Mining Compressing Sequential Patterns

Hoang Thanh Lam[♯], Fabian Mörchen[§], Dmitriy Fradkin[§], and Toon Calders[♯]
[§]Siemens Corporation [♯]Technische Universiteit Eindhoven
Corporate Research and Technology Department of Maths and Computer Science
Princeton, NJ, USA Eindhoven, Netherlands
Emails: t.l.hoang@tue.nl, fabian.moerchen@siemens.com, dmitriy.fradkin@siemens.com, and t.calders@tue.nl

Abstract
Compression based pattern mining has been successfully applied to many data mining tasks. We propose an approach based on the minimum description length principle to extract sequential patterns that compress a database of sequences well. We show that mining compressing patterns is NP-Hard and belongs to the class of inapproximable problems. We propose two heuristic algorithms to mining compressing patterns. The first uses a two-phase approach similar to Krimp for itemset data. To overcome performance with the required candidate generation we propose GoKrimp, an effective greedy algorithm that directly mines compressing patterns. We conduct an empirical study on six real-life datasets to compare the proposed algorithms by run time, compressibility, and classification accuracy using the patterns found as features for SVM classifiers.

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 are proposed to find sequential patterns effectively [12]. 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 Pioneer dataset which contains sequences of a robot's actions when it is moving around. Table 1 shows the 10 most frequent closed sequential patterns in order of decreasing frequency. This set of patterns is clearly very redundant. The first and the second patterns are almost identical, and patterns in positions 3 through 10 are also very similar. A significant negative consequence of such redundancy is that a huge number of patterns are sent to the users although only a few of them are actually interesting or novel.

In order to solve the redundancy problem, we have to find alternative interestingness measures rather than continuing to rely on frequency alone. For itemset data, an interesting approach has been proposed recently. The Krimp algorithm mines patterns that compress the data well [1] using the minimum description length (MDL) principle [10]. This approach has been shown to reduce redundancy and generate patterns that are useful for classification [1], components identifying [3] and change detection [4].

The success of the Krimp algorithm [1-4] on itemset data calls for greater effort on theoretical, algorithmic and empirical aspects of MDL-based algorithms not only for itemset data but also for the other popular types of data such as sequence and graph data.

However, 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 able to deal with gaps, overlapping, and repeating patterns; all properties not present in itemset data. This paper proposes MDL-based algorithms for sequence data. Although our solutions are inspired by Krimp, the bonus contributions of the work are as follows:

1. We propose an encoding scheme for sequence data.

Table 1: 10 most frequent closed patterns in Pioneer dataset

Frequency	Patterns
125	RobotLow$^-$, RobotHigh$^+$, RobotHigh$^-$
124	RobotLow$^-$, RobotHigh$^+$, RobotHigh$^-$,RobotLow$^+$
123	R-Wheel Medium$^-$, RobotHigh$^-$
123	L-Wheel Medium$^-$, RobotHigh$^-$
122	R-Wheel Medium$^-$,R-Wheel Medium$^+$, RobotLow$^+$
122	R-Wheel Medium$^-$, L-Wheel Medium$^+$, RobotLow$^+$
122	R-Wheel Medium$^-$, RobotHigh$^-$, RobotLow$^+$
122	L-Wheel Medium$^-$, R-Wheel Medium$^+$, RobotLow$^+$
122	L-Wheel Medium$^-$, L-Wheel Medium$^+$, RobotLow$^+$
122	L-Wheel Medium$^-$, RobotHigh$^-$, RobotLow$^+$

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 algorithms that avoids the expensive candidate generation phase and directly mines compressing patterns from the data. GoKrimp is shown to find compression patterns of similar quality to SeqKrimp while being orders of magnitude faster.

5. We perform an empirical study with real-life data to compare different sets of patterns when they are using as attributes for classification tasks

2 Preliminaries

2.1 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 identifier from the alphabet Σ and $t(e_i)$ is timestamp of event e_i. Given a sequence P, we say that S *matches* P if P is a subsequence of S. There might be a lot of instances of P in S, however, in a very long sequence only those contained completely in windows of size W are considered. Two instances of P in S are *overlapping* if they share a common event.

Let $D = \{S_1, S_2, \cdots, S_n\}$ 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 1. (Frequent Mining) *Given a sequence database D, a window size constraint condition and a minimum support value minsup, find all the sequences of events (patterns) P such that $f_P \geq minsup$.*

A pattern P is called *closed* if it is frequent and there doesn't exist another sequence Q such that $f_P = f_Q$ and $P \subset Q$. The problem of mining all closed frequent patterns is formulated as follows:

Definition 2. (Closed Mining) *Given a database of sequences D, a window size constraint condition and a minimum support value minsup, find all patterns P such that $f_P \geq minsup$ and P is closed.*

2.2 Minimum Description Length Principle Although the set of closed sequential patterns is typically much smaller than the set of all frequent patterns, it still contains a lot of redundancy. Many patterns are highly overlapping and uninteresting. Reducing the number of uninteresting patterns is an active and emerging research area [8]. The proposed solutions can be grouped into three main categories: statistical approaches such as hypothesis testing [19, 22], information-theoretic approaches [6, 7] and MDL-based approaches [1]. This work focuses on the latter direction. Let us start explaining the MDL principle in the following definition:

Definition 3. (Hypothesis) *A hypothesis H is a set of subsequences $\{P_1, P_2, \cdots, P_h\}$.*

Let $L_C(H)$ be the description length of hypothesis H and $L_C(D|H)$ be the description length of data D when encoded with the help of the hypothesis and an encoding scheme C. For brevity, we will omit C when the encoding scheme is clear from the context. Informally, the MDL principle states that the best hypothesis always compresses the data most. Therefore, a two-part version of the MDL principle suggests that we should look for hypothesis H and the encoding scheme C such that $L_C(D) = L_C(H) + L_C(D|H)$ is minimized.

The MDL principle is very general and flexible. A central question in designing an MDL-based algorithm is: how can we encode data given a hypothesis? The answer to this question is left to the algorithm's designers. Depending on which problem and which dataset they are working on, different data encoding schemes could be used. In this work, we propose a novel encoding scheme for sequence data.

2.3 Pattern Mining and Data Compression There is straightforward connection from data compression to pattern mining. Data compression looks for regularities in data and uses them to compress the data. These regularities usually correspond to important patterns in data. For example, a data compression algorithm may discover that, most of the time, an event a is followed by event b. This regularity could be an important pattern in the data which could also be discovered by frequent pattern mining.

