epsilon}{2}$ $\forall n > N$. Moreover, because $\lim_{n \mapsto \infty} a_n = A$ there exists an upper bound D on $|a_i - A|$.

Given N, since $\lim_{n \mapsto \infty} b_i(n) = 0$ we can choose M_i $(i = 1, 2, \cdots, N)$ such that $b_i(n) < \frac{\epsilon}{2ND}$ $\forall n > M_i$. Let us denote M as the maximum value of the set $\{N, M_1, M_2, \cdots, M_N\}$. For

any $n > M$, we have:

$$
\begin{aligned}
|c_n - A| &= |\sum_{i=1}^{n} b_i(n)a_i - A| && (7.9)\\
&\leq |\sum_{i=1}^{N-1} b_i(n)(a_i - A)| && (7.10)\\
&\quad +|\sum_{i=N}^{n} b_i(n)(a_i - A)| && (7.11)\\
&\leq (N-1)D\frac{\epsilon}{2ND} + \frac{\epsilon}{2} && (7.12)\\
&\leq \epsilon && (7.13)
\end{aligned}
$$

The last inequality proves the lemma. □

Theorem 14 (Entropy of an independent decomposition)**.** *Assume that* $C = \{C_1, C_2, \cdots, C_k\}$ *is an independent decomposition and* $\mathfrak{P}_\mathfrak{i}$ *is the stochastic process that generates* $S(C_i)$*. Denote* α_i *as the probability that we observe an event belonging to the cluster* C_i*, we have:*

$$H(\mathfrak{P}) = \sum_{i=1}^{k} \alpha_i H(\mathfrak{P}_\mathfrak{i}) + H(\alpha_1, \alpha_2, \cdots, \alpha_k) \tag{7.14}$$

Proof. We first prove a special case with $k = 2$ from which the general case for any k can be directly implied. In fact, by the definition of the joint entropy we can perform simple calculations as follows:

$$
\begin{aligned}
H(X_1, X_2, \cdots, X_n) &= -\sum_{S:|S|=n} P(S)\log P(S) && (7.15)\\
&\quad -\sum_{S:|S|=n} \alpha_1^{|S(C_1)|} P_1(S(C_1))\alpha_2^{|S(C_2)|} && (7.16)\\
P_2(S(C_2))\log\Big(\alpha_1^{|S(C_1)|} P_1(S(C_1))\alpha_2^{|S(C_2)|} P_2(S(C_2))\Big) && (7.17)\\
= C_n^{|S_1|} \sum_{S_1:|S_1|\leq n} \sum_{S_2:|S_2|=n-|S_1|} \alpha_1^{|S_1|} P_1(S_1)\alpha_2^{|S_2|} P_2(S_2) && (7.18)\\
\log\Big(\alpha_1^{|S_1|} P_1(S_1)\alpha_2^{|S_2|} P_2(S_2)\Big) && (7.19)
\end{aligned}
$$

We denote each term of Equation 7.19 as follows:

$$X = -C_n^{|S_1|} \sum_{S_1:|S_1|\leq n} \sum_{S_2:|S_2|=n-|S_1|} \alpha_1^{|S_1|} P_1(S_1) \tag{7.20}$$

$$\alpha_2^{|S_2|} P_2(S_2) \log \alpha_1^{|S_1|} \tag{7.21}$$

$$Y = -C_n^{|S_1|} \sum_{S_1:|S_1|\leq n} \sum_{S_2:|S_2|=n-|S_1|} \alpha_1^{|S_1|} P_1(S_1) \tag{7.22}$$

$$\alpha_2^{|S_2|} P_2(S_2) \log \alpha_2^{|S_2|} \tag{7.23}$$

$$Z = -C_n^{|S_1|} \sum_{S_1:|S_1|\leq n} \sum_{S_2:|S_2|=n-|S_1|} \alpha_1^{|S_1|} P_1(S_1) \tag{7.24}$$

$$\alpha_2^{|S_2|} P_2(S_2) \log P_1(S_1) \tag{7.25}$$

$$T = -C_n^{|S_1|} \sum_{S_1:|S_1|\leq n} \sum_{S_2:|S_2|=n-|S_1|} \alpha_1^{|S_1|} P_1(S_1) \tag{7.26}$$

$$\alpha_2^{|S_2|} P_2(S_2) \log P_2(S_2) \tag{7.27}$$

We calculate each term of Equation 7.19 as follows:

$$X = -\sum_{i=0}^{n} C_n^i \sum_{|S_1|=i} \sum_{|S_2|=n-i} \alpha_1^i \alpha_2^{n-i} P_1(S_1) \tag{7.28}$$

$$P_2(S_2) \log \alpha_1^i \tag{7.29}$$

$$= -\sum_{i=0}^{n} C_n^i \alpha_1^i \alpha_2^{n-i} \log \alpha_1^i \tag{7.30}$$

$$\sum_{|S_1|=i} \sum_{|S_2|=n-i} P_1(S_1) P_2(S_2) \tag{7.31}$$

$$= -\sum_{i=0}^{n} C_n^i \alpha_1^i \alpha_2^{n-i} \log \alpha_1^i \tag{7.32}$$

$$= -\log \alpha_1 \sum_{i=0}^{n} i C_n^i \alpha_1^i \alpha_2^{n-i} \tag{7.33}$$

$$= -n\alpha_1 \log \alpha_1 \tag{7.34}$$

With similar calculation we have $Y = -n\alpha_2 \log \alpha_2$. We continue with the calculation of Z:

$$\begin{aligned} Z &= -\sum_{i=0}^{n} C_n^i \sum_{|S_1|=i} \sum_{|S_2|=n-i} \alpha_1^i \alpha_2^{n-i} && (7.35)\\ & P_1(S_1)P_2(S_2)\log P_1(S_1) && (7.36)\\ &= -\sum_{i=0}^{n} C_n^i \sum_{|S_1|=i} \alpha_1^i \alpha_2^{n-i} P_1(S_1)\log P_1(S_1) && (7.37)\\ & \sum_{|S_2|=n-i} P_2(S_2) && (7.38)\\ &= -\sum_{i=0}^{n} C_n^i \sum_{|S_1|=i} \alpha_1^i \alpha_2^{n-i} P_1(S_1)\log P_1(S_1) && (7.39)\\ &= -\sum_{i=0}^{n} C_n^i \alpha_1^i \alpha_2^{n-i} \sum_{|S_1|=i} P_1(S_1)\log P_1(S_1) && (7.40)\\ &= \sum_{i=1}^{n} C_n^i \alpha_1^i \alpha_2^{n-i} H_{P_1}(X_1, X_2, \cdots, X_i) && (7.41) \end{aligned}$$

With similar calculation we also have $T = \sum_{i=1}^{n} C_n^i \alpha_1^{n-i} \alpha_2^i H_{P_2}(X_1, X_2, \cdots, X_i)$. Therefore we further imply that:

$$\begin{aligned} H(X_1, X_2, \cdots, X_n) &= X + Y + Z + T && (7.42)\\ &= nH(\alpha_1, \alpha_2) + && (7.43)\\ & \sum_{i=1}^{n} C_n^i \alpha_1^i \alpha_2^{n-i} H_{P_1}(X_1, X_2, \cdots, X_i) + && (7.44)\\ & \sum_{i=1}^{n} C_n^i \alpha_1^{n-i} \alpha_2^i H_{P_2}(X_1, X_2, \cdots, X_i) && (7.45) \end{aligned}$$

$$\begin{aligned} \frac{1}{n} H(X_1, \cdots, X_n) &= H(\alpha_1, \alpha_2) + && (7.46)\\ \alpha_1 \sum_{i=1}^{n} C_{n-1}^{i-1} \alpha_1^{i-1} \alpha_2^{n-i} \frac{1}{i} H_{P_1}(X_1, \cdots, X_i) && (7.47)\\ +\alpha_2 \sum_{i=1}^{n} C_{n-1}^{i-1} \alpha_1^{n-i} \alpha_2^{i-1} \frac{1}{i} H_{P_2}(X_1, X_2, \cdots, X_i) && (7.48) \end{aligned}$$

Since $\lim_{n\mapsto\infty} \frac{1}{n} H(X_1, \cdots, X_n) = H(\mathfrak{P})$, and $\lim_{n\mapsto\infty} \frac{1}{n} H_{P_1}(X_1, \cdots, X_n) = H(\mathfrak{P}_1)$, moreover $\lim_{n\mapsto\infty} \frac{1}{n} H_{P_2}(X_1, \cdots, X_n) = H(\mathfrak{P}_2)$ therefore according to Lemma 18 from the last equation

we can imply that $H(\mathfrak{P}) = \alpha_1 H(\mathfrak{P}_1) + \alpha_2 H(\mathfrak{P}_2) + H(\alpha_1, \alpha_2)$.

