Characterising the Difference and the Norm between Sequence Databases

Frauke Hinrichs and Jilles Vreeken

Max-Planck Institute for Informatics and Saarland University, Germany
{hinrichs,jilles}@mpi-inf.mpg.de

Abstract. In pattern set mining we are after a small set of patterns that together are characteristic for the data at hand. In this paper we consider the problem of characterizing not one, but a set of sequence databases, such as a collection of articles or the chapters of a book. Our main objective is to find a set of patterns that captures the individual features of each database, while also finding shared characteristics of any subset of the data we are interested in.
We formulate this problem in terms of MDL, and propose SqsNorm, an efficient algorithm to extract high quality models directly from data. We devise a heuristic to quickly find and evaluate candidates, as well as a model that takes shared *and* singular patterns into account.
Experiments on synthetic data confirm that SqsNorm ably reconstructs the ground truth model. Experiments on text data, including a set of speeches, several book chapters, and a collection of songs, show that SqsNorm discovers informative, non-redundant and easily interpretable pattern sets that give clear insight in the data.

1 Introduction

In this paper we propose SqsNorm, a method to characterise the difference and norm between multiple sequence databases in terms of sequential patterns. To make this less abstract, we will consider the case of analysing fairy tales. Assume we pick a set of fairy tales and analyze it with the goal of finding expressions that are typical for one or more of these stories. For example, most of them will have a happy ending. Some might even employ the catchy "*and they lived happily ever after*". We are likely to find a healthy share of *king and queen*s, *beautiful daughter*s and *evil stepmother*s. We will also notice expressions (or patterns, if you will) that have their grand entrance only once. Imagine candidates like *Snow White* or the *Seven Little Kids*, who are both very prominent in their respective tales but have no other appearances elsewhere.

The goal of SqsNorm is to discover exactly such a result. We want to find expressions that characterize a chosen subset of fairy tales, as well as patterns that are typical for only one story. It is worth mentioning that we had quite the head start, since we have an intuition of things like a "common expression" and an algorithm does not. Introducing a frequency threshold would lead to a summary far longer than the original data, as well as many unwanted redundancies.

This effect stems from the fact that every part of a frequent pattern is frequent, and every pattern has exponentially many parts, and is also called the "pattern explosion". Nobody wants a summary that is longer than the object that was originally summarized, and at this point, it would be shorter and cheaper to just read the fairy tale itself and be done with it. To avoid the pitfall of the pattern explosion, we will make use of the Minimum Description Length (MDL) principle to guide us to small, succinct, and non-redundant descriptions. The general idea is that if there is regularity in the data, anything other than a truly random sequence of symbols, then we can make use of this to *compress* our data. Regularity is equated with *information*.

There exist algorithms, such as SQS [15], and ISM [5], to discover good summaries of a given *single* event sequence database. There does exist an algorithm for jointly summarising multiple databases, called DIFFNORM, but as it only considers transaction data it cannot tell the difference between *the wolf eats grandmother* and *grandmother eats the wolf*. In this paper, we draw inspiration from both SQS and DIFFNORM, and propose SQSNORM, to efficiently discover meaningful pattern sets that joinly describe the difference and the norm between multiple sequence databases.

This paper is structured as usual. After quickly covering notation and introducing the main concepts, we will specify how to encode data and model for our system. We proceed to discuss finding good pattern candidates, and finally propose our algorithm SQSNORM. Through extensive evaluation we show that SQSNORM discovers good results on both synthetic and real world data, including some interesting results found when analyzing the inaugural addresses of American presidents (arguably a close enough alternative to the fairy tale example) and some of the works of J.R.R Tolkien.

2 Preliminaries

In this section we introduce the notation we will use throughout this paper, give a brief primer to the Minimum Description Length principle, and a short introduction to the SQS algorithm.