However, there are also important differences between pattern mining and data compression. Data compression looks for any type of regularity with the only goal of reducing the size of the representation. Not all of the regularities in the data utilized for compression are necessarily interesting from a data mining point of view. For example, the Huffman coding [18] encodes individual character by a bit string with the length proportional to the entropy of the given character. This entropy-encoding scheme is optimal for data generated by independent character generation sources. However,

Table 2: A database and a dictionary

Data D	**Dictionary** H
S_1=(a,1)(c,3)(a,4)(b,7)(c,8)(b,9)	1: ab
S_2= (c,2)(a,4)(b,5)(c,6)(a,7)(b,8)	2: cab

it does not uncover any sequential structure in the data - which is our primary goal. Hence we have to design a compression approach that exploits existing sequential structure for compression.

3 Problem Definitions

The most important aspect of designing an MDL-based algorithm is choosing an appropriate encoding scheme. We start this section by explaining how to encode the data given a hypothesis.

3.1 Dictionary-based Encoding Before discussing our encoding for sequence data we revisit the encoding scheme used in the Krimp algorithm proposed by Vreeken et al. [1]. An itemset I is encoded with the help of itemset patterns by replacing every non-overlapping instance of a pattern occurring in I with a pointer to the pattern in a code table (dictionary). This way an itemset can be encoded to a more compact representation and decoded back to the original itemset.

In this paper we use a similar dictionary-based encoding scheme for sequence database. Given a dictionary consisting of sequential patterns $H = \{P_1, P_2, \cdots, P_m\}$, a sequence of events S is encoded by replacing instances of any pattern P_i in S with pointers to dictionary.

An important difference between itemset data and sequence data is that events in a sequence are ordered (and may even be associated with timestamps). Since sequential patterns can overlap in time with each other and with single events (not encoded), we need to store additional information to ensure accurate decoding back to the original sequence. Note, that this is in contrast with existing compression algorithms for sequence data such as text. These algorithms typically use frequent sequences of symbols but they need to be consecutive and cannot overlap.

As an example, consider dictionary H in table 2. Using H, sequence S_1 can be encoded as $S_1^c = p_1^1 p_2^1 (c, 8)$, where every pointer consists of two parts. The first part is the identifier of the corresponding pattern in the dictionary and the second part contains information about the timestamps of every events. So p_1^1 is a replacement of the subsequence ab corresponding to the first a and b in S, i.e. $p_1^1 = [1|1, 7]$. Similarly, $p_2^1 = [2|3, 4, 9]$ replaces cab in the same manner. Note that patterns p_1^1 and p_2^1 overlap in time with each other and with $(c, 8)$.

Definition 4. (Dictionary Encoding) *A dictionary encoding of a sequence S consists of single events with timestamps and of codebook pointers. A pointer consists of two parts: a reference to the codeword (a subsequence of S it replaces) and the timestamps of the events in the replaced subsequence.*

At first, it may seem that we are not doing a great job here in compressing the data because of all the book-keeping needed to ensure accurate decoding. In particular for sequences of events not associated with timestamps such as DNA sequences or text we may incur extra cost for encoding the ordering information. This may lead to a surprising situation where the compressed data is larger than the uncompressed data. This is disconcerting from perspective of pure data compression, but not from perspective of pattern mining purposes, because the aim is **not** to compress the data. *Data compression tries to compress data best, meanwhile an MDL-based pattern mining approach simply evaluates patterns by how well they compress the data.*

Therefore, it is not important if the encoded data is larger or smaller than the original data. It is more important that the encoding scheme reflects interestingness of the set of patterns in the sense that the most interesting set of patterns must compress the data better than other sets of patterns. One encoding scheme that does not exploit any sequential structure is to simply use a codebook with all single events. Our encoding scheme results in the same overhead, in the form of time-stamp keeping information, with this codebook as it does with codebooks with longer sequential patterns. So this cost does not affect comparison of two codebooks. In fact we drop this constant cost when calculating the data description length (see below).

3.2 Data Description Length We have defined an encoding scheme for sequences. The next step is to determine the description length of data under such encoding. Traditionally the MDL principle suggests calculating the description length as the number of bits, requiring a transformation of the encoded data into bit representations. However, bit representation lengths can vary significantly depending on the type of bit encoding, e.g. arithmetic code or any type of prefix-free codes, introducing undesired dependency on bit encoding into performance of MDL-based approaches, and causing difficulties in theoretical algorithm analysis and empirical evaluation.

In this work, we assume that any number or character in data has a fixed length bit representation which requires a unit memory cell. This enables us to avoid the issue of bit representation. In particular, the description length of a dictionary can be calculated as the total lengths of the patterns and the number of associated pointers, i.e. $L(H) = \sum_{i=1}^{m} |P_i| + |H|$.

Besides, the length of the data D when encoded with the help of dictionary H can be calculated as $L(D|H) = \sum_{i=1}^{n} |S_i^c|$. For example, the description length of the dictionary in Table 2 is calculated as $L(H) = |ab| + |cab| + |H| = 2 + 3 + 2 = 7$. Similarly, the description length of the the sequence S_1 encoded by H is $L(S_1|H) = |p_1^1| + |p_2^1| + |(c, 8)| = 3 + 4 + 2 = 9$.

Note: For any given dictionary and the data D, $L(D)$ contains an invariant. The cost of storing the timestamp information is always constant (proportional to the number of events in the data) regardless the size of the dictionary. Therefore, the comparison of two encoding schemas is independent of the timestamp information. In other words, an encoding is better than another if it is able to replace more long and frequent subsequences by pointers to the dictionary.

3.3 Problem Definition

Multiple dictionary encodings can result from replacing subsequences of S with the patterns in H. Different replacements lead to different description lengths of S. For example, the sequence S_1 in Table 2 can be alternatively encoded as $S_1^c = p_1^1(c, 3)p_1^2, (c, 8)$, where $p_1^1 = [1|1, 7]$, $p_1^2 = [1|4, 9]$. The description length of the sequence S_1 with this replacement is $L(S_1|H) = 3 + 2 + 3 + 2 = 10$.

Given a dictionary H, denote $L^*(D) = L^*(H) + L^*(D|H) = argmin_C\{L_C(H) + L_C(D|H)\}$ as the shortest description length of D when encoded with the help of dictionary H. The problem of finding compressing patterns can be formulated as follows:

Definition 5. (Compressing Patterns Problem) *Given a sequence database D, a constraint window condition, a set of candidate patterns $\mathbf{C} = \{P_1, P_2, ..., P_m\}$, find an optimal dictionary $H^* = argmin_H L^*(H) + L^*(D|H)$, where $H^* \subset \mathbf{C}$.*

Definition 6. (Optimal Encoding Problem) *Given a sequence database D, a constraint window condition and a dictionary H, find an optimal encoding of the data D with the help of the dictionary H.*

In order to solve the compressing patterns problem we need to find at the same time the optimal dictionary and the optimal replacement of data D when encoded with the help of the optimal dictionary.