The last result can be easily generalized for an independent decomposition with any k clusters by induction. Indeed, we assume that the theorem is correct with $k = l$ we prove that the result holds for $k = l + 1$. Denote α as $\sum_{i=1}^{l} \alpha_i$. Given two independent stochastic processes $\mathfrak{P}$ and $\mathfrak{Q}$ denote the random mixture of them as $\mathfrak{P} \bigoplus \mathfrak{Q}$. Consider the process defined as the random mixture of $\mathfrak{P}_1, \mathfrak{P}_2 \cdots \mathfrak{P}_l$ denoted as $\mathfrak{P}_1 \bigoplus \mathfrak{P}_2 \bigoplus \cdots \bigoplus \mathfrak{P}_l$. Since $\mathfrak{P}_{l+1}$ and $\mathfrak{P}_1 \bigoplus \mathfrak{P}_2 \bigoplus \cdots \bigoplus \mathfrak{P}_l$ are independent we have:

$$
\begin{aligned}
H(\mathfrak{P}) &= H(\mathfrak{P}_1 \bigoplus \mathfrak{P}_2 \bigoplus \cdots \bigoplus \mathfrak{P}_{l+1}) && (7.49)\\
&= \alpha H(\mathfrak{P}_1 \bigoplus \mathfrak{P}_2 \bigoplus \cdots \bigoplus \mathfrak{P}_l) + && (7.50)\\
&\quad \alpha_{l+1} H(\mathfrak{P}_{l+1}) + H(\alpha, \alpha_{l+1}) && (7.51)
\end{aligned}
$$

Moreover by the induction assumption we further imply that: $H(\mathfrak{P}_1 \bigoplus \mathfrak{P}_2 \bigoplus \cdots \bigoplus \mathfrak{P}_l) = \sum_{i=1}^{k} \frac{\alpha_i}{\alpha} H(\mathfrak{P}_i) + H(\frac{\alpha_1}{\alpha}, \frac{\alpha_2}{\alpha}, \cdots, \frac{\alpha_l}{\alpha})$. By replacing this value to Equation 7.51 we can obtain Equation 7.14 from which the theorem is proved. □

Theorem 14 shows that the entropy of the stochastic process $\mathfrak{P}$ can be represented as the sum of two meaningful terms. The first term $H(\alpha_1, \alpha_2, \cdots, \alpha_k)$ actually corresponds to the average cost per element of the cluster identifier. Meanwhile the second term $\sum_{i=1}^{k} \alpha_i H(\mathfrak{P}_i)$ corresponds to the average cost per element to encode the subsequences $S(C_i)$. By that important observation we now can prove Theorem 13:

Proof of Theorem 13. Given an independent decomposition $C = \{C_1, C_2, \cdots, C_k\}$, for any n assume that S is a sequence with length n. Under an asymptotic encoding, the description length of the data is:

$$L^C(S) = |E(n)| + |E^+(I(S))| + \sum_{i=1}^{k} Z^+(S(C_i))$$

$$\frac{L^C(S)}{|S|} = \frac{|E(n)|}{|S|} + \frac{|E^+(I(S))|}{|S|} + \frac{\sum_{i=1}^{k} Z^+(S(C_i))}{|S|}$$

$$\frac{L^C(S)}{|S|} = \frac{|E(n)|}{|S|} + \frac{|E^+(I(S))|}{|S|} + \sum_{i=1}^{k} \frac{|S(C_i)|}{|S|} \frac{Z^+(S(C_i))}{|S(C_i)|}$$

$$Pr\left(\lim_{|S|\mapsto\infty}\frac{L^C(S)}{|S|} = H(\alpha_1,\alpha_2,\cdots,\alpha_k) + \sum_{i=1}^{k}\alpha_i H(\mathfrak{P}_\mathrm{i})\right) = 1$$

$$Pr\left(\lim_{|S|\mapsto\infty}\frac{L^C(S)}{|S|} = H(\mathfrak{P})\right) = 1$$

The last equation is a direct result of Theorem 14. Since $H(\mathfrak{P})$ is the lower-bound on the expectation of the average compression size per element of any data compression algorithm the encoding using the independent decomposition is asymptotically optimal. □

Chapter 8

Conclusions

Recent applications in heath care systems, social media, sensor and mobile networks produce different types of data streams. These streams are often monitored and analyzed in real-time. Solving pattern mining problems in real-time with additional constraints on memory limit, high-speed stream updates and the single-pass condition is very challenging. In this doctoral dissertation, we addressed these problems in the streaming context, the conclusions of each chapter are as follows:

- Chapter 2: We studied the problem of mining top-k most frequent items with respect to the MaxFreq measure in the flexible sliding window model. We showed a lower-bound on the memory usage of any exact deterministic algorithm for computing the top-k most frequent items in a stream. Although the lower-bound on the memory usage is prohibitive for applications with limited memory such as mobile devices or sensor networks, fortunately, memory-efficient approximation algorithms exist. We proposed an approximation algorithm using significantly less memory than the straightforward approach while preserving the high accuracy of the top-k most frequent item results. The experiment conducted with real-life and synthetic datasets confirms the significance of the proposed algorithm.

- Chapter 3: We introduced and discussed the online *top-k* most similar motifs discovery problem. A theoretical memory lower-bound for any exact algorithm was shown. This lower-bound is tight and easily achievable by simply storing the complete window and recomputing the top-k most similar motif from scratch. We proposed an data stream summary approach that uses optimal memory and is capable of handling quick

updates for answering continuous top-k most similar motif queries. Experiments with real-world datasets showed that the proposed approach dominated the straightforward solution and the state-of-the-art solution in terms of memory usage, update time and query response time.

- Chapter 4: We proposed two algorithms for mining top-k largest tiles from a data stream in the sliding window model. The first candidate-based algorithm *cTile* solves the problem exactly but the update time is slow when the window size increases. The second one is an efficient approximation algorithm with theoretical bounds on the error rate. Experiments with two real-life datasets showed that the approximation algorithm can find large tiles with high accuracy while being an order of magnitude faster than the candidate based algorithm and two to three order of magnitude faster than the other naive algorithms that use the large tile mining algorithm for databases to find large tiles in streams. We also showed a potential application of large tiles mining in data stream through a use-case in which top-k largest tiles of the AOL query log stream are visualized by the *wordcloud* tool to create a stream of snapshots which shows the evolution of tiles overtime.

- Chapter 5: We have explored mining of sequential patterns that compress the data well utilizing the MDL principle. A key contribution is our encoding scheme targeted at sequence data. We have shown that mining the most compressing pattern set is NP-Hard and designed two algorithms to approach the problem. SeqKrimp is a candidate-based algorithm that turned out to be sensitive to parameter settings and inefficient due to the candidate generation phase. GoKrimp is an algorithm that directly looks for compressing patterns and was shown to be effective and efficient. The experiments showed that the most compressing patterns are less redundant and better than the frequent closed patterns as feature sets for different classifiers. The dependency test technique used in the GoKrimp algorithm was shown to be very useful to speed up the GoKrimp algorithm significantly. Both GoKrimp and SeqKrimp were shown to be effective in finding non-redundant and meaningful patterns. However, the GoKrimp algorithm is 1-2 orders of magnitude faster than the SeqKrimp algorithm and the SQS algorithm.

- Chapter 6: We studied the mining compressing patterns problem in a data stream. A

new encoding scheme for sequences was proposed. The new encoding is convenient for streaming applications because it allows us to encode the data in an online manner. We proposed an efficient solution that solves the problem effectively. In the experiments with three synthetic datasets with known ground-truths, the proposed algorithm was able to extract the most compressing patterns with high accuracy. Meanwhile, in the experiments with three real-world datasets it could find patterns that are similar to the-state-of-the-art algorithms extract from these datasets. More importantly, the proposed algorithm was able to scale linearly with the size of the stream while the-state-of-the-art algorithms were not.

- Chapter 7: Our theoretical analysis showed the correspondence between the problem of independent component decomposition and the problem of finding the optimal encoding of the stream given a decomposition. This theoretical result encouraged us to design a compression-based method called Dzip for the sequence independent decomposition problem. Beside being justified by a theoretical analysis, in experiments with both synthetic and real-life datasets, Dzip was shown to be more effective than the state of the art method based on statistical hypothesis testing.

Based on the results of the dissertation we can draw some general conclusions as follows:

1. *"The results of chapters 2, 4 and 6 show that despite the fact that many pattern mining problems in data streams are theoretically intractable efficient practical approximation solutions do exist. These solutions work well for real-world datasets."*

2. *"It has been shown that many data mining tasks including data clustering [17], classifications [37] can be solved effectively using data compression techniques. The result of this thesis work (chapter 5, 6 and chapter 7 on pattern mining using data compression) is an additional strong evidence to the results of previous work [17, 37, 75] showing that data mining is equivalent to data compression."*

Bibliography

[1] Charu C. Aggarwal, editor. *Data Streams - Models and Algorithms*, volume 31 of *Advances in Database Systems*. Springer, 2007.

[2] Christoph Ambühl, Monaldo Mastrolilli, and Ola Svensson. Inapproximability results for maximum edge biclique, minimum linear arrangement, and sparsest cut. *SIAM J. Comput.*, 40(2):567–596, 2011.

[3] Roberto Bayardo. Efficiently mining long patterns from databases. In *SIGMOD Conference*, pages 85–93, 1998.

[4] Jérémy Besson, Céline Robardet, and Jean-François Boulicaut. Mining formal concepts with a bounded number of exceptions from transactional data. In *KDID*, pages 33–45, 2004.

[5] Tijl De Bie. Maximum entropy models and subjective interestingness: an application to tiles in binary databases. *Data Min. Knowl. Discov.*, 23(3):407–446, 2011.

[6] Tijl De Bie, Kleanthis-Nikolaos Kontonasios, and Eirini Spyropoulou. A framework for mining interesting pattern sets. *SIGKDD Explorations*, 12(2):92–100, 2010.

[7] Mario Boley, Claudio Lucchese, Daniel Paurat, and Thomas Gärtner. Direct local pattern sampling by efficient two-step random procedures. In *KDD*, pages 582–590, 2011.

[8] Björn Bringmann, Siegfried Nijssen, and Albrecht Zimmermann. Pattern-based classification: A unifying perspective. *CoRR*, abs/1111.6191, 2011.

[9] Toon Calders, Nele Dexters, and Bart Goethals. Mining frequent itemsets in a stream. In *ICDM*, pages 83–92, 2007.

[10] Toon Calders, Nele Dexters, and Bart Goethals. Mining frequent items in a stream using flexible windows. *Intell. Data Anal.*, 12(3):293–304, 2008.

[11] Toon Calders and Bart Goethals. Mining all non-derivable frequent itemsets. In *PKDD*, volume 2431 of *Lecture Notes in Computer Science*, pages 74–85. Springer, 2002.

[12] Nuno Castro and Paulo J. Azevedo. Multiresolution motif discovery in time series. In *SDM*, pages 665–676. SIAM, 2010.

[13] Nuno Castro and Paulo J. Azevedo. Time series motifs statistical significance. In *SDM*, pages 687–698. SIAM / Omnipress, 2011.

[14] Deepayan Chakrabarti, Spiros Papadimitriou, Dharmendra S. Modha, and Christos Faloutsos. Fully automatic cross-associations. In Won Kim, Ron Kohavi, Johannes Gehrke, and William DuMouchel, editors, *KDD*, pages 79–88. ACM, 2004.