2.1 Basic Notation

The data we are working with is a bag $\mathcal{D}$ of $d = |\mathcal{D}|$ databases. A single database $D \in \mathcal{D}$ consists of $|D|$ sequences S, and a sequence S consists of $|S|$ consecutive events e drawn from an alphabet $\mathcal{I}$. We write $||D||$ for the number of all events in D and naturally extend this to denote the number of all events in all databases as $||\mathcal{D}||$. Let $supp(e|S)$ denote the number of times an event occurs in a sequence. We refer to this as the *support* of an event in a sequence, and accordingly define $supp(e|D) = \sum_{S \in D} supp(e|S)$ as the support of an event in a database, and $supp(e|\mathcal{D}) = \sum_{D \in \mathcal{D}} supp(e|D)$ as the support of an event in all databases. A pattern X is a sequence of $k = |X|$ events $e \in \mathcal{I}$. A pattern X occurs in a sequence S if some subsequence of S is equal to X. Note that we allow gaps of

(consecutive) singleton events between the events of a pattern. We call a tuple (i, j, X) a *window* for pattern X in a sequence. Indexes i and j specify start and end of the window. A window w is minimal if no other subsequence of w contains X.

2.2 The MDL principle

The Minimum Description Length (MDL) Principle is a guide to finding structure in data. It was first introduced by Rissanen [11] and attempts to achieve an incarnation of the (incomputable) Kolmogorov complexity [8]. The basic idea is that information corresponds to compression. The compressors used are called *models*, and after choosing a model class $\mathcal{M}$ for a problem setting, the models $M \in \mathcal{M}$ can be compared with respect to their ability to compress. The better a model compresses our data, the more information it incorporates. The principle tells us that if we want to find the best model M for some data D, we should pick the one that minimizes the sum

$$L(M) + L(D|M) \quad .$$

The first term describes the number of bits used for the encoded model, while $L(D|M)$ represents the number of bits required to encode the data using the information gained from M. Both terms should ideally keep each other in the balance, preventing overly complex, but expensive models, as well as cheap, but uninformative ones. Note that we require the compression to be lossless in order to fairly compare models of the same model class. Moreover, as in MDL we are interested in MDL theoretical complexity we are only interested in (optimal) code lengths, not actual instantiated code words.

2.3 Summarising a Single Database

Tatti and Vreeken [15] recently proposed the SQS algorithm (short for Summarising event seQuenceS) to efficiently discover good summaries of a single event sequence database, using the MDL principle to identify the optimal model. As we will build upon and extend the SQS algorithm, we will give a short introduction to its main aspects below.

The models SQS uses are called *code tables* (CT), which are lookup tables between patterns, and their associated codes. To ensure lossless encoding of any data over $\mathcal{I}$, all singleton events are always included.

The data is encoded as two separate streams C_p (pattern stream) and C_g (gap stream). The first one contains the encoded data, represented as a sequence of codes from the second column of the code table. The second stream is needed since we have no information on potential gaps when reading a pattern code. It is composed of *fill-codes* and *gap-codes*, and while decoding the data we read it in parallel to the pattern stream to fill in the gaps. For more detailed information about SQS we refer to the original paper [15].

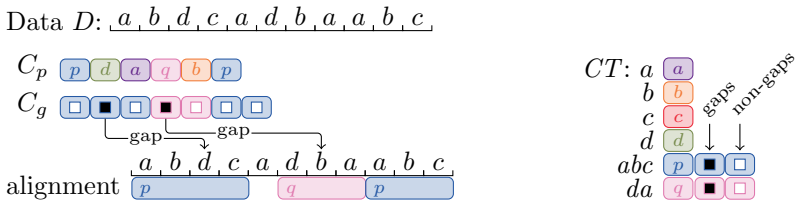

Fig. 1: Toy example of a possible encoding, using patterns abc and da

Encoding the data For a pattern X in the code table, let us write $usg(X)$ to denote how often the code of this entry is used in the encoding. Intuitively, the more often a code is used, the shorter it should be.

Shannon entropy [4] tells us that the shortest prefix code of a pattern corresponds to the negative logarithm of its probability, which in our case is its relative occurrence. We define the length of a pattern-code as follows:

$$L(code_p(X|CT)) = -\log\left(\frac{usg(X)}{\sum_{Y \in CT} usg(Y)}\right).$$

The length of the pattern stream C_p can now be defined intuitively. Since the complete data has to be covered with codes of either patterns or singleton events, the cost of a single code is added each time this code is used. $L(C_p|CT) = \sum_{X \in CT} usg(X)L(code_p(X))$. The same strategy can be applied for C_g — obviously, the gap code of a pattern should become shorter if gaps in this pattern are likely.