4 Complexity Analysis

This section discusses the complexity of the mining compressing patterns problems. The main result states that finding optimal set of compressing pattern is hard.

Lemma 1. *The compressing pattern problem and the optimal dictionary problem are NP-Hard.*

Proof. For brevity, we only prove the NP-hardness of the former problem by reducing it to the *Data Compression by Textual Substitution Problem* (DCTS), which is known to be NP-Hard [25]. DCTS problem looks for a dictionary and a replacement of strings (subsequences with consecutive events) in a given sequence S such that the compression length is minimized. The compression length is calculated in the same way as we have discussed in subsection 3.2.

We can easily reduce the mining optimal compressing patterns problem to the DCTS problem by setting the set of candidate $\mathbf{C}$ as the set of all possible substrings of the sequence S. In addition to that, the window constraint condition is set such that every instance of a pattern P in S is completely contained in a window of size being equal to the size of pattern. In other words, P is only allowed to match with a substring in S. In doing so, finding the optimal compressing patterns is as hard as solving the corresponding instance of the DCTS problem. This proves the NP-hardness property of the given problem.

5 Algorithms

Since mining optimal compressing patterns is NP-hard, we propose two heuristic algorithms inspired by the idea of the Krimp algorithm. The first one called SeqKrimp is described in the following subsection.

5.1 SeqKrimp: Krimp Algorithm for Sequence Database

SeqKrimp, described in Algorithm 1, consists of two steps. In the first step, a set of candidate patterns is generated by using a traditional frequent closed sequential patterns mining algorithm (line 3). The second step is a strategy for greedily choosing the next pattern to put into the dictionary. This is achieved by calculating the compression size of the data D assuming that it is encoded by a pattern $P \in C$ (lines 6-8). Function *CompressionSize*, described in Algorithm 2, returns the optimal compression length $L^*(D|P)$ of data D when encoded with the help of pattern P (line 7). Subsequently, the most compressing pattern P^* is appended into the dictionary (lines 9-10) and removed from the set of candidates (line 11). Finally, all the instances of P^* in the data D are replaced by the pointers to P^* in the dictionary (line 12). These actions are repeated as long as the dictionary H is not filled up with K distinct patterns.

An important part of SeqKrimp is calculating the value of $L^*(D|P)$, i.e. the optimal compression size

Algorithm 1 SeqKrimp(D, K)

```
1:  Input: Database $D$ and parameter $K$
2:  Output: compressing patterns $H$
3:  $\mathbf{C} \longleftarrow$ GetCandidate()
4:  $H \longleftarrow \{\emptyset\}$
5:  while $|H| < K$ do
6:      for $P \in \mathbf{C}$ do
7:          $L^*(D|P) \longleftarrow$ CompressSize($D|P$)
8:      end for
9:      $P^* \longleftarrow argmin_P(L^*(D|P))$
10:     $H \longleftarrow H \cup \{P^*\}$
11:     $\mathbf{C} \longleftarrow \mathbf{C} \setminus \{P^*\}$
12:     Replace all instances of $P^*$ by its pointers
13: end while
14: Return $H$
```

Algorithm 2 CompressSize($D|P$)

```
1:  Input: Database $D = \{S_1, S_2, \cdots, S_n\}$ and pattern $P$
2:  Output: $L^*(D|P)$ optimal compressing size of $D$ given it is encoded by $P$
3:  $c \longleftarrow |P| + \Sigma_{i=1}^n |S_i|$
4:  for $S_i \in D$ do
5:      while $S_i$ has an instance of P do
6:          $s \longleftarrow$ the left-most instance of $P$ in $S_i$
7:          Remove $s$ from $S_i$
8:          $c = c - |P| + 1$
9:      end while
10: end for
11: Return $c$
```

of data D when encoded with the help of pattern P. This could be done by first looking for the maximum set of non-overlapping instances of the given pattern P in each sequence S_i of the database. Subsequently, by replacing each of the maximum non-overlapping instance with a pointer to the dictionary we obtain an optimal replacement of P in D.

Let us represent every instance of P in S_i as a vertex in a graph. There is an edge between two vertices if the corresponding instances are overlapping. The problem of finding maximum set of non-overlapping instances turns out to be the classical maximum independent set problem on graph. This problem for an arbitrary graph is NP-Hard. However, with the given graph this problem has an efficient solution, which is linear in the size of graph and described in algorithm 2.

Let us consider two arbitrary sequences $A = (a_1, t(a_1)), (a_2, t(a_2)), \cdots, (a_p, t(a_p))$ and $B = (b_1, t(b_1)), (b_2, t(b_2)), \cdots, (b_p, t(b_p))$. We say that A is *earlier* than B if there exists a number $q (0 < q < p)$ such that $t(a_j) = t(b_j) \forall 0 < j < q$ and $t(a_q) < t(b_q)$. We call an instance of P in a sequence S_i the *left-most instance* if it is the earliest instance of P in that sequence. The main idea behind the algorithm to find the maximum set of non-overlapping instances of P in S (lines 5-9 Algorithm 2) is that it always finds the most left instance of P in S. Then this instance is removed from S. This action is repeated until there are no more instances of P in S.

LEMMA 2. *Algorithm 2 finds the correct optimal compression size $L^*(D|P)$ of data D given it is encoded with the help of pattern P.*

Proof. [Sketch] We sketch the main idea of the proof of this lemma. The correctness of Algorithm 2 is decided by the fact that it correctly finds the maximum set of non-overlapping instances of P in every sequence S_i. For a given sequence S we claim that there exists a maximum set of non-overlapping instances of P in S such that it always contains the left most instance of P. If this claim is true then the correctness of the lemma is directly followed.

The main idea of the proof is as follows: assume that we have a maximum set M of non-overlapping instances of P but it does not contain the left-most instance of P in S. Then we can create a new maximum set of non-overlapping instances of P by either exchanging events between instances of M or events from outside with events in M such that the earliest instance of the new set is strictly earlier than the earliest instance of the former set. By repeating this action we will finally end up with a maximum set of non-overlapping instances including the left-most instance of P in S.

5.2 GoKrimp: Direct Mining Of Compressing Patterns

5.2.1 Disadvantages of SeqKrimp:

SeqKrimp suffers from two problems. First, it has to generate candidate set before performing a greedy pattern selection procedure. Hence, performance of SeqKrimp is highly dependent on the candidate set. One could claim that the frequent patterns is a good candidate set but another one may also claim that the set of closed patterns or margin closed is better [15]. As a result, the users who want to deploy a Krimp-based pattern mining approach have to choose the candidate set by themselves which is not a trivial task in many applications.