[15] Hong Cheng, Xifeng Yan, Jiawei Han, and Philip S. Yu. Direct discriminative pattern mining for effective classification. In *Proceedings of the 2008 IEEE 24th International Conference on Data Engineering*, ICDE '08, pages 169–178, Washington, DC, USA, 2008. IEEE Computer Society.

[16] Bill Yuan chi Chiu, Eamonn J. Keogh, and Stefano Lonardi. Probabilistic discovery of time series motifs. In *KDD*, pages 493–498. ACM, 2003.

[17] R. Cilibrasi and P.M.B. Vitanyi. Clustering by compression. *IEEE Transactions on Information Theory*, 51(4):1523–1545, April 2005.

[18] William Jay Conover. *Practical nonparametric statistics*. Wiley, New York [u.a.], wiley international ed edition, 1971.

[19] Thomas H. Cormen. *Introduction to algorithms*. The MIT Press, Cambridge, Masachusetts; London, 2009.

[20] Thomas M. Cover and Joy A. Thomas. *Elements of information theory (2. ed.)*. Wiley, 2006.

[21] Erik D. Demaine, Alejandro López-Ortiz, and J. Ian Munro. Frequency estimation of internet packet streams with limited space. In *ESA '02: Proceedings of the 10th Annual European Symposium on Algorithms*, pages 348–360, London, UK, 2002. Springer-Verlag.

[22] Christos Faloutsos and Vasileios Megalooikonomou. On data mining, compression, and kolmogorov complexity. *Data Min. Knowl. Discov.*, 15(1):3–20, 2007.

[23] Basura Fernando, Élisa Fromont, and Tinne Tuytelaars. Effective use of frequent itemset mining for image classification. In *ECCV (1)*, pages 214–227, 2012.

[24] Pedro Gabriel Ferreira, Paulo J. Azevedo, Candida G. Silva, and Rui M. M. Brito. Mining approximate motifs in time series. In *Discovery Science*, volume 4265 of *Lecture Notes in Computer Science*, pages 89–101. Springer, 2006.

[25] Philippe Flajolet, Danièle Gardy, and Loÿs Thimonier. Birthday paradox, coupon collectors, caching algorithms and self-organizing search. *Discrete Appl. Math.*, 39(3):207–229, 1992.

[26] Dmitriy Fradkin and Fabian Moerchen. Margin-closed frequent sequential pattern mining. In *Proceedings of the ACM SIGKDD Workshop on Useful Patterns*, UP '10, pages 45–54, New York, NY, USA, 2010. ACM.

[27] Floris Geerts, Bart Goethals, and Taneli Mielikäinen. Tiling databases. In *Discovery Science*, pages 278–289, 2004.

[28] Robertson E. Giannella C., Han J. and Liu C. Mining frequent itemsets over arbitrary time intervals in data streams. In *Technical Report TR587 at Indiana University, Bloomington*, page 37 pages, 2003.

[29] Aristides Gionis, Heikki Mannila, Taneli Mielikäinen, and Panayiotis Tsaparas. Assessing data mining results via swap randomization. *TKDD*, 1(3), 2007.

[30] Peter D. Grünwald. *The Minimum Description Length Principle (Adaptive Computation and Machine Learning)*. The MIT Press, 2007.

[31] Robert Gwadera, Mikhail J. Atallah, and Wojciech Szpankowski. Markov models for identification of significant episodes. In *SDM*, 2005.

[32] Jiawei Han, Hong Cheng, Dong Xin, and Xifeng Yan. Frequent pattern mining: current status and future directions. *Data Min. Knowl. Discov.*, 15(1):55–86, August 2007.

[33] Sami Hanhijärvi, Gemma Garriga, and Kai Puolamäki. Randomization techniques for graphs. 2009.

[34] Dmitriy Fradkin Hoang Thanh Lam, Fabian Mörchen and Toon Calders. Mining compressing sequential patterns. *Statitiscal Analysis and Data Mining*, 28(3):345–358, 2013.

[35] Lawrence B. Holder, Diane J. Cook, and Surnjani Djoko. Substucture discovery in the subdue system. In *KDD Workshop*, pages 169–180. AAAI Press, 1994.

[36] A Hyvärinen and E Oja. Independent component analysis: algorithms and applications. *Neural Netw*, 13(4-5):411–430, May-Jun 2000.

[37] Eamonn J. Keogh, Stefano Lonardi, Chotirat Ann Ratanamahatana, Li Wei, Sang-Hee Lee, and John Handley. Compression-based data mining of sequential data. *Data Min. Knowl. Discov.*, 14(1):99–129, 2007.

[38] Julia Kiseleva, Hoang Thanh Lam, Mykola Pechenizkiy, and Toon Calders. Discovering temporal hidden contexts in web sessions for user trail prediction. In *Proceedings of the 22nd international conference on World Wide Web companion*, WWW '13 Companion, pages 1067–1074, Republic and Canton of Geneva, Switzerland, 2013. International World Wide Web Conferences Steering Committee.

[39] Arno J. Knobbe, Hendrik Blockeel, Arne Koopman, Toon Calders, Bas Obladen, Carlos Bosma, Hessel Galenkamp, Eddy Koenders, and Joost N. Kok. Infrawatch: Data management of large systems for monitoring infrastructural performance. In Paul R. Cohen, Niall M. Adams, and Michael R. Berthold, editors, *IDA*, volume 6065 of *Lecture Notes in Computer Science*, pages 91–102. Springer, 2010.

[40] Hoang Thanh Lam, Fabian Moerchen, Dmitriy Fradkin, and Toon Calders. Mining compressing sequential patterns. In *SDM*, pages 319–330. SIAM / Omnipress, 2012.

[41] L. K. Lee and H. F. Ting. A simpler and more efficient deterministic scheme for finding frequent items over sliding windows. In *PODS '06: Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 290–297, New York, NY, USA, 2006. ACM.

[42] Zhenhui Li, Jiawei Han, Bolin Ding, and Roland Kays. Mining periodic behaviors of object movements for animal and biological sustainability studies. *Data Min. Knowl. Discov.*, 24(2):355–386, 2012.

[43] Jessica Lin, Eamonn Keogh, Stefano Lonardi, and Pranav Patel. Finding motifs in time series. In *Proceedings of the Second Workshop on Temporal Data Mining*, Edmonton, Alberta, Canada, July 2002.

[44] Haibing Lu, Jaideep Vaidya, and Vijayalakshmi Atluri. Optimal boolean matrix decomposition: Application to role engineering. In *ICDE*, pages 297–306, 2008.

[45] Michael Mampaey, Nikolaj Tatti, and Jilles Vreeken. Tell me what i need to know: succinctly summarizing data with itemsets. In *KDD*, pages 573–581. ACM, 2011.

[46] Michael Mampaey and Jilles Vreeken. Summarizing categorical data by clustering attributes. *Data Min. Knowl. Discov.*, 26(1):130–173, 2013.

[47] Heikki Mannila and Dmitry Rusakov. Decomposition of event sequences into independent components. In *Proceedings of the First SIAM Conference on Data Mining*, pages 1–17. SIAM, 2001.

[48] Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. Efficient computation of frequent and top-k elements in data streams. In *ICDT*, volume 3363 of *Lecture Notes in Computer Science*, pages 398–412. Springer, 2005.

[49] Pauli Miettinen, Taneli Mielikäinen, Aristides Gionis, Gautam Das, and Heikki Mannila. The discrete basis problem. *IEEE Transactions on Knowledge and Data Engineering*, 20(10):1348–1362, October 2008.

[50] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon. Network motifs: Simple building blocks of complex networks. *Science*, 298(5594):824–827, 2002.

[51] Fabian Mörchen. Unsupervised pattern mining from symbolic temporal data. *SIGKDD Explor. Newsl.*, 9(1):41–55, June 2007.

[52] Fabian Mörchen and Dmitriy Fradkin. Robust mining of time intervals with semi-interval partial order patterns. In *SDM*, pages 315–326. SIAM, 2010.

[53] Behzad Mortazavi-Asl, Jianyong Wang, Helen Pinto, Qiming Chen, and Mei-Chun Hsu. Mining sequential patterns by pattern-growth: The prefixspan approach. *IEEE Trans. on Knowl. and Data Eng.*, 16(11):1424–1440, 2004. Member-Jian Pei and Senior Member-Jiawei Han and Member-Umeshwar Dayal.

[54] Rajeev Motwani and Prabhakar Raghavan. *Randomized algorithms*. Cambridge University Press, USA, 1995.

[55] Rajeev Motwani and Prabhakar Raghavan. Randomized algorithms. *ACM Comput. Surv.*, 28(1):33–37, 1996.

[56] Abdullah Mueen and Eamonn J. Keogh. Online discovery and maintenance of time series motifs. In *KDD*, pages 1089–1098. ACM, 2010.

[57] Abdullah Mueen, Eamonn J. Keogh, and Nima Bigdely Shamlo. Finding time series motifs in disk-resident data. In *ICDM*, pages 367–376. IEEE Computer Society, 2009.

[58] Abdullah Mueen, Eamonn J. Keogh, Qiang Zhu, Sydney Cash, and M. Brandon Westover. Exact discovery of time series motifs. In *SDM*, pages 473–484, 2009.

[59] Amy N. Myers and Herbert S. Wilf. Some new aspects of the coupon collector's problem. *SIAM J. Discret. Math.*, 17(1):1–17, 2004.

[60] Nicolas Pasquier, Yves Bastide, Rafik Taouil, and Lotfi Lakhal. Discovering frequent closed itemsets for association rules. ICDT, pages 398–416. Springer-Verlag, 1999.

[61] Nicolas Pasquier, Yves Bastide, Rafik Taouil, and Lotfi Lakhal. Discovering frequent closed itemsets for association rules. In Catriel Beeri and Peter Buneman, editors, *ICDT*, volume 1540 of *Lecture Notes in Computer Science*, pages 398–416. Springer, 1999.