The overall encoded length of a database D given a code table CT is then

$$L(D \mid CT) = L_{\mathbb{N}}(|D|) + \sum_{S \in D} L_{\mathbb{N}}(|S|) + L(C_p \mid CT) + L(C_g \mid CT)$$

where we first encode the number of sequences in the database, as well as the length of each sequence, using the MDL optimal universal code for integers [12], which is defined for $n > 1$ as $L_{\mathbb{N}}(n) = \log^*(n) + \log(c_0)$, where $\log^*(n) = \log(n) + \log\log(n) + \ldots$. To make it a valid encoding, i.e. to satisfy the Kraft inequality, we set c_0 to 2.865064.

Finding the optimal alignment To encode a sequence database we need an *alignment* of which codes are used when. In particular, we want that alignment that encodes our data most succinctly. Whether a pattern should be used depends on its code length, however, which in turn depends on how often we use it. There are exponentially many alignments for a sequence database given a set of patterns, and hence exhaustive search is not possible. Tatti and Vreeken therefore propose a heuristic solution, which we call SqsAlign for the remainder of this paper. SqsAlign exploits the fact that fixing an alignment leads to computable code lengths and vice versa. In particular, given a set of patterns,

and initializing the usages to the supports of the patterns, SqsAlign alternates between finding the most succinct alignment given the code lengths these usages determine, and updating the code lengths accordingly, until convergence. As the starting values for the usages are estimates, the algorithm will only find a local minimum, yet experiments show that in practice this allows for the discovery of good models. For more details about the Sqs aligning scheme we refer the reader to Algorithm 2 in the original paper [15].

Mining good models This leaves us with the question of how to discover good models. Sqs does so in a bottom-up fashion, starting with the singleton only code table ST, and then iteratively searching for that candidate pattern X that maximally improves the compression, until we cannot find no candidate that improves the score and we halt. As candidates, it can either consider a given pre-mined collection of serial episodes, or, mine good candidates on the fly by considering combinations PQ of patterns $P, Q \in CT$ of already in the code table. Evaluating the quality of every candidate is expensive, and therefore Sqs rather first estimates the quality of a candidate, and evaluates the candidates in estimated order of gain in compression.

Limitations of Sqs for our purpose Although Sqs discovers good summaries for single databases, it does not suffice to jointly summarize multiple databases; when we run it on the concatenation of all databases, it will likely forego local detail, whereas when we run it on each of the individual databases it will not be able to pick up locally infrequent yet globally important patterns. Ignoring this, even concatenating these models will be a bad summary of the whole; as Sqs is a heuristic it may be that two very similar databases might lead to similar scores, yet very dissimilar pattern sets, and hence their concatenation could produce yet another different result. This would take away our basis of reasoning: instead of finding one set of patterns that describes well the *entire* set of databases, we would end up with multiple (possibly unrelated) pattern sets that each compress some subset of the data.

Our approach is hence to instead find patterns from a unified set of candidates, assuring consistent and comparable results.

3 Theory

Given the preliminaries above, we can now proceed to formalize our problem. To this end we will first specify our model class $\mathcal{M}$. A single code table, like for Sqs, will not suffice in the setting of multiple databases. In our case, a model $M \in \mathcal{M}$ will be a set $\mathcal{S}$ of pattern sets S_j, each S_j characterizing one of the database subsets the user is interested in. We will use the original Sqs encoding to obtain $L(D_i \mid CT_i)$, which means that the receiver must be able to reconstruct code tables $CT_1, \ldots, CT_d$ from model M to decode each database.

Let $\mathcal{D}$ be a set of d sequence databases over alphabet $\mathcal{I}$. Furthermore, let $\mathcal{J} = \{1, \ldots, d\}$ be a set of indexes. Any index set $j \subset \mathcal{J}$ can then be identified with

a set of databases $\{D_i \in \mathcal{D} \mid i \in j\}$. On receiving a set U of index sets as input, our goal is to find a model M that compresses $\mathcal{D}$. Specifically, for each of the $j \in U$ we want to find a set of patterns S_j that succinctly describe the subset of databases j is associated with.