Second, the candidate generation step is usually very expensive. Even for moderate-size datasets the state-of-the-art algorithms for extracting frequent or closed patterns from sequence database such as PrefixSpan [11] or BIDE algorithm [9, 13] are very time-

Algorithm 3 GoKrimp(D, K)

```
1: Input: Database $D = \{S_1, S_2, \cdots, S_n\}$ and parameter $K$
2: Output: $H$ the set of compressing patterns
3: $H \longleftarrow \{\emptyset\}$
4: while $|H| < K$ do
5:     $P^* \longleftarrow$ BestCompressing($D$))
6:     $H \longleftarrow H \cup \{P^*\}$
7:     Delete from $D$ all subsequences replaced by pointer to $P^*$
8: end while
9: return $H$
```

consuming. Therefore, in this section we propose a very simple, but fast and effective algorithm which looks for compressing patterns directly from the data, without generating a candidate set. Interestingly, we show that the given algorithm is closely related to the maximum tiling algorithm [16].

5.2.2 The most compressing pattern:

Recall that SeqKrimp greedily chooses the best compressing pattern among a set of candidates. Can we directly find the most compressing pattern without the candidate generation step?

Let $f^*_D(P)$ be the maximum number of non-overlapping instances of P in D, when D is clear from the context we just use $f^*(P)$ without the subscript. The benefit of using the pattern P to compress data D can be calculated as $C^*(P) = \sum_i^n |S_i| - \sum_i^n |S_i^c| - |P| = f^*(P)|P| - f^*(P) - |P|$, where $f^*(P)|P|$ is the total size of the instances of P before compression. Term $f^*(P)$ is due to the number of pointers replacing these instances in D and $|P|$ is the cost of storing P in the dictionary.

For instance, when $P = ab$, two sequences S_1 and S_2 in Table 2 can be encoded using P as follows: $S_1^c = p_1^1(c,3)p_1^2(c,8)$ and $S_2^c = (c,2)p_1^3(c,6)p_1^4$. The benefit of using P to encode D is equal to the difference between the data size before and after encoding, i.e $24 - 20 - 2 = 2$

Therefore, looking for the most compressing pattern is equivalent to looking for the pattern with the maximum compression benefit $C^*(P) = f^*(P)|P| - f^*(P) - |P|$). The value of $C^*(P)$ is strongly related to the area of pattern P as defined in the work on tiling database [16]. The following lemma shows the hardness of finding the most compressing pattern:

Lemma 3. *Given data D, the problem of finding the sequential pattern P^* such that $P^* = argmax_P C^*(P)$ is NP-Hard.*

Proof. We can prove the lemma by reducing the given

Algorithm 4 BestCompressing(D)

```
1: Input: Database $D = \{S_1, S_2, \cdots, S_n\}$
2: Output: $P$ the best compressing pattern
3: $P \longleftarrow \{\emptyset\}$
4: $b \longleftarrow 0$
5: while (true) do
6:     $e \longleftarrow$ event in $D$ with highest support
7:     Append $e$ to P
8:     if CompressSize($D|P$)$< b$ then
9:         $D \longleftarrow D$ projected to $e$
10:    else
11:        Remove $e$ from $P$ iff $|P| > 1$
12:        break
13:    end if
14: end while
15: return $P$
```

problem to the maximum tiling problem [16]. Given a database of transactions, each transaction is an itemset, the maximum tiling problem looks for the itemset with largest area, i.e. with the maximum value of $f^*(P)|P|$.

Let us consider an arbitrary instance of the maximum tiling problem with a database $D = \{T_1, T_2, \cdots, T_n\}$, where each transaction T_i is a subset of an alphabet $\Sigma = \{a_1, a_2, \cdots, a_m\}$. We create an instance of the maximum compressing sequential pattern problem as follows. Let $S = \{S_1, S_2, \cdots, S_n\}$ be a database of sequences where each sequence S_i is created from T_i by ordering items in T_i lexicographically.

Besides, we also add to the alphabet Σ a new character $\sharp$, different from any existing character in Σ. This character is appended to the end of every sequence S_i. In addition to that an extra sequence S_{n+1} being equal to Σ appended by the extra character $\sharp$ is added to S. It is straightforward to show that the most compressing sequential pattern of S has the form $P\sharp$, where P is the largest tile of the database D.

Lemma 4. *Given data D, the problem of finding the sequential pattern P^* such that $P^* = argmax_P C^*(P)$ is inapproximable.*

Proof. The maximum tiling problem is equivalent to the Maximum Edge Biclique problem [16] which is inapproximable [17]. Given an arbitrary instance of the maximum tiling problem, we create a sequence database S in the same way as in lemma 3.

Assume that there is a polynomial algorithm that can find a solution P such that $C^*(P) > \alpha C^*(P^*\sharp)$ $(0 < \alpha < 1)$ then $f^*_S(P).|P| - f^*_S(P) - |P| > \alpha f^*_D(P^*).|P^*|$.

If P has the form $Q\sharp$, then $f^*_D(Q).|Q| + 1 > \alpha f^*_D(P^*).|P^*|$, for which, Q approximates the maximum tile P^* with the constant factor $\frac{\alpha}{4}$.

If P does not contain $\sharp$, then $(f^*_D(P) + 1).|P| - f^*_D(P) - |P| - 1 > \alpha f^*_D(P^*).|P^*|$, for which, P approximates the maximum tile P^* with the constant factor α. Therefore, by approximating the maximum compressing pattern we obtain an approximation of the maximum tiling which is in contradiction with the results of [17].

5.2.3 Direct mining of compressing patterns:

GoKrimp, shown in Algorithm 3, directly approximates the most compressing patterns P^* in D (line 5). Subsequently, every instance of P^* which was replaced by a pointer to the dictionary is removed from D and P^* is included into the set of patterns H (lines 6-7). These actions are repeated until H is filled up with K patterns.

Function *BestCompressing*(D) in Algorithm 4 heuristically returns the best compressing pattern P^* from D. This algorithm implements a greedy strategy in which pattern P is extended until no more compression benefit can be obtained. In particular, *BestCompressing* starts with an empty pattern $P = \emptyset$ (line 3). Subsequently, P is extended with the most frequent event e in D (lines 6-7). If the extension gives compression benefit (line 8) then the original data is updated by its projection to the given event (line 9). If the extension does not give any compression benefit, function returns.

6 Experiments and Results

This section discusses results of experiments carried out on real-life datasets. We will compare the set of patterns produced by SeqKrimp and GoKrimp algorithms with the set of closed sequential patterns extracted from the data using the BIDE algorithm [9, 13]. BIDE was chosen because it is *the-state-of-the art* approach for closed sequential pattern mining. BIDE is also used to generate the set of candidates for SeqKrimp, i.e. a replacement of the **GetCandidate**(.) function in line 3 Algorithm 1. We evaluate the proposed approaches according to the following criteria:

1. *Run time* to measure the scalability of the approaches.

2. *Compression* to measure how well the data is compressed.