[62] Pranav Patel, Eamonn J. Keogh, Jessica Lin 0001, and Stefano Lonardi. Mining motifs in massive time series databases. In *ICDM*, pages 370–377. IEEE Computer Society, 2002.

[63] Jian Pei, Guozhu Dong, Wei Zou, and Jiawei Han. On computing condensed frequent pattern bases. In *ICDM*, pages 378–385. IEEE Computer Society, 2002.

[64] Kira Radinsky and Eric Horvitz. Mining the web to predict future events. In Stefano Leonardi, Alessandro Panconesi, Paolo Ferragina, and Aristides Gionis, editors, *WSDM*, pages 255–264. ACM, 2013.

[65] W.M. Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical association*, 66(336):846–850, 1971.

[66] Koen Smets and Jilles Vreeken. The odd one out: Identifying and characterising anomalies. In *SDM*, pages 804–815, 2011.

[67] Koen Smets and Jilles Vreeken. Slim: Directly mining descriptive patterns. In *SDM*, pages 236–247. SIAM / Omnipress, 2012.

[68] James A. Storer and Thomas G. Szymanski. Data compression via textual substitution. *J. ACM*, 29(4):928–951, October 1982.

[69] Nikolaj Tatti and Jilles Vreeken. Comparing apples and oranges: measuring differences between exploratory data mining results. *Data Min. Knowl. Discov.*, 25(2):173–207, 2012.

[70] Nikolaj Tatti and Jilles Vreeken. Discovering descriptive tile trees by mining optimal geometric subtiles. In *ECML/PKDD (1)*, pages 9–24, 2012.

[71] Nikolaj Tatti and Jilles Vreeken. The long and the short of it: summarising event sequences with serial episodes. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '12, pages 462–470, New York, NY, USA, 2012. ACM.

[72] Jaideep Vaidya, Vijayalakshmi Atluri, and Qi Guo. The role mining problem: A formal perspective. *ACM Trans. Inf. Syst. Secur.*, 13(3), 2010.

[73] Matthijs van Leeuwen and Arno Siebes. Streamkrimp: Detecting change in data streams. In Walter Daelemans, Bart Goethals, and Katharina Morik, editors, *ECML/PKDD (1)*, volume 5211 of *Lecture Notes in Computer Science*, pages 672–687. Springer, 2008.

[74] Matthijs van Leeuwen, Jilles Vreeken, and Arno Siebes. Identifying the components. *Data Min. Knowl. Discov.*, 19(2):176–193, 2009.

[75] Jilles Vreeken, Matthijs van Leeuwen, and Arno Siebes. Krimp: mining itemsets that compress. *Data Min. Knowl. Discov.*, 23(1):169–214, 2011.

[76] Jilles Vreeken, Matthijs van Leeuwen, and Arno Siebes. Krimp: mining itemsets that compress. *Data Min. Knowl. Discov.*, 23(1):169–214, 2011.

[77] Jianyong Wang and Jiawei Han. Bide: Efficient mining of frequent closed sequences. In Z. Meral Özsoyoglu and Stanley B. Zdonik, editors, *ICDE*, pages 79–90. IEEE Computer Society, 2004.

[78] Ian H. Witten, Alistair Moffat, and Timothy C. Bell. *Managing Gigabytes : Compressing and Indexing Documents and Images*. Morgan Kaufmann, San Francisco, CA, 2 edition, 1999.

[79] Dong Xin, Jiawei Han, Xifeng Yan, and Hong Cheng. Mining compressed frequent-pattern sets. In *VLDB '05: Proceedings of the 31st international conference on Very large data bases*, pages 709–720. VLDB Endowment, 2005.

[80] Dragomir Yankov, Eamonn J. Keogh, Jose Medina, Bill Chiu, and Victor B. Zordan. Detecting time series motifs under uniform scaling. In *KDD*, pages 844–853. ACM, 2007.

SIKS Dissertations series

1998

1998-1 Johan van den Akker (CWI) DEGAS - An Active, Temporal Database of Autonomous Objects

1998-2 Floris Wiesman (UM) Information Retrieval by Graphically Browsing Meta-Information

1998-3 Ans Steuten (TUD) A Contribution to the Linguistic Analysis of Business Conversations within the Language/Action Perspective

1998-4 Dennis Breuker (UM) Memory versus Search in Games

1998-5 E.W.Oskamp (RUL) Computerondersteuning bij Straftoemeting

1999

1999-1 Mark Sloof (VU) Physiology of Quality Change Modelling; Automated modelling of Quality Change of Agricultural Products

1999-2 Rob Potharst (EUR) Classification using decision trees and neural nets

1999-3 Don Beal (UM) The Nature of Minimax Search

1999-4 Jacques Penders (UM) The practical Art of Moving Physical Objects

1999-5 Aldo de Moor (KUB) Empowering Communities: A Method for the Legitimate User-Driven Specification of Network Information Systems

1999-6 Niek J.E. Wijngaards (VU) Re-design of compositional systems

1999-7 David Spelt (UT) Verification support for object database design

1999-8 Jacques H.J. Lenting (UM) Informed Gambling: Conception and Analysis of a Multi-Agent Mechanism for Discrete Reallocation.

2000

2000-1 Frank Niessink (VU) Perspectives on Improving Software Maintenance

2000-2 Koen Holtman (TUE) Prototyping of CMS Storage Management

2000-3 Carolien M.T. Metselaar (UVA) Sociaal-organisatorische gevolgen van kennistechnologie; een procesbenadering en actorperspectief.

2000-4 Geert de Haan (VU) ETAG, A Formal Model of Competence Knowledge for User Interface Design

2000-5 Ruud van der Pol (UM) Knowledge-based Query Formulation in Information Retrieval.

2000-6 Rogier van Eijk (UU) Programming Languages for Agent Communication

2000-7 Niels Peek (UU) Decision-theoretic Planning of Clinical Patient Management

2000-8 Veerle Coup (EUR) Sensitivity Analysis of Decision-Theoretic Networks

2000-9 Florian Waas (CWI) Principles of Probabilistic Query Optimization

2000-10 Niels Nes (CWI) Image Database Management System Design Considerations, Algorithms and Architecture

2000-11 Jonas Karlsson (CWI) Scalable Distributed Data Structures for Database Management

2001

2001-1 Silja Renooij (UU) Qualitative Approaches to Quantifying Probabilistic Networks

2001-2 Koen Hindriks (UU) Agent Programming Languages: Programming with Mental Models

2001-3 Maarten van Someren (UvA) Learning as problem solving

2001-4 Evgueni Smirnov (UM) Conjunctive and Disjunctive Version Spaces with Instance-Based Boundary Sets

2001-5 Jacco van Ossenbruggen (VU) Processing Structured Hypermedia: A Matter of Style

2001-6 Martijn van Welie (VU) Task-based User Interface Design

2001-7 Bastiaan Schonhage (VU) Diva: Architectural Perspectives on Information Visualization

2001-8 Pascal van Eck (VU) A Compositional Semantic Structure for Multi-Agent Systems Dynamics.

2001-9 Pieter Jan 't Hoen (RUL) Towards Distributed Development of Large Object-Oriented Models, Views of Packages as Classes

2001-10 Maarten Sierhuis (UvA) Modeling and Simulating Work Practice BRAHMS: a multiagent modeling and simulation language for work practice analysis and design

2001-11 Tom M. van Engers (VUA) Knowledge Management: The Role of Mental Models in Business Systems Design

2002

2002-01 Nico Lassing (VU) Architecture-Level Modifiability Analysis

2002-02 Roelof van Zwol (UT) Modelling and searching web-based document collections

2002-03 Henk Ernst Blok (UT) Database Optimization Aspects for Information Retrieval

2002-04 Juan Roberto Castelo Valdueza (UU) The Discrete Acyclic Digraph Markov Model in Data Mining

2002-05 Radu Serban (VU) The Private Cyberspace Modeling Electronic Environments inhabited by Privacy-concerned Agents

2002-06 Laurens Mommers (UL) Applied legal epistemology; Building a knowledge-based ontology of the legal domain

2002-07 Peter Boncz (CWI) Monet: A Next-Generation DBMS Kernel For Query-Intensive Applications

2002-08 Jaap Gordijn (VU) Value Based Requirements Engineering: Exploring Innovative E-Commerce Ideas

2002-09 Willem-Jan van den Heuvel(KUB) Integrating Modern Business Applications with Objectified Legacy Systems

2002-10 Brian Sheppard (UM) Towards Perfect Play of Scrabble

2002-11 Wouter C.A. Wijngaards (VU) Agent Based Modelling of Dynamics: Biological and Organisational Applications

2002-12 Albrecht Schmidt (Uva) Processing XML in Database Systems

2002-13 Hongjing Wu (TUE) A Reference Architecture for Adaptive Hypermedia Applications

2002-14 Wieke de Vries (UU) Agent Interaction: Abstract Approaches to Modelling, Programming and Verifying Multi-Agent Systems

2002-15 Rik Eshuis (UT) Semantics and Verification of UML Activity Diagrams for Workflow Modelling

2002-16 Pieter van Langen (VU) The Anatomy of Design: Foundations, Models and Applications

2002-17 Stefan Manegold (UVA) Understanding, Modeling, and Improving Main-Memory Database Performance

2003

2003-01 Heiner Stuckenschmidt (VU) Ontology-Based Information Sharing in Weakly Structured Environments

2003-02 Jan Broersen (VU) Modal Action Logics for Reasoning About Reactive Systems

2003-03 Martijn Schuemie (TUD) Human-Computer Interaction and Presence in Virtual Reality Exposure Therapy

2003-04 Milan Petkovic (UT) Content-Based Video Retrieval Supported by Database Technology

2003-05 Jos Lehmann (UVA) Causation in Artificial Intelligence and Law - A modelling approach

2003-06 Boris van Schooten (UT) Development and specification of virtual environments

2003-07 Machiel Jansen (UvA) Formal Explorations of Knowledge Intensive Tasks

2003-08 Yongping Ran (UM) Repair Based Scheduling

2003-09 Rens Kortmann (UM) The resolution of visually guided behaviour

2003-10 Andreas Lincke (UvT) Electronic Business Negotiation: Some experimental studies on the interaction between medium, innovation context and culture