Encoded length of the model

We now proceed to describe all the information necessary to encode the pattern sets S_j and the additional information needed to build a code table CT_i for each D_i. To encode a pattern set S_j, we first encode the number of patterns it contains using the universal code for integers introduced in the previous chapter. For each pattern in S_j we encode its length as well as the events of the pattern. Patterns are encoded like in SQS, with the singleton only code table over the concatenated databases, ST_Ω, that assigns optimal Shannon codes to all singletons according to their global support in $\mathcal{D}$.

$$L(S_j) = L_{\mathbb{N}}(|S_j + 1|) + \sum_{X \in S_j} \Big(L_{\mathbb{N}}(|X|) + \sum_{x \in X} L(\mathit{code}_p(x \mid ST_\Omega)) \Big)$$

The final cost for all the pattern sets then amounts to the sum of the costs of all the individual pattern sets.

With this information it is already possible to obtain the first column of the code tables CT_i. Let $\pi_i(\mathcal{S}) = \{S_j \in \mathcal{S} \mid i \in j\}$ denote all pattern sets that contain relevant non singleton patterns for a database D_i, and $\mathcal{P}_i = \bigcup \pi_i(\mathcal{S})$ the set of patterns needed to encode D_i. The entries of the first column of a code table CT_i then consist of the patterns in $\mathcal{P}_i$, together with the singletons to ensure lossless encoding.

We still need to encode the information for the pattern, gap, and fill codes per database to make this scheme work. Let $\mathit{usg}(\mathcal{P}_i)$ denote the sum of all usages of patterns in $\mathcal{P}_i$. The length of the code columns $\mathcal{C}_i$ for code table CT_i for database D_i is then defined as

$$\begin{aligned} L(\mathcal{C}_i) = {} & L_{\mathbb{N}}(||D_i||) + \log \binom{||D_i|| + |\mathcal{I}| - 1}{|\mathcal{I}| - 1} \\ & + L_{\mathbb{N}}(\mathit{usg}(\mathcal{P}_i) + 1) + \log \binom{\mathit{usg}(\mathcal{P}_i) + |\mathcal{P}_i| - 1}{|\mathcal{P}_i| - 1} \\ & + \sum_{X \in \mathcal{P}_i} L_{\mathbb{N}}(\mathit{gaps}(X) + 1) \quad , \end{aligned}$$

where the first two terms encode the support of each singleton in a database. This is done using a weak number composition. The second part uses the same technique to encode the usages for each pattern. Lastly, we encode the number of gaps for each pattern to obtain the gap codes. The fill code of a pattern can be computed knowing its length and usage, which we do at this point. Having defined the encoding for both the pattern sets and the code tables, the final cost for our model can now be computed as

$$L(M) = L_{\mathbb{N}}(|\mathcal{I}|) + L(\mathcal{S}) + \sum_{i \in \mathcal{J}} L(\mathcal{C}_i) \ .$$

For some database D_i, it is now trivially possible to extract the corresponding code table CT_i from our model. For the leftmost column, we add the singletons as well as the patterns in $\mathcal{P}_i$ which we know from $\mathcal{S}$. Lastly, we fill in the codes with the information $\mathcal{C}_i$ provides.

Encoded length of the data

Given a code table CT_i for a database D_i, we can simply re-use the encoding scheme from SQS. The pattern codes $code_p(X \mid CT_i)$ and gap codes $code_{gap}(X \mid CT_i)$ and $code_{fill}(X \mid CT_i)$ for a database D_i correspond to the respective pattern X in CT_i. Each database is encoded as two streams C_p and C_g, with

$$L(C_p|CT_i) = \sum_{X \in CT_i} usg(X) L(code_p(X|CT_i)) \qquad \text{, and}$$

$$L(C_g|CT_i) = \sum_{\substack{X \in CT_i \\ |X|>1}} gaps(X) L(code_{gap}(X|CT_i)) + fills(X) L(code_{fill}(X|CT_i)) \ .$$

The cost for a database D_i then amounts to

$$L(D_i \mid CT_i) = L_{\mathbb{N}}(|D_i|) + \sum_{S \in D_i} L_{\mathbb{N}}(|S|) + L(C_p \mid CT_i) + L(C_g \mid CT_i) \ .$$

$L(\mathcal{D})$ is then obtained by adding up the codes for each $D_i \in \mathcal{D}$. This score enables us to evaluate how well different pattern sets compress our data, and whether a candidate should be added to the model.

Problem definition

Having formalized our score, we can formally now state the problem we consider.