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

Compressing patterns may be useful for different data mining tasks. We chose classification because a class labeling was known for the data at hand. Accuracy is evaluated using the sets of patterns as binary features for learning SVM classifiers as commonly done in pattern-based classification.

Table 3: Summary of Datatsets

Datasets	Events	Sequences	Classes
aslbu	36500	441	7
aslgt	178494	3493	40
auslan2	1800	200	10
pioneer	9766	160	3
skating	37186	530	7
unix	295008	11133	10

Table 4: Running time in seconds

Datasets	BIDE	SeqKimp	GoKrimp
aslbu	260.617	706.071	2.687
aslgt	379.894	1123.042	22.483
auslan2	1.547	1.688	0.265
unix	192.353	245.82	48.921
pioneer	121.356	278.147	0.672
skating	422.815	633.651	2.641

We use six different real-life datasets introduced in [14] to evaluate the proposed approaches. Each datasets is composed of a database of symbolic interval sequences with class labels. For our experiments the interval sequences are converted to 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 3. All the benchmark datasets are available for download on request at the website [1]. All code was written in Java. Evaluation was done in a 4 x 2.4 Ghz, 4 GB of RAM, Fedora 10 / 64-bit station.

6.1 Compressibility

We measure the compressibility of the algorithms by using their $Top-K$ patterns as dictionaries for encoding the data. For the SeqKrimp and the GoKrimp algorithms the compression benefits could be calculated as the sum of the compression benefit returned after each greedy step. For Closed patterns, compression benefit is calculated according to the greedy encoding scheme we are using for SeqKrimp. We perform this experiment by fixing *minsup* to the smallest values in the corresponding experiments shown in Figure 3. Compression benefits is measured as the number of memory cells we save up when encoding the original data with the help of the patterns in the dictionary. All single-event patterns are filtered out as they do not give any compression benefit.

Figure 1 shows the obtained results in six different datasets. As we expect, in most of the data sets, SeqKrimp and GoKrimp are able to find better compressing patterns than BIDE. Especially, in the large

[1] http://www.timeseriesknowledgemining.org

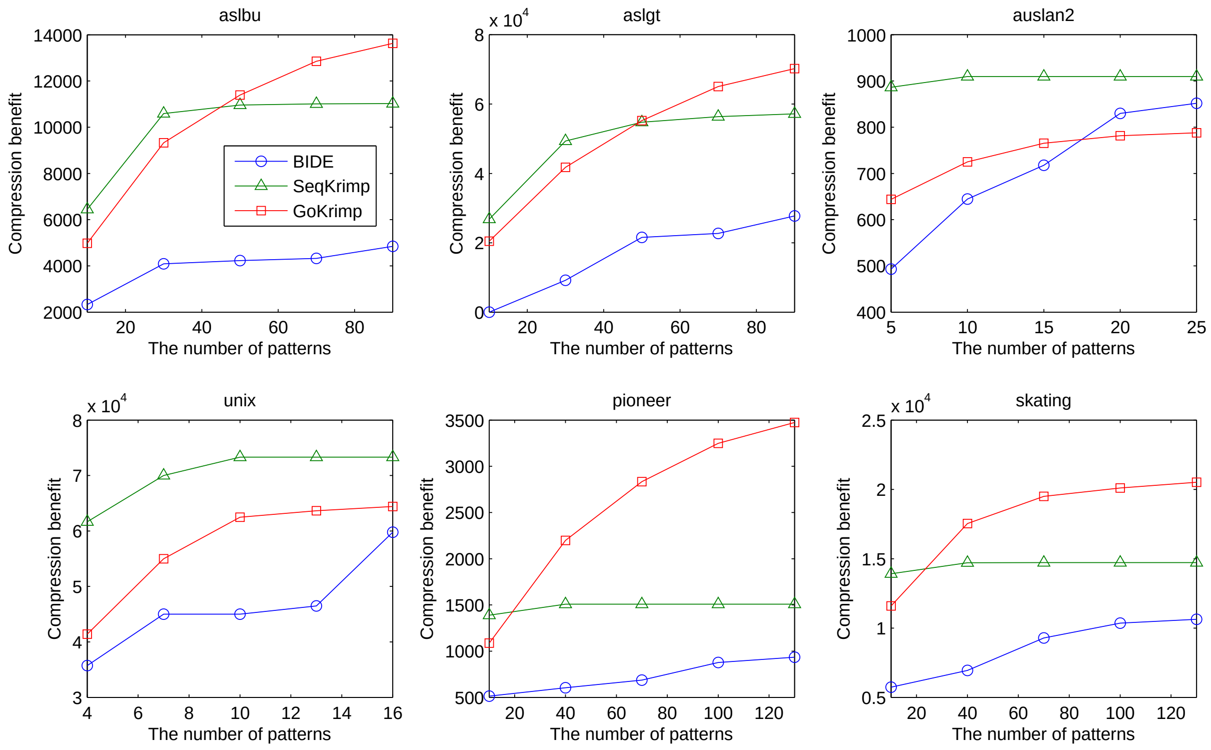

Figure 1: Compressibility (higher is better) of three algorithms when *minsup* is fixed and K is varied. SeqKrim and GoKrimp outperform BIDE (closed patterns). SeqKrimp is better than GoKrimp when K is small and becomes worse when K is set higher.

datasets such as *aslgt, unix* and *skating* the differences between SeqKrimp, GoKrimp and BIDE are very significant. Another important property of SeqKrimp algorithm could be observed clearly in Figure 1. That is, SeqKrimp algorithm gives highly compressing patterns when K is small. However, when K becomes larger SeqKrimp seems to-be less effective and it is outperformed by GoKrimp in several datasets including aslbu, aslgt, pioneer and skating.

6.2 Running Time Running time of each algorithm is measured by repeating the experiment in subsection 6.1. The running time is not much varied when K is changed, therefore for brevity, we only show the results when K is fixed to 20. The experimental result is illustrated in table 4.

As we can see in this table, algorithm SeqKrimp is always slower than algorithm BIDE because it needs an extra procedure to select compressing patterns from the set of candidates returned by BIDE algorithm. GoKrimp was named by the observation that it is 1-2 orders of magnitude faster than SeqKrimp or BIDE, giving results 'to go' when in a hurry.

6.3 Classification Accuracy In order to measure the predictive power of the sets of compressing patterns we consider each of them as a feature set for learning SVM classifiers. The Java implementation of the libsvm [2] library is chosen for our experiments because libsvm has a useful default setting which automatically selects optimal parameters for SVM classifiers. We have played with different SVM classifiers in the libsvm library before ending up with linear SVM because it outperforms the non-linear SVM classifiers in our specific datasets. In order to have stable comparison results, 10-fold cross-validation is performed with 4 runs. Single-event patterns are not filtered out as some of them are predictive even they do not give any compression benefit.