2003-11 Simon Keizer (UT) Reasoning under Uncertainty in Natural Language Dialogue using Bayesian Networks

2003-12 Roeland Ordelman (UT) Dutch speech recognition in multimedia information retrieval

2003-13 Jeroen Donkers (UM) Nosce Hostem - Searching with Opponent Models

2003-14 Stijn Hoppenbrouwers (KUN) Freezing Language: Conceptualisation Processes across ICT-Supported Organisations

2003-15 Mathijs de Weerdt (TUD) Plan Merging in Multi-Agent Systems

2003-16 Menzo Windhouwer (CWI) Feature Grammar Systems - Incremental Maintenance of Indexes to Digital Media Warehouses

2003-17 David Jansen (UT) Extensions of Statecharts with Probability, Time, and Stochastic Timing

2003-18 Levente Kocsis (UM) Learning Search Decisions

2004

2004-01 Virginia Dignum (UU) A Model for Organizational Interaction: Based on Agents, Founded in Logic

2004-02 Lai Xu (UvT) Monitoring Multi-party Contracts for E-business

2004-03 Perry Groot (VU) A Theoretical and Empirical Analysis of Approximation in Symbolic Problem Solving

2004-04 Chris van Aart (UVA) Organizational Principles for Multi-Agent Architectures

2004-05 Viara Popova (EUR) Knowledge discovery and monotonicity

2004-06 Bart-Jan Hommes (TUD) The Evaluation of Business Process Modeling Techniques

2004-07 Elise Boltjes (UM) Voorbeeldig onderwijs; voorbeeldgestuurd onderwijs, een opstap naar abstract denken, vooral voor meisjes

2004-08 Joop Verbeek(UM) Politie en de Nieuwe Internationale Informatiemarkt, Grensregionale politile gegevensuitwisseling en digitale expertise

2004-09 Martin Caminada (VU) For the Sake of the Argument; explorations into argument-based reasoning

2004-10 Suzanne Kabel (UVA) Knowledge-rich indexing of learning-objects

2004-11 Michel Klein (VU) Change Management for Distributed Ontologies

2004-12 The Duy Bui (UT) Creating emotions and facial expressions for embodied agents

2004-13 Wojciech Jamroga (UT) Using Multiple Models of Reality: On Agents who Know how to Play

2004-14 Paul Harrenstein (UU) Logic in Conflict. Logical Explorations in Strategic Equilibrium

2004-15 Arno Knobbe (UU) Multi-Relational Data Mining

2004-16 Federico Divina (VU) Hybrid Genetic Relational Search for Inductive Learning

2004-17 Mark Winands (UM) Informed Search in Complex Games

2004-18 Vania Bessa Machado (UvA) Supporting the Construction of Qualitative Knowledge Models

2004-19 Thijs Westerveld (UT) Using generative probabilistic models for multimedia retrieval

2004-20 Madelon Evers (Nyenrode) Learning from Design: facilitating multidisciplinary design teams

2005

2005-01 Floor Verdenius (UVA) Methodological Aspects of Designing Induction-Based Applications

2005-02 Erik van der Werf (UM)) AI techniques for the game of Go

2005-03 Franc Grootjen (RUN) A Pragmatic Approach to the Conceptualisation of Language

2005-04 Nirvana Meratnia (UT) Towards Database Support for Moving Object data

2005-05 Gabriel Infante-Lopez (UVA) Two-Level Probabilistic Grammars for Natural Language Parsing

2005-06 Pieter Spronck (UM) Adaptive Game AI

2005-07 Flavius Frasincar (TUE) Hypermedia Presentation Generation for Semantic Web Information Systems

2005-08 Richard Vdovjak (TUE) A Model-driven Approach for Building Distributed Ontology-based Web Applications

2005-09 Jeen Broekstra (VU) Storage, Querying and Inferencing for Semantic Web Languages

2005-10 Anders Bouwer (UVA) Explaining Behaviour: Using Qualitative Simulation in Interactive Learning Environments

2005-11 Elth Ogston (VU) Agent Based Matchmaking and Clustering - A Decentralized Approach to Search

2005-12 Csaba Boer (EUR) Distributed Simulation in Industry

2005-13 Fred Hamburg (UL) Een Computermodel voor het Ondersteunen van Euthanasiebeslissingen

2005-14 Borys Omelayenko (VU) Web-Service configuration on the Semantic Web; Exploring how semantics meets pragmatics

2005-15 Tibor Bosse (VU) Analysis of the Dynamics of Cognitive Processes

2005-16 Joris Graaumans (UU) Usability of XML Query Languages

2005-17 Boris Shishkov (TUD) Software Specification Based on Re-usable Business Components

2005-18 Danielle Sent (UU) Test-selection strategies for probabilistic networks

2005-19 Michel van Dartel (UM) Situated Representation

2005-20 Cristina Coteanu (UL) Cyber Consumer Law, State of the Art and Perspectives

2005-21 Wijnand Derks (UT) Improving Concurrency and Recovery in Database Systems by Exploiting Application Semantics

2006

2006-01 Samuil Angelov (TUE) Foundations of B2B Electronic Contracting

2006-02 Cristina Chisalita (VU) Contextual issues in the design and use of information technology in organizations

2006-03 Noor Christoph (UVA) The role of metacognitive skills in learning to solve problems

2006-04 Marta Sabou (VU) Building Web Service Ontologies

2006-05 Cees Pierik (UU) Validation Techniques for Object-Oriented Proof Outlines

2006-06 Ziv Baida (VU) Software-aided Service Bundling - Intelligent Methods and Tools for Graphical Service Modeling

2006-07 Marko Smiljanic (UT) XML schema matching – balancing efficiency and effectiveness by means of clustering

2006-08 Eelco Herder (UT) Forward, Back and Home Again - Analyzing User Behavior on the Web

2006-09 Mohamed Wahdan (UM) Automatic Formulation of the Auditor's Opinion

2006-10 Ronny Siebes (VU) Semantic Routing in Peer-to-Peer Systems

2006-11 Joeri van Ruth (UT) Flattening Queries over Nested Data Types

2006-12 Bert Bongers (VU) Interactivation - Towards an e-cology of people, our technological environment, and the arts

2006-13 Henk-Jan Lebbink (UU) Dialogue and Decision Games for Information Exchanging Agents

2006-14 Johan Hoorn (VU) Software Requirements: Update, Upgrade, Redesign - towards a Theory of Requirements Change

2006-15 Rainer Malik (UU) CONAN: Text Mining in the Biomedical Domain

2006-16 Carsten Riggelsen (UU) Approximation Methods for Efficient Learning of Bayesian Networks

2006-17 Stacey Nagata (UU) User Assistance for Multitasking with Interruptions on a Mobile Device

2006-18 Valentin Zhizhkun (UVA) Graph transformation for Natural Language Processing

2006-19 Birna van Riemsdijk (UU) Cognitive Agent Programming: A Semantic Approach

2006-20 Marina Velikova (UvT) Monotone models for prediction in data mining

2006-21 Bas van Gils (RUN) Aptness on the Web

2006-22 Paul de Vrieze (RUN) Fundaments of Adaptive Personalisation

2006-23 Ion Juvina (UU) Development of Cognitive Model for Navigating on the Web

2006-24 Laura Hollink (VU) Semantic Annotation for Retrieval of Visual Resources

2006-25 Madalina Drugan (UU) Conditional log-likelihood MDL and Evolutionary MCMC

2006-26 Vojkan Mihajlovic (UT) Score Region Algebra: A Flexible Framework for Structured Information Retrieval

2006-27 Stefano Bocconi (CWI) Vox Populi: generating video documentaries from semantically annotated media repositories

2006-28 Borkur Sigurbjornsson (UVA) Focused Information Access using XML Element Retrieval

2007

2007-01 Kees Leune (UvT) Access Control and Service-Oriented Architectures

2007-02 Wouter Teepe (RUG) Reconciling Information Exchange and Confidentiality: A Formal Approach

2007-03 Peter Mika (VU) Social Networks and the Semantic Web

2007-04 Jurriaan van Diggelen (UU) Achieving Semantic Interoperability in Multi-agent Systems: a dialogue-based approach

2007-05 Bart Schermer (UL) Software Agents, Surveillance, and the Right to Privacy: a Legislative Framework for Agent-enabled Surveillance

2007-06 Gilad Mishne (UVA) Applied Text Analytics for Blogs

2007-07 Natasa Jovanovic' (UT) To Whom It May Concern - Addressee Identification in Face-to-Face Meetings

2007-08 Mark Hoogendoorn (VU) Modeling of Change in Multi-Agent Organizations

2007-09 David Mobach (VU) Agent-Based Mediated Service Negotiation

2007-10 Huib Aldewereld (UU) Autonomy vs. Conformity: an Institutional Perspective on Norms and Protocols

2007-11 Natalia Stash (TUE) Incorporating Cognitive/Learning Styles in a General-Purpose Adaptive Hypermedia System

2007-12 Marcel van Gerven (RUN) Bayesian Networks for Clinical Decision Support: A Rational Approach to Dynamic Decision-Making under Uncertainty

2007-13 Rutger Rienks (UT) Meetings in Smart Environments; Implications of Progressing Technology

2007-14 Niek Bergboer (UM) Context-Based Image Analysis

2007-15 Joyca Lacroix (UM) NIM: a Situated Computational Memory Model

2007-16 Davide Grossi (UU) Designing Invisible Handcuffs. Formal investigations in Institutions and Organizations for Multi-agent Systems

2007-17 Theodore Charitos (UU) Reasoning with Dynamic Networks in Practice

2007-18 Bart Orriens (UvT) On the development an management of adaptive business collaborations

2007-19 David Levy (UM) Intimate relationships with artificial partners

2007-20 Slinger Jansen (UU) Customer Configuration Updating in a Software Supply Network

2007-21 Karianne Vermaas (UU) Fast diffusion and broadening use: A research on residential adoption and usage of broadband internet in the Netherlands between 2001 and 2005

2007-22 Zlatko Zlatev (UT) Goal-oriented design of value and process models from patterns