Minimal Serial Episode Composition Problem *Let $\mathcal{I}$ be a finite alphabet of events, $\mathcal{D}$ a bag of sequence databases over $\mathcal{I}$, and U a set of index sets over $\mathcal{D}$. Further, let $\mathcal{F}$ be the set of all possible sequences over $\mathcal{I}$ of length at most $\max_{S \in \mathcal{D}} |S|$. Find that set $\mathcal{S} \in \mathcal{P}(F)^{|U|}$ of sets of serial episodes that minimizes*

$$L(\mathcal{S}) + L(\mathcal{D} \mid \mathcal{S}) \ .$$

Regarding the complexity of this problem, we note that summarising a single sequence database is already hard: the search space is exponential, and there is no easily exploitable structure [15]. Unfortunately, our problem setting does not make it easier. We now have to consider all possible episodes in $\mathcal{D}$, and all possible ways to divide them up into $|U|$ many pattern sets. Exhaustively searching for the optimal model is not feasible, and there is again no exploitable structure in the search space. We therefore propose an algorithm that finds good pattern sets on a heuristic basis.

4 Algorithm

In the previous section we introduced MDL scores for model and data and defined the problem we are facing. Having seen that an exact computation of the optimal model is not feasible, we propose SqsNorm, a heuristic solution that allows us to mine good models directly from data.

Estimating candidates We denote the number of bits needed to encode database D_i with code table CT_i as $L(D_i, CT_i)$. Let us write $\Delta L(D_i, CT_i \oplus X)$ for the gain of adding a pattern X to the code table of a database D_i.

$$\Delta L(D_i, CT_i \oplus X) = L(D_i, CT_i) - L(D_i, CT_i \oplus X)$$

To compute this difference we use the Sqs encoding and compare the scores. We have seen in the previous sections that Sqs estimates the gain of a candidate and chooses the best extension PX for a pattern P based on this estimate.

We propose a slightly altered version of the estimate algorithm used in Sqs. Instead of returning only the extension with the highest gain d_X it returns the set of all possible extensions for P together with their scores. Its outline is shown in Algorithm 1. This tweak allows us to compare scores of candidates over several

Algorithm 1: Estimates(D, A, P)

input : Database D_i, current alignment A_i, pattern $P \in CT_i$
output: All possible candidates PX and their estimated gain d_X

```
for windows v of P in A_i do
    for windows w of X that occur after v do
        d_X ← updateGain(P, X);
return {(X, d_X) | X ∈ CT_i};
```

databases. Let us denote the estimated gain achieved by adding a pattern P somewhere in our model as $\hat{\Delta}L(D, \mathcal{S} \oplus P)$. Our goal is then to find candidates with a high $\hat{\Delta}L(D, \mathcal{S} \oplus P)$.

For any candidate pattern PX where P and X are drawn from $\mathcal{P} \cup \mathcal{I}$ we define a function $\hat{\Delta}L(D_i, CT_i \oplus PX)$ to describe the estimated gain when PX is added to the code table CT_i for D_i. This is not equivalent to d_X, because d_X is only defined for entries P and X that have at least one active window in D_i. We set this new score to zero for patterns that either are not relevant for the given database, or are expected to have a negative gain.

$$\hat{\Delta}L(D_i, CT_i \oplus PX) = \begin{cases} d_X \text{ if } (X, d_X) \in \textsc{Estimates}(D_i, A_i, P) \wedge d_X \geq 0 \\ 0 \quad \text{otherwise.} \end{cases}$$

Algorithm 2 gives an outline of how to find a good extension X for a given pattern P in the multiple database setting.

Algorithm 2: GetCandidate($\mathcal{D}, \mathcal{A}, P$)

input : Bag of d databases $\mathcal{D}$, set $\mathcal{A}$ of alignments $A_1, \ldots, A_d$, $P \in \mathcal{P}$
output: Pattern PX with $x \in \mathcal{P}$ and high $\hat{\Delta}L(D, \mathcal{S} \oplus PX)$

```
1 for D_i ∈ 𝒟 do
2     if P ∈ CT_i then
3         cands_i ← {Estimates(D_i, A_i, P)}
4 for X ∈ 𝒫 do
5     gain_X ← Σ cands_i( ΔL̂(D_i, CT_i ⊕ PX) )
6 return PX with highest gain_X;
```

For simplicity, we assume that a pattern will be added to all databases where it has a positive gain d_X, even though an index set representing these databases might not even exist in U. The task of choosing the right S_j for a pattern is addressed after the estimating process.