There are two different approaches to create the feature sets from the sets of compressing patterns. The unsupervised feature set is selected by extracting the *Top-K* patterns from the entire training data without considering the class labels of the data records. On the other hands, the supervised feature set is created by first dividing the training data into partitions, each of which contains only the training records belonging to one class. Subsequently, the *Top-K* compressing

[2] www.csie.ntu.edu.tw/∼cjlin/libsvm/

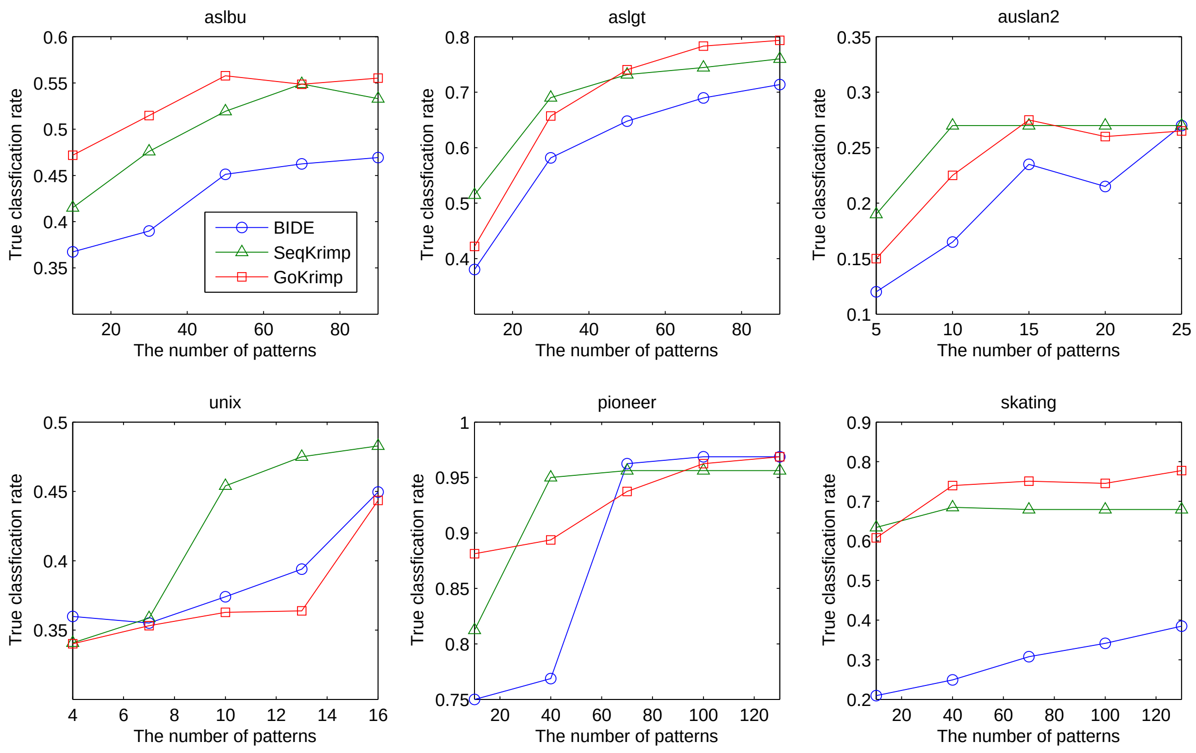

Figure 2: Classification accuracy of unsupervised feature selection when *minsup* is fixed and K is varied. Seqkrimp and GoKrimp outperform BIDE (closed patterns)

patterns are extracted from each partition and merged into a single set of features. In each of the following subsections, the experimental results of every feature sets created as aforementioned are discussed separately.

6.3.1 Unsupervised features When the class labels of the training data are completely available, we may prefer supervised feature selection to unsupervised one. This is actually the case in the six datasets used in our experiments. However, it is important to keep in mind that the inherent property of exploratory pattern mining is that patterns are discovered in an unsupervised manner. Therefore, it is interesting to see how predictive the sets of patterns are when they are discovered independently from class labels information.

Two parameters needed to-be set in our experiment. The first one is the minimum support value *minsup* and the second one is the maximum number of patterns K used to create the feature sets. All three algorithms in our experiments are dependent on K, while only two of them including the SeqKrimp and BIDE algorithms are sensitive to the value of *minsup*. We consider both cases, by fixing one parameter and varying the other. In the first experiment we fix *minsup* to the same value we use in subsection 6.1 for each dataset and vary K.

The accuracy results in terms of the true classification rates are plotted in Figure 2. We can see that SeqKrimp and GoKrimp are both better than the BIDE algorithm. They behave slightly differently depending on K. In particular, GoKrimp is worse than SeqKrimp for small K but becomes better than SeqKrimp for larger K. This fact suggests that the patterns at the very top positions in the *Top-K* list extracted by SeqKrimp are very predictive. However, the ones at the below positions of the *Top-K* list are less predictive. This result is consistent with the experiment in subsection 6.1 which suggests that there might be correlation between compressibility and classification accuracy.

In Figure 3 we plot results of runs where $K = 20$ and *minsup* is varied. Since GoKrimp is independent from *minsup* it is a straight line. We can see that SeqKrimp and GoKrimp are both better than BIDE algorithm. when minsup is set higher SeqKrimp outperforms GoKrimp. This is another advantage of GoKrimp because it does not depend on *minsup* set the value for which is not trivial in many applications.

In summary, for unsupervised feature selection, the Krimp-based approaches outperform the approach using closed sequential patterns. This shows that useful patterns are found by using MDL-based approaches, in line with prior research [1].