2007-23 Peter Barna (TUE) Specification of Application Logic in Web Information Systems

2007-24 Georgina Ramírez Camps (CWI) Structural Features in XML Retrieval

2007-25 Joost Schalken (VU) Empirical Investigations in Software Process Improvement

2008

2008-01 Katalin Boer-Sorbán (EUR) Agent-Based Simulation of Financial Markets: A modular, continuous-time approach

2008-02 Alexei Sharpanskykh (VU) On Computer-Aided Methods for Modeling and Analysis of Organizations

2008-03 Vera Hollink (UVA) Optimizing hierarchical menus: a usage-based approach

2008-04 Ander de Keijzer (UT) Management of Uncertain Data - towards unattended integration

2008-05 Bela Mutschler (UT) Modeling and simulating causal dependencies on process-aware information systems from a cost perspective

2008-06 Arjen Hommersom (RUN) On the Application of Formal Methods to Clinical Guidelines, an Artificial Intelligence Perspective

2008-07 Peter van Rosmalen (OU) Supporting the tutor in the design and support of adaptive e-learning

2008-08 Janneke Bolt (UU) Bayesian Networks: Aspects of Approximate Inference

2008-09 Christof van Nimwegen (UU) The paradox of the guided user: assistance can be counter-effective

2008-10 Wauter Bosma (UT) Discourse oriented summarization

2008-11 Vera Kartseva (VU) Designing Controls for Network Organizations: A Value-Based Approach

2008-12 Jozsef Farkas (RUN) A Semiotically Oriented Cognitive Model of Knowledge Representation

2008-13 Caterina Carraciolo (UVA) Topic Driven Access to Scientific Handbooks

2008-14 Arthur van Bunningen (UT) Context-Aware Querying; Better Answers with Less Effort

2008-15 Martijn van Otterlo (UT) The Logic of Adaptive Behavior: Knowledge Representation and Algorithms for the Markov Decision Process Framework in First-Order Domains.

2008-16 Henriette van Vugt (VU) Embodied agents from a user's perspective

2008-17 Martin Op 't Land (TUD) Applying Architecture and Ontology to the Splitting and Allying of Enterprises

2008-18 Guido de Croon (UM) Adaptive Active Vision

2008-19 Henning Rode (UT) From Document to Entity Retrieval: Improving Precision and Performance of Focused Text Search

2008-20 Rex Arendsen (UVA) Geen bericht, goed bericht. Een onderzoek naar de effecten van de introductie van elektronisch berichtenverkeer met de overheid op de administratieve lasten van bedrijven.

2008-21 Krisztian Balog (UVA) People Search in the Enterprise

2008-22 Henk Koning (UU) Communication of IT-Architecture

2008-23 Stefan Visscher (UU) Bayesian network models for the management of ventilator-associated pneumonia

2008-24 Zharko Aleksovski (VU) Using background knowledge in ontology matching

2008-25 Geert Jonker (UU) Efficient and Equitable Exchange in Air Traffic Management Plan Repair using Spender-signed Currency

2008-26 Marijn Huijbregts (UT) Segmentation, Diarization and Speech Transcription: Surprise Data Unraveled

2008-27 Hubert Vogten (OU) Design and Implementation Strategies for IMS Learning Design

2008-28 Ildiko Flesch (RUN) On the Use of Independence Relations in Bayesian Networks

2008-29 Dennis Reidsma (UT) Annotations and Subjective Machines - Of Annotators, Embodied Agents, Users, and Other Humans

2008-30 Wouter van Atteveldt (VU) Semantic Network Analysis: Techniques for Extracting, Representing and Querying Media Content

2008-31 Loes Braun (UM) Pro-Active Medical Information Retrieval

2008-32 Trung H. Bui (UT) Toward Affective Dialogue Management using Partially Observable Markov Decision Processes

2008-33 Frank Terpstra (UVA) Scientific Workflow Design; theoretical and practical issues

2008-34 Jeroen de Knijf (UU) Studies in Frequent Tree Mining

2008-35 Ben Torben Nielsen (UvT) Dendritic morphologies: function shapes structure

2009

2009-01 Rasa Jurgelenaite (RUN) Symmetric Causal Independence Models

2009-02 Willem Robert van Hage (VU) Evaluating Ontology-Alignment Techniques

2009-03 Hans Stol (UvT) A Framework for Evidence-based Policy Making Using IT

2009-04 Josephine Nabukenya (RUN) Improving the Quality of Organisational Policy Making using Collaboration Engineering

2009-05 Sietse Overbeek (RUN) Bridging Supply and Demand for Knowledge Intensive Tasks - Based on Knowledge, Cognition, and Quality

2009-06 Muhammad Subianto (UU) Understanding Classification

2009-07 Ronald Poppe (UT) Discriminative Vision-Based Recovery and Recognition of Human Motion

2009-08 Volker Nannen (VU) Evolutionary Agent-Based Policy Analysis in Dynamic Environments

2009-09 Benjamin Kanagwa (RUN) Design, Discovery and Construction of Service-oriented Systems

2009-10 Jan Wielemaker (UVA) Logic programming for knowledge-intensive interactive applications

2009-11 Alexander Boer (UVA) Legal Theory, Sources of Law and the Semantic Web

2009-12 Peter Massuthe (TUE, Humboldt-Universitaet zu Berlin) Operating Guidelines for Services

2009-13 Steven de Jong (UM) Fairness in Multi-Agent Systems

2009-14 Maksym Korotkiy (VU) From ontology-enabled services to service-enabled ontologies (making ontologies work in e-science with ONTO-SOA)

2009-15 Rinke Hoekstra (UVA) Ontology Representation - Design Patterns and Ontologies that Make Sense

2009-16 Fritz Reul (UvT) New Architectures in Computer Chess

2009-17 Laurens van der Maaten (UvT) Feature Extraction from Visual Data

2009-18 Fabian Groffen (CWI) Armada, An Evolving Database System

2009-19 Valentin Robu (CWI) Modeling Preferences, Strategic Reasoning and Collaboration in Agent-Mediated Electronic Markets

2009-20 Bob van der Vecht (UU) Adjustable Autonomy: Controling Influences on Decision Making

2009-21 Stijn Vanderlooy (UM) Ranking and Reliable Classification

2009-22 Pavel Serdyukov (UT) Search For Expertise: Going beyond direct evidence

2009-23 Peter Hofgesang (VU) Modelling Web Usage in a Changing Environment

2009-24 Annerieke Heuvelink (VUA) Cognitive Models for Training Simulations

2009-25 Alex van Ballegooij (CWI) "RAM: Array Database Management through Relational Mapping"

2009-26 Fernando Koch (UU) An Agent-Based Model for the Development of Intelligent Mobile Services

2009-27 Christian Glahn (OU) Contextual Support of social Engagement and Reflection on the Web

2009-28 Sander Evers (UT) Sensor Data Management with Probabilistic Models

2009-29 Stanislav Pokraev (UT) Model-Driven Semantic Integration of Service-Oriented Applications

2009-30 Marcin Zukowski (CWI) Balancing vectorized query execution with bandwidth-optimized storage

2009-31 Sofiya Katrenko (UVA) A Closer Look at Learning Relations from Text

2009-32 Rik Farenhorst (VU) and Remco de Boer (VU) Architectural Knowledge Management: Supporting Architects and Auditors

2009-33 Khiet Truong (UT) How Does Real Affect Affect Affect Recognition In Speech?

2009-34 Inge van de Weerd (UU) Advancing in Software Product Management: An Incremental Method Engineering Approach

2009-35 Wouter Koelewijn (UL) Privacy en Politiegegevens; Over geautomatiseerde normatieve informatie-uitwisseling

2009-36 Marco Kalz (OUN) Placement Support for Learners in Learning Networks

2009-37 Hendrik Drachsler (OUN) Navigation Support for Learners in Informal Learning Networks

2009-38 Riina Vuorikari (OU) Tags and self-organisation: a metadata ecology for learning resources in a multilingual context

2009-39 Christian Stahl (TUE, Humboldt-Universitaet zu Berlin) Service Substitution – A Behavioral Approach Based on Petri Nets

2009-40 Stephan Raaijmakers (UvT) Multinomial Language Learning: Investigations into the Geometry of Language

2009-41 Igor Berezhnyy (UvT) Digital Analysis of Paintings

2009-42 Toine Bogers (UvT) Recommender Systems for Social Bookmarking

2009-43 Virginia Nunes Leal Franqueira (UT) Finding Multi-step Attacks in Computer Networks using Heuristic Search and Mobile Ambients

2009-44 Roberto Santana Tapia (UT) Assessing Business-IT Alignment in Networked Organizations

2009-45 Jilles Vreeken (UU) Making Pattern Mining Useful

2009-46 Loredana Afanasiev (UvA) Querying XML: Benchmarks and Recursion

2010

2010-01 Matthijs van Leeuwen (UU) Patterns that Matter

2010-02 Ingo Wassink (UT) Work flows in Life Science

2010-03 Joost Geurts (CWI) A Document Engineering Model and Processing Framework for Multimedia documents

2010-04 Olga Kulyk (UT) Do You Know What I Know? Situational Awareness of Co-located Teams in Multidisplay Environments

2010-05 Claudia Hauff (UT) Predicting the Effectiveness of Queries and Retrieval Systems

2010-06 Sander Bakkes (UvT) Rapid Adaptation of Video Game AI

2010-07 Wim Fikkert (UT) Gesture interaction at a Distance

2010-08 Krzysztof Siewicz (UL) Towards an Improved Regulatory Framework of Free Software. Protecting user freedoms in a world of software communities and eGovernments

2010-09 Hugo Kielman (UL) A Politiele gegevensverwerking en Privacy, Naar een effectieve waarborging

2010-10 Rebecca Ong (UL) Mobile Communication and Protection of Children

2010-11 Adriaan Ter Mors (TUD) The world according to MARP: Multi-Agent Route Planning

2010-12 Susan van den Braak (UU) Sensemaking software for crime analysis

2010-13 Gianluigi Folino (RUN) High Performance Data Mining using Bio-inspired techniques