The SqsNorm Algorithm Since we can now create a list of candidates for our model, we proceed by computing their actual value. Since this requires us to call SqsAlign for each candidate and database, we only want to do an expensive computation if it is possible for the candidate to exist. Consider the following function δ:

$$\delta(D_i, P) = \begin{cases} 1 \text{ if } \exists P_1, P_2 \in CT_i : P = P_1 P_2 \\ 0 \text{ otherwise.} \end{cases}$$

We give the pseudo-code of SqsNorm as Algorithm 3. We first get the preliminary alignments by using SqsAlign without any patterns (line 1), and then start by creating a batch of candidates (3) that we order by their estimated gain. For each database in which a candidate may occur (5–6), we compute the code lengths to obtain their actual gain (7). The SetCover section is inherited from DiffNorm [2]: The set w contains the gain of a candidate PX for each index set j as a *weight* (10), and we choose to add PX to those S_j that maximize the gain (11–12). Since adding a new pattern to a code table might change the usages of other patterns, or even make them redundant (imagine *big bad* becoming dispensable as soon as *big bad wolf* has been found), each candidate check is followed by a pruning step (13), in which we re-evaluate all patterns in the model one by one, permanently removing them if this gives a gain in compression.

Complexity Last, we consider the computational complexity of SqsNorm. Since Estimate takes $\mathcal{O}(|P_i| + |\mathcal{I}| + ||D_i||)$ steps [15], GetCandidate has a complexity of $\mathcal{O}(d \times (|P_i| + |\mathcal{I}| + ||D_i||))$. The overall number of patterns $\mathcal{S}$ in our model can be bounded by the number $\mathcal{F}$ of frequent serial episodes over $\mathcal{I}$. $\mathcal{O}$(Sqs) for one database D_i is $\mathcal{O}(|P_i| \times ||D_i||)$ [15], and pruning takes

Algorithm 3: SqsNorm($\mathcal{D}, U$)

input : Bag of d databases $\mathcal{D}$, set of index sets U
output: Model $\mathcal{S}$= $\{S_1, \ldots, S_{|U|}\}$ that compresses $\mathcal{D}$ well

$\mathcal{P} \leftarrow \emptyset$; $\mathcal{S} \leftarrow \{\emptyset \mid j \in U\}$; $\mathcal{A} \leftarrow \{\text{Sqs}(D_i, \emptyset) \mid D_i \in \mathcal{D}\}$
do
  $F \leftarrow \{\text{GetCandidate}(\mathcal{D},\ \mathcal{A}\ ,\text{P}) \mid P \in \mathcal{P}, e \in \mathcal{I}\}$
  for $PX \in F$ ordered by $gain_X$ **do**
    for $D_i \in \mathcal{D}$ **do**
      if $\gamma(D_i, PX)$ **then**
        $gain_i \leftarrow \Delta L(D_i, C_i \oplus PX)$
      else
        $gain_i \leftarrow 0$
    $w \leftarrow \{\max(0, \sum_{i \in j} gain_i \mid j \in U\}$;
    $U' \leftarrow \text{WeightedSetCover}(\mathcal{J}, U, w)$;
    $\mathcal{S}' \leftarrow \mathcal{S}$ with PX added to every S_j with $j \in U'$ and positive w_j;
    $\mathcal{S} \leftarrow \text{Prune}(\mathcal{D}, \mathcal{S}, \mathcal{S}')$;
while $L(\mathcal{D}, \mathcal{S})$ decreases;
return $\mathcal{S}$;

$\mathcal{O}(|\mathcal{S}|^2 \times |\mathcal{D}|)$ time [2]. All in all, SqsNorm executes GetCandidate, SqsAlign and Prune at most $|\mathcal{F}|$ times, which results in a worst case complexity of $\mathcal{O}(|\mathcal{F}| \times \Big((|\mathcal{F}| \times ||\mathcal{D}||) + (|\mathcal{F}| + ||\mathcal{D}|| + d \times |\mathcal{I}|) + (|\mathcal{F}|^2 \times |\mathcal{D}|)\Big)) < \mathcal{O}(|\mathcal{F}|^3 \times ||\mathcal{D}||)$. In practice, SqsNorm converges quickly, only evaluating few carefully chosen candidates, calling SqsAlign only called where it makes sense to do so, and the model is pruned to have only relevant patterns in the model at all times—in our experiments SqsNorm takes up to a few minutes on reasonably sized data.