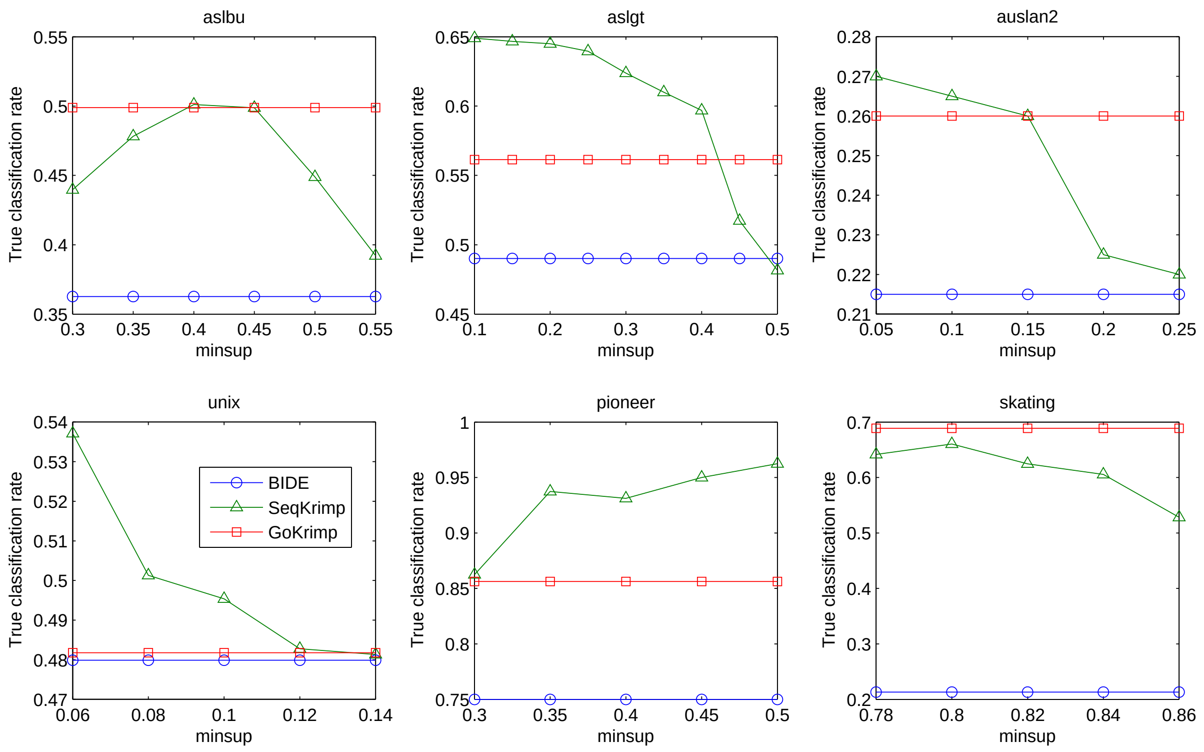

Figure 3: Classification accuracy of unsupervised feature selection when *minsup* is varied and $K = 20$.

6.3.2 Supervised features In this subsection, we repeat the experiments in subsection 6.3.1 with the same parameter settings. The only difference between this subsection and the prior one is that the training data is split into partitions each of which corresponds to one class. Subsequently, for each partition the *Top-K* patterns are extracted and merged with the other partition's patterns to form a single feature set for learning the SVM classifier.

The experimental results are shown in Figure 4 where the classification accuracy is plotted for six datasets. Unlike the unsupervised feature selection case, three algorithms behave very similarly here. A possible explanation of this phenomenon is that the number of features here is larger than the number of features in the prior experiments, allowing SVM classifier to be more accurate. Besides, the features extracted by BIDE are no longer frequent across the class but are frequent only in the each specific partition. As a result, these patterns maybe more predictive than the closed patterns extracted from the entire training data. We also discover that for the unix and pioneer dataset, frequent singletons are very predictive. This may explain the reason why BIDE performs remarkably well in these datasets.

The differences between the experimental results of unsupervised and supervised features selection suggest that compressing patterns are more appropriate for applications in which unsupervised feature extraction are required. In applications where supervised features selection is possible, compressing patterns does not provide much benefit in terms of predictive performance compared to the traditional frequent patterns.

7 Related work

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

The second direction concerns with statistical hypothesis testing in which data must be 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 pass the test it is considered significant and interesting. For example, A. Gionis et al. [19, 30] 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 S. Hanhijärvi et al. [21] and R. Milo et al. [20]. In those works, random graphs with prescribed degree distribution are generated, and significance of a subgraph is

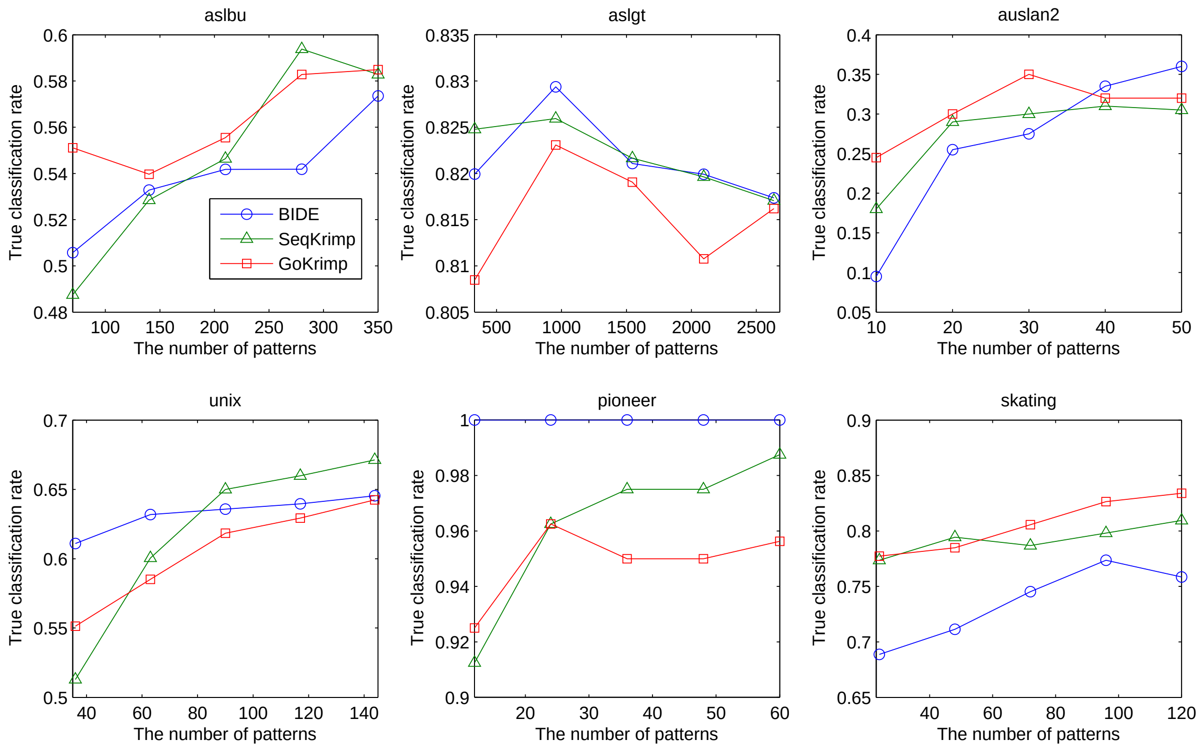

Figure 4: Classification accuracy of supervised feature selection when *minsup* is fixed and K varied. Three algorithms behave similarly, the significance of BIDE in pioneer and unix maybe due to the fact that singletons are very predictive in these datasets

estimated on the set of random generated graphs. Similar approach has also been applied to find interesting motifs in time-series data by N. Castro et al. [22].

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.

Another alternative research direction looks for 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 [1] and direct mining descriptive patterns algorithm [29] for itemset data and the algorithms for graph data [23, 24]. The usefulness of compressing patterns was demonstrated in various applications such as classification [1], components identifying [3] and change detection [4].