2010-14 Sander van Splunter (VU) Automated Web Service Reconfiguration

2010-15 Lianne Bodenstaff (UT) Managing Dependency Relations in Inter-Organizational Models

2010-16 Sicco Verwer (TUD) Efficient Identification of Timed Automata, theory and practice

2010-17 Spyros Kotoulas (VU) Scalable Discovery of Networked Resources: Algorithms, Infrastructure, Applications

2010-18 Charlotte Gerritsen (VU) Caught in the Act: Investigating Crime by Agent-Based Simulation

2010-19 Henriette Cramer (UvA) People's Responses to Autonomous and Adaptive Systems

2010-20 Ivo Swartjes (UT) Whose Story Is It Anyway? How Improv Informs Agency and Authorship of Emergent Narrative

2010-21 Harold van Heerde (UT) Privacy-aware data management by means of data degradation

2010-22 Michiel Hildebrand (CWI) End-user Support for Access to
Heterogeneous Linked Data

2010-23 Bas Steunebrink (UU) The Logical Structure of Emotions

2010-24 Dmytro Tykhonov Designing Generic and Efficient Negotiation Strategies

2010-25 Zulfiqar Ali Memon (VU) Modelling Human-Awareness for Ambient Agents: A Human Mindreading Perspective

2010-26 Ying Zhang (CWI) XRPC: Efficient Distributed Query Processing on Heterogeneous XQuery Engines

2010-27 Marten Voulon (UL) Automatisch contracteren

2010-28 Arne Koopman (UU) Characteristic Relational Patterns

2010-29 Stratos Idreos(CWI) Database Cracking: Towards Auto-tuning Database Kernels

2010-30 Marieke van Erp (UvT) Accessing Natural History - Discoveries in data cleaning, structuring, and retrieval

2010-31 Victor de Boer (UVA) Ontology Enrichment from Heterogeneous Sources on the Web

2010-32 Marcel Hiel (UvT) An Adaptive Service Oriented Architecture: Automatically solving Interoperability Problems

2010-33 Robin Aly (UT) Modeling Representation Uncertainty in Concept-Based Multimedia Retrieval

2010-34 Teduh Dirgahayu (UT) Interaction Design in Service Compositions

2010-35 Dolf Trieschnigg (UT) Proof of Concept: Concept-based Biomedical Information Retrieval

2010-36 Jose Janssen (OU) Paving the Way for Lifelong Learning; Facilitating competence development through a learning path specification

2010-37 Niels Lohmann (TUE) Correctness of services and their composition

2010-38 Dirk Fahland (TUE) From Scenarios to components

2010-39 Ghazanfar Farooq Siddiqui (VU) Integrative modeling of emotions in virtual agents

2010-40 Mark van Assem (VU) Converting and Integrating Vocabularies for the Semantic Web

2010-41 Guillaume Chaslot (UM) Monte-Carlo Tree Search

2010-42 Sybren de Kinderen (VU) Needs-driven service bundling in a multi-supplier setting - the computational e3-service approach

2010-43 Peter van Kranenburg (UU) A Computational Approach to Content-Based Retrieval of Folk Song Melodies

2010-44 Pieter Bellekens (TUE) An Approach towards Context-sensitive and User-adapted Access to Heterogeneous Data Sources, Illustrated in the Television Domain

2010-45 Vasilios Andrikopoulos (UvT) A theory and model for the evolution of software services

2010-46 Vincent Pijpers (VU) e3alignment: Exploring Inter-Organizational Business-ICT Alignment

2010-47 Chen Li (UT) Mining Process Model Variants: Challenges, Techniques, Examples

2010-48 Withdrawn

2010-49 Jahn-Takeshi Saito (UM) Solving difficult game positions

2010-50 Bouke Huurnink (UVA) Search in Audiovisual Broadcast Archives

2010-51 Alia Khairia Amin (CWI) Understanding and supporting information seeking tasks in multiple sources

2010-52 Peter-Paul van Maanen (VU) Adaptive Support for Human-Computer Teams: Exploring the Use of Cognitive Models of Trust and Attention

2010-53 Edgar Meij (UVA) Combining Concepts and Language Models for Information Access

2011

2011-01 Botond Cseke (RUN) Variational Algorithms for Bayesian Inference in Latent Gaussian Models

2011-02 Nick Tinnemeier(UU) Organizing Agent Organizations. Syntax and Operational Semantics of an Organization-Oriented Programming Language

2011-03 Jan Martijn van der Werf (TUE) Compositional Design and Verification of Component-Based Information Systems

2011-04 Hado van Hasselt (UU) Insights in Reinforcement Learning; Formal analysis and empirical evaluation of temporal-difference learning algorithms

2011-05 Base van der Raadt (VU) Enterprise Architecture Coming of Age - Increasing the Performance of an Emerging Discipline.

2011-06 Yiwen Wang (TUE) Semantically-Enhanced Recommendations in Cultural Heritage

2011-07 Yujia Cao (UT) Multimodal Information Presentation for High Load Human Computer Interaction

2011-08 Nieske Vergunst (UU) BDI-based Generation of Robust Task-Oriented Dialogues

2011-09 Tim de Jong (OU) Contextualised Mobile Media for Learning

2011-10 Bart Bogaert (UvT) Cloud Content Contention

2011-11 Dhaval Vyas (UT) Designing for Awareness: An Experience-focused HCI Perspective

2011-12 Carmen Bratosin (TUE) Grid Architecture for Distributed Process Mining

2011-13 Xiaoyu Mao (UvT) Airport under Control. Multiagent Scheduling for Airport Ground Handling

2011-14 Milan Lovric (EUR) Behavioral Finance and Agent-Based Artificial Markets

2011-15 Marijn Koolen (UvA) The Meaning of Structure: the Value of Link Evidence for Information Retrieval

2011-16 Maarten Schadd (UM) Selective Search in Games of Different Complexity

2011-17 Jiyin He (UVA) Exploring Topic Structure: Coherence, Diversity and Relatedness

2011-18 Mark Ponsen (UM) Strategic Decision-Making in complex games

2011-19 Ellen Rusman (OU) The Mind ' s Eye on Personal Profiles

2011-20 Qing Gu (VU) Guiding service-oriented software engineering - A view-based approach

2011-21 Linda Terlouw (TUD) Modularization and Specification of Service-Oriented Systems

2011-22 Junte Zhang (UVA) System Evaluation of Archival Description and Access

2011-23 Wouter Weerkamp (UVA) Finding People and their Utterances in Social Media

2011-24 Herwin van Welbergen (UT) Behavior Generation for Interpersonal Coordination with Virtual Humans On Specifying, Scheduling and Realizing Multimodal Virtual Human Behavior

2011-25 Syed Waqar ul Qounain Jaffry (VU)) Analysis and Validation of Models for Trust Dynamics

2011-26 Matthijs Aart Pontier (VU) Virtual Agents for Human Communication - Emotion Regulation and Involvement-Distance Trade-Offs in Embodied Conversational Agents and Robots

2011-27 Aniel Bhulai (VU) Dynamic website optimization through autonomous management of design patterns

2011-28 Rianne Kaptein(UVA) Effective Focused Retrieval by Exploiting Query Context and Document Structure

2011-29 Faisal Kamiran (TUE) Discrimination-aware Classification

2011-30 Egon van den Broek (UT) Affective Signal Processing (ASP): Unraveling the mystery of emotions

2011-31 Ludo Waltman (EUR) Computational and Game-Theoretic Approaches for Modeling Bounded Rationality

2011-32 Nees-Jan van Eck (EUR) Methodological Advances in Bibliometric Mapping of Science

2011-33 Tom van der Weide (UU) Arguing to Motivate Decisions

2011-34 Paolo Turrini (UU) Strategic Reasoning in Interdependence: Logical and Game-theoretical Investigations

2011-35 Maaike Harbers (UU) Explaining Agent Behavior in Virtual Training

2011-36 Erik van der Spek (UU) Experiments in serious game design: a cognitive approach

2011-37 Adriana Burlutiu (RUN) Machine Learning for Pairwise Data, Applications for Preference Learning and Supervised Network Inference

2011-38 Nyree Lemmens (UM) Bee-inspired Distributed Optimization

2011-39 Joost Westra (UU) Organizing Adaptation using Agents in Serious Games

2011-40 Viktor Clerc (VU) Architectural Knowledge Management in Global Software Development

2011-41 Luan Ibraimi (UT) Cryptographically Enforced Distributed Data Access Control

2011-42 Michal Sindlar (UU) Explaining Behavior through Mental State Attribution

2011-43 Henk van der Schuur (UU) Process Improvement through Software Operation Knowledge

2011-44 Boris Reuderink (UT) Robust Brain-Computer Interfaces

2011-45 Herman Stehouwer (UvT) Statistical Language Models for Alternative Sequence Selection

2011-46 Beibei Hu (TUD) Towards Contextualized Information Delivery: A Rule-based Architecture for the Domain of Mobile Police Work

2011-47 Azizi Bin Ab Aziz(VU) Exploring Computational Models for Intelligent Support of Persons with Depression

2011-48 Mark Ter Maat (UT) Response Selection and Turn-taking for a Sensitive Artificial Listening Agent

2011-49 Andreea Niculescu (UT) Conversational interfaces for task-oriented spoken dialogues: design aspects influencing interaction quality

2012

2012-01 Terry Kakeeto (UvT) Relationship Marketing for SMEs in Uganda

2012-02 Muhammad Umair(VU) Adaptivity, emotion, and Rationality in Human and Ambient Agent Models

2012-03 Adam Vanya (VU) Supporting Architecture Evolution by Mining Software Repositories

2012-04 Jurriaan Souer (UU) Development of Content Management System-based Web Applications

2012-05 Marijn Plomp (UU) Maturing Interorganisational Information Systems

2012-06 Wolfgang Reinhardt (OU) Awareness Support for Knowledge Workers in Research Networks

2012-07 Rianne van Lambalgen (VU) When the Going Gets Tough: Exploring Agent-based Models of Human Performance under Demanding Conditions

2012-08 Gerben de Vries (UVA) Kernel Methods for Vessel Trajectories

2012-09 Ricardo Neisse (UT) Trust and Privacy Management Support for Context-Aware Service Platforms

2012-10 David Smits (TUE) Towards a Generic Distributed Adaptive Hypermedia Environment

2012-11 J.C.B. Rantham Prabhakara (TUE) Process Mining in the Large: Preprocessing, Discovery, and Diagnostics

2012-12 Kees van der Sluijs (TUE) Model Driven Design and Data Integration in Semantic Web Information Systems

2012-13 Suleman Shahid (UvT) Fun and Face: Exploring non-verbal expressions of emotion during playful interactions

2012-14 Evgeny Knutov(TUE) Generic Adaptation Framework for Unifying Adaptive Web-based Systems

2012-15 Natalie van der Wal (VU) Social Agents. Agent-Based Modelling of Integrated Internal and Social Dynamics of Cognitive and Affective Processes.