5 Related Work

Discovering interesting patterns from sequential data has a long history. Traditional approaches focus on mining all patterns that satisfy a local intererestingness constraint. In sequential data, however, there exist multiple reasonable definitions for support. Laxman et al. proposed to count the largest set of non-overlapping windows using finite state automata, tracking the frequencies of each episode with an automaton [7]. Their method discovers serial and parallel episodes. Mannila at al. propose counting minimal windows, starting with singletons and building episodes of increasing length [9]. The algorithm relies on the fact that a frequent episode implies that its sub-episodes are frequent and requires window size and minimal support as user parameters. Tatti et al. introduce *strict* episodes and define a subset relation between them, using directed

acyclic graphs to mine pattern candidates [14]. This approach also generates candidate episodes from already discovered patterns.

A related class of methods aims to discover those patterns that are prevalent in one class, and rare in the other [6], or to mine such discriminative patterns directly from data for classification purposes [3]. Quiniou et al. consider multiple text corpora with the goal of discovering typical linguistic patterns for each of them [10]. Our objective, on the other hand, is not to discriminate between databases nor to describe them independently, but to jointly characterize sets of databases the user is interested in.

In this paper we do not consider mining all patterns, but rather return a small set of patterns that together describe the given databases well. As far as pattern set mining is concerned, KRIMP [16] made the first step toward using MDL in market basket analysis, selecting a small set of patterns that compress a single database well out of a possibly huge set of frequent patterns. SLIM [13] later refined this approach, mining candidates directly from the data instead of depending on an input set of candidates.

In this paper we build upon SQS [15], an MDL-based heuristic to mine descriptive sets of serial episodes for a single sequence database. To encode the data, it uses look-up tables for patterns and codes, and considers candidates either from a pre-mined pool, or by iteratively considering combinations of patterns in the current model. While this approach works well for a single dataset, it cannot trivially be extended for multiple databases, since the heuristic nature of the algorithm would not produce consistent candidates for each database.

In a more recent approach, SQUISH enriches the concept of SQS with a broader pattern language [1]. In addition to serial episodes, SQUISH allows for *choice episodes*, where a position in an episode may be filled with one of a predefined choice of events. In addition, SQUISH adopts a greedy heuristic to find good covers of the data. It will be interesting to see to what extent these ideas can be carried over into SQSNORM.

DIFFNORM [2] is the other inspiration of this work. Its goal is to find difference and norm among itemset databases with the help of MDL, which we here extend to the setting of sequential data; unlike SQSNORM, DIFFNORM does not consider order, repetitions and possible gaps, and as a result neither its score nor search algorithm can be trivially applied, or adapted to sequential data.

6 Experiments

Next we proceed to empirically evaluate SQSNORM. To this end we consider both synthetic and real world data. As real world data, for interpretability of the results, we will consider text data. All experiments were run single-threaded on a quad-core Intel machine with 8GB of memory, running windows.

6.1 Synthetic data

All synthetic datasets feature $10,000$ events per database. Events are drawn from alphabet $\mathcal{I} = \{0, \ldots, 500\}$. Index set U includes pattern sets $S_1, \ldots, S_d$ for

Dataset	# L	# G	$L(\mathcal{D}, S_0)$	$L\%$	$\lvert\mathcal{S}\rvert$	=	$\cup$	?	time(s)
Random	0	0	461 180	100	0	0	0	0	34
Local	50	0	460 612	97	50	50	0	0	161
Global	0	10	460 540	97	10	10	0	0	111
Mixture	10	90	906 089	71	100	100	0	0	765

Table 1: Results on synthetic datasets. From left to right: number of locally (# L) and globally (# G) planted patterns, length using the empty model ($L(\mathcal{D}, S_0)$), compression rate ($L\% = \frac{L(\mathcal{D},\mathcal{S})}{L(\mathcal{D},\mathcal{S}_0)}$), total number of patterns ($|\mathcal{S}|$), exactly recovered patterns assigned to the correct S_j (=), concatenations of patterns ($\cup$), patterns that are unrelated to any planted patterns (?).

each individual database as well as a global pattern set S_Ω. In the following we describe four basic datasets we used to ascertain the most important properties of SqsNorm.