The idea of using data compression for data mining was first proposed by R. Cilibrasi et al. [28] for data clustering problem. This idea was also explored by Keogh et al. [26], who propose to use compressibility as a measure of distance between two sequences. They empirically show that by using this measure for classification problem, they are able to avoid setting complicated parameters, which is not trivial in many data mining tasks, while obtaining promising classification results. Another related work by C. Faloutsos et al. [27] suggests that there is a connection between data mining and Kolmogorov complexity. The connection is explained informally there, this notion quickly becomes the central idea of 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. To the best of our knowledge, the MDL principle has not been applied before to sequence data with the goal of finding compressing patterns. In addition to that, our work provides an extra study on the complexity of the MDL-based patterns mining problem which is absent in many works on this topic.

8 Conclusions and future work

In this work we have explored the mining of sequential patterns that compress the data well utilizing to MDL principle. A key contribution is our encoding scheme targeted at sequence data. We have shown that mining the most compression 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 show that the most compressing patterns are less redundant and better than the frequent closed patterns as feature sets for SVM classifiers when both are obtained in an unsupervised setting. As for itemset data it is likely that compression patterns are also useful for other data mining tasks where class labels are unavailable or rare, such as change detection or outlier detection. Future work will include further improvements to the mining algorithm using ideas from compression but keeping the focus on usefulness for data mining.

Acknowledgements

This work is done when one of the authors was visiting Siemens Corporate Research, a division of Siemens Corporation, in Princeton, NJ. The work is also funded by the NWO project Mining Complex Pattern in Streams (COMPASS). We would like to thank all the anonymous reviewers for the useful comments which help improving our work significantly.

References

[1] Vreeken, J., van Leeuwen, M. and Siebes, A. Krimp: Mining Itemsets that Compress. Data Mining and Knowledge Discovery, vol.23(1), Springer, 2011.

[2] Vreeken, J. Making Pattern Mining Useful. ACM SIGKDD Explorations, vol.12(1), ACM Press, 2010.

[3] Matthijs van Leeuwen, Jilles Vreeken, Arno Siebes. Identifying the components. Data Mining and Knowledge Discovery 19(2): 176-193 (2009)

[4] Matthijs van Leeuwen, Arno Siebes. StreamKrimp: Detecting Change in Data Streams. ECML/PKDD (1) 2008: 672-687

[5] Nikolaj Tatti, Jilles Vreeken. Finding Good Itemsets by Packing Data. ICDM 2008: 588-597

[6] Tijl De Bie. Maximum entropy models and subjective interestingness: an application to tiles in binary databases. DMKD Journal 2011

[7] Tijl De Bie, Kleanthis-Nikolaos Kontonasios, Eirini Spyropoulou. A framework for mining interesting pattern sets. SIGKDD Explorations 12(2): 92-100 (2010)

[8] Jiawei Han. Mining Useful Patterns: My Evolutionary View. Keynote talk at the Mining Useful Patterns workshop KDD 2010

[9] J. Wang and J. Han. BIDE: Efficient mining of frequent closed sequences. In *Proc. of the 20th Intl. Conf. on Data Engineering (ICDE)*, 79–90. IEEE Press, 2004.

[10] Peter Grünwald. The Minimum Description Length Principle. The MIT Press 2007

[11] Jian Pei, Jiawei Han, Mortazavi-Asl, B. Jianyong Wang Pinto, H. Qiming Chen Dayal, U. Mei-Chun Hsu. Mining sequential patterns by pattern-growth: the PrefixSpan approach. TKDE 2004

[12] F. Mörchen. Unsupervised pattern mining from symbolic temporal data. *SIGKDD Explor. Newsl.*, 2007.

[13] Dmitriy Fradkin and Fabian Moerchen. Margin-Closed Frequent Sequential Pattern Mining. Workshop on Mining Useful Patterns. KDD 2010

[14] Fabian Moerchen and Dmitriy Fradkin. Robust mining of time intervals with semi-interval partial order patterns, In Proc. SIAM SDM 2010, pp. 315-326

[15] Moerchen, F., Thies, M., Ultsch, A.: Efficient mining of all margin-closed itemsets with applications in temporal knowledge discovery and classification by compression, Knowledge and Information Systems 2010

[16] Floris Geerts , Bart Goethals , Taneli Mielikainen. Tiling Databases. Discovery Science 2004

[17] Christoph Ambuhl, Monaldo Mastrolilli, and Ola Svensson. Inapproximability Results for Maximum Edge Biclique, Minimum Linear Arrangement, and Sparsest Cut. SIAM J. on Computing volume 40 2011

[18] D.A. Huffman. A Method for the Construction of Minimum-Redundancy Codes. Proceedings of the I.R.E., September 1952, pp 1098-1102.

[19] Aristides Gionis, Heikki Mannila, Taneli Mielikäinen, Panayiotis Tsaparas. Assessing data mining results via swap randomization. TKDD 1(3): (2007)

[20] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, U. Alon. Network Motifs: Simple Building Blocks of Complex Networks. Science, Vol. 298, No. 5594. 2002

[21] Sami Hanhijärvi, Gemma C. Garriga, Kai Puolamäki. Randomization Techniques for Graphs. SDM 2009

[22] Nuno Castro, Paulo J. Azevedo. Time Series Motifs Statistical Significance. SDM 2011: 687-698

[23] Lawrence B. Holder, Diane J. Cook, Surnjani Djoko. Substucture Discovery in the SUBDUE System. KDD Workshop 1994: 169-180

[24] Deepayan Chakrabarti, Spiros Papadimitriou, Dharmendra S. Modha, Christos Faloutsos: Fully automatic cross-associations. KDD 2004: 79-88

[25] James A. Storer. Data compression via textual substitution Journal of the ACM (JACM) 1982

[26] Eamonn J. Keogh, Stefano Lonardi, Chotirat Ann Ratanamahatana, Li Wei, Sang-Hee Lee, John Handley. Compression-based data mining of sequential data. Data Mining Knowledge Discovery 14(1) (2007)

[27] Christos Faloutsos, Vasileios Megalooikonomou. On data mining, compression, and Kolmogorov complexity. Data Mining Knowledge Discovery 15(1) (2007)

[28] R. Cilibrasi, P.M.B. Vitanyi, Clustering by compression, IEEE Transaction Information Theory, 51:4 2005

[29] Smets, K. and Vreeken, J. Slim: Directly Mining Descriptive Patterns. SIAM SDM 2012.

[30] Miettinen, P.; Mielikainen, T.; Gionis, A.; Das, G.; Mannila, H.; The Discrete Basis Problem Knowledge and Data Engineering, IEEE Transactions on, 2008