2012-16 Fiemke Both (VU) Helping people by understanding them - Ambient Agents supporting task execution and depression treatment

2012-17 Amal Elgammal (UvT) Towards a Comprehensive Framework for Business Process Compliance

2012-18 Eltjo Poort (VU) Improving Solution Architecting Practices

2012-19 Helen Schonenberg (TUE) What's Next? Operational Support for Business Process Execution

2012-20 Ali Bahramisharif (RUN) Covert Visual Spatial Attention, a Robust Paradigm for Brain-Computer Interfacing

2012-21 Roberto Cornacchia (TUD) Querying Sparse Matrices for Information Retrieval

2012-22 Thijs Vis (UvT) Intelligence, politie en veiligheidsdienst: verenigbare grootheden?

2012-23 Christian Muehl (UT) Toward Affective Brain-Computer Interfaces: Exploring the Neurophysiology of Affect during Human Media Interaction

2012-24 Laurens van der Werff (UT) Evaluation of Noisy Transcripts for Spoken Document Retrieval

2012-25 Silja Eckartz (UT) Managing the Business Case Development in Inter-Organizational IT Projects: A Methodology and its Application

2012-26 Emile de Maat (UVA) Making Sense of Legal Text

2012-27 Hayrettin Gurkok (UT) Mind the Sheep! User Experience Evaluation and Brain-Computer Interface Games

2012-28 Nancy Pascall (UvT) Engendering Technology Empowering Women

2012-29 Almer Tigelaar (UT) Peer-to-Peer Information Retrieval

2012-30 Alina Pommeranz (TUD) Designing Human-Centered Systems for Reflective Decision Making

2012-31 Emily Bagarukayo (RUN) A Learning by Construction Approach for Higher Order Cognitive Skills Improvement, Building Capacity and Infrastructure

2012-32 Wietske Visser (TUD) Qualitative multi-criteria preference representation and reasoning

2012-33 Rory Sie (OUN) Coalitions in Cooperation Networks (COCOON)

2012-34 Pavol Jancura (RUN) Evolutionary analysis in PPI networks and applications

2012-35 Evert Haasdijk (VU) Never Too Old To Learn – On-line Evolution of Controllers in Swarm- and Modular Robotics

2012-36 Denis Ssebugwawo (RUN) Analysis and Evaluation of Collaborative Modeling Processes

2012-37 Agnes Nakakawa (RUN) A Collaboration Process for Enterprise Architecture Creation

2012-38 Selmar Smit (VU) Parameter Tuning and Scientific Testing in Evolutionary Algorithms

2012-39 Hassan Fatemi (UT) Risk-aware design of value and coordination networks

2012-40 Agus Gunawan (UvT) Information Access for SMEs in Indonesia

2012-41 Sebastian Kelle (OU) Game Design Patterns for Learning

2012-42 Dominique Verpoorten (OU) Reflection Amplifiers in self-regulated Learning

2012-43 Withdrawn

2012-44 Anna Tordai (VU) On Combining Alignment Techniques

2012-45 Benedikt Kratz (UvT) A Model and Language for Business-aware Transactions

2012-46 Simon Carter (UVA) Exploration and Exploitation of Multilingual Data for Statistical Machine Translation

2012-47 Manos Tsagkias (UVA) Mining Social Media: Tracking Content and Predicting Behavior

2012-48 Jorn Bakker (TUE) Handling Abrupt Changes in Evolving Time-series Data

2012-49 Michael Kaisers (UM) Learning against Learning - Evolutionary dynamics of reinforcement learning algorithms in strategic interactions

2012-50 Steven van Kervel (TUD) Ontologogy driven Enterprise Information Systems Engineering

2012-51 Jeroen de Jong (TUD) Heuristics in Dynamic Sceduling; a practical framework with a case study in elevator dispatching

2013

2013-01 Viorel Milea (EUR) News Analytics for Financial Decision Support

2013-02 Erietta Liarou (CWI) MonetDB/DataCell: Leveraging the Column-store Database Technology for Efficient and Scalable Stream Processing

2013-03 Szymon Klarman (VU) Reasoning with Contexts in Description Logics

2013-04 Chetan Yadati(TUD) Coordinating autonomous planning and scheduling

2013-05 Dulce Pumareja (UT) Groupware Requirements Evolutions Patterns

2013-06 Romulo Goncalves(CWI) The Data Cyclotron: Juggling Data and Queries for a Data Warehouse Audience

2013-07 Giel van Lankveld (UvT) Quantifying Individual Player Differences

2013-08 Robbert-Jan Merk(VU) Making enemies: cognitive modeling for opponent agents in fighter pilot simulators

2013-09 Fabio Gori (RUN) Metagenomic Data Analysis: Computational Methods and Applications

2013-10 Jeewanie Jayasinghe Arachchige(UvT) A Unified Modeling Framework for Service Design.

2013-11 Evangelos Pournaras(TUD) Multi-level Reconfigurable Self-organization in Overlay Services

2013-12 Marian Razavian(VU) Knowledge-driven Migration to Services

2013-13 Mohammad Safiri(UT) Service Tailoring: User-centric creation of integrated IT-based homecare services to support independent living of elderly

2013-14 Jafar Tanha (UVA) Ensemble Approaches to Semi-Supervised Learning Learning

2013-15 Daniel Hennes (UM) Multiagent Learning - Dynamic Games and Applications

2013-16 Eric Kok (UU) Exploring the practical benefits of argumentation in multi-agent deliberation

2013-17 Koen Kok (VU) The PowerMatcher: Smart Coordination for the Smart Electricity Grid

2013-18 Jeroen Janssens (UvT) Outlier Selection and One-Class Classification

2013-19 Renze Steenhuizen (TUD) Coordinated Multi-Agent Planning and Scheduling

2013-20 Katja Hofmann (UvA) Fast and Reliable Online Learning to Rank for Information Retrieval

2013-21 Sander Wubben (UvT) Text-to-text generation by monolingual machine translation

2013-22 Tom Claassen (RUN) Causal Discovery and Logic

2013-23 Patricio de Alencar Silva(UvT) Value Activity Monitoring

2013-24 Haitham Bou Ammar (UM) Automated Transfer in Reinforcement Learning

2013-25 Agnieszka Anna Latoszek-Berendsen (UM) Intention-based Decision Support. A new way of representing and implementing clinical guidelines in a Decision Support System

2013-26 Alireza Zarghami (UT) Architectural Support for Dynamic Homecare Service Provisioning

2013-27 Mohammad Huq (UT) Inference-based Framework Managing Data Provenance

2013-28 Frans van der Sluis (UT) When Complexity becomes Interesting: An Inquiry into the Information eXperience

2013-29 Iwan de Kok (UT) Listening Heads

2013-30 Joyce Nakatumba (TUE) Resource-Aware Business Process Management: Analysis and Support

2013-31 Dinh Khoa Nguyen (UvT) Blueprint Model and Language for Engineering Cloud Applications

2013-32 Kamakshi Rajagopal (OUN) Networking For Learning; The role of Networking in a Lifelong Learner's Professional Development

2013-33 Qi Gao (TUD) User Modeling and Personalization in the Microblogging Sphere

2013-34 Kien Tjin-Kam-Jet (UT) Distributed Deep Web Search

2013-35 Abdallah El Ali (UvA) Minimal Mobile Human Computer Interaction Promotor: Prof. dr. L. Hardman (CWI/UVA)

2013-36 Thanh Lam Hoang (TUe) Pattern Mining in Data Streams

2013-37 Dirk Börner (OUN) Ambient Learning Displays

2013-38 Eelco den Heijer (VU) Autonomous Evolutionary Art

2013-39 Joop de Jong (TUD) A Method for Enterprise Ontology based Design of Enterprise Information Systems

2013-40 Pim Nijssen (UM) Monte-Carlo Tree Search for Multi-Player Games

2013-41 Jochem Liem (UVA) Supporting the Conceptual Modelling of Dynamic Systems: A Knowledge Engineering Perspective on Qualitative Reasoning

Curriculum Vitae

Lam was born in Vietnam in 1982 where he got education at a high school for gifted students in mathematics. In 2002, he was sent to Russia by the Vietnamese Oversea Scholarship program to study at Lomonosov Moscow State University. In 2007, Lam got the diploma degree with honor (red diploma) in applied mathematics and computer science after defending his dissertation on parallel particle swarm optimization algorithms for document clustering problems. In 2008, Lam got the PhD scholarship at the Galileo Galilei PhD School, university of Pisa Italy where he was working on the indexing problems for information retrieval systems. After one year working at university of Pisa, in 2009, Lam decided to move to the Netherlands where he can spend time working on the PhD project with the topic that attracts him a lot since his undergraduate study: data mining. At Technical University of Eindhoven, besides doing research on the PhD project on mining patterns from data streams, Lam involved in advising master students who have been doing practical data mining projects at industry labs of Phillips and the ASML company. In 2011, Lam was awarded a research internship opportunity at Siemens Research Center in Princeton NJ, USA, where he spent more than two months working on the project which mines useful patterns from sequence data for the machine failure prediction problem. His work at Siemens was published at the SIAM data mining conference in 2012 and was nominated as one of the best papers of that conference.