Random To make sure our algorithm does not pick up noise, we create the *Random* dataset, consisting of five databases. It contains no information, since events are picked uniformly at random.

Local For this dataset, we set up five databases and plant ten unique patterns per database (all in all 50 unique patterns: $(1, \ldots, 5)$ to $(246, \ldots, 250)$). In this case we would like to find all the planted patterns in their respective S_j, while S_Ω has to stay empty.

Global We then proceed by planting 10 global patterns in five databases. Patterns consist of randomly chosen events from $\mathcal{I}$, which allows for arbitrary patterns that might have repetitions or shared events. This time, the pattern should be identified as typical for all databases and put in S_Ω.

Mixture In the *Mixture* set we run SqsNorm on a set of 10 databases and plant a total of 100 patterns, 90 of which are global, while each database retains exactly one local pattern. Patterns are created exactly as for the *global* set.

Table 1 shows results for each of the introduced datasets. SqsNorm correctly identifies the planted patterns and assigns them to the pattern set that fits them without picking up noise. These results do not change even if gaps occur with a probability of up to 50%, which doubles the average pattern cover.

6.2 Real data

We now proceed to test SqsNorm on some real world datasets. We first explain the data structure and preprocessing methods and then present some results.

Addresses per Era First, we regard the *Addresses* dataset. It consists of the presidential inaugural addresses from George Washington to Donald Trump. The individual words are stemmed and stopwords are removed. Each word in

Dataset	$\lvert\mathcal{D}\rvert$	$\lvert\mathcal{I}\rvert$	$\lvert\lvert\mathcal{D}\rvert\rvert$	$\lvert\mathcal{J}\rvert$	$\lvert S_\Omega\rvert$	avg($\lvert S_j\rvert$)
Addresses/Era	67	5386	64035	8	14	21.5
Addresses	67	5386	64035	57	4	1.37
LOTR1-12	12	4060	44347	12	8	6
Hobbit	19	4021	45650	19	7	2.74
Songs	4	26	3223	4	13	7.75

Table 2: Statistics about the datasets used. From left to right: number of sequences, size of the alphabet, overall number of events, number of databases, size of S_Ω, the average number of patterns representing an individual database.

the resulting corpus is seen as an event, and we group the addresses into eight eras of American history, regarding each of these eras as a database and the individual addresses as sequences. Figure 2 shows some results for each era and S_Ω.

Addresses individual For comparison, we then treat each inaugural address as a database and run SqsNorm again. We would expect S_Ω to shrink, since 57 individual inaugural addresses have probably less patterns in common than entire eras from the previous experiment. Also, it would be natural for the individual S_j to become smaller, since a single address is not likely to have more than two or three recurring slogans. The experiments confirm this effect. S_Ω shrinks, and the average number of patterns per individual address goes down to about 1. This often corresponds to one special catchphrase or slogan used during the address. Interestingly, these slogans were often not found in the *Era* dataset, because it is too costly to call such a pattern era-typical when it only occurs during one speech. Some patterns that were only found on an address-wise run of SqsNorm are *deliver up* (Lincoln), *new spirit* (Carter), *put out [my] hand* (Bush) and *make America [. . .] again* (Trump).

LOTR1-12 To obtain more meaningful results, we analyze the first twelve chapters of J.R.R. Tolkien's *Lord of the Rings: The Fellowship of the Ring*. The text is stemmed and stopwords are removed. We regard individual chapters as sequence databases containing a single sequence. Quite correctly, SqsNorm find general themes like "Mr Bilbo" or "Sam Gamgee" in S_Ω. Characteristics of each chapter include chapter-bound events like Bilbo's *eleventy-first birthday*, and new concepts that are introduced and explained during a chapter such as *the one ring*. Figure 3 shows patterns found for selected chapters.

The Hobbit We then proceed to explore J.R.R Tolkien's book *The Hobbit* in the same way. Again, the patterns found conform to our expectations. Local patterns correspond to specific scenes or places (*dining room*, *birthday present*, *lake town*), while the main characters' names and concepts that weave through the entire story (*King under the Mountain, Mr Baggins, Fili Kili,. . . *) are found in S_Ω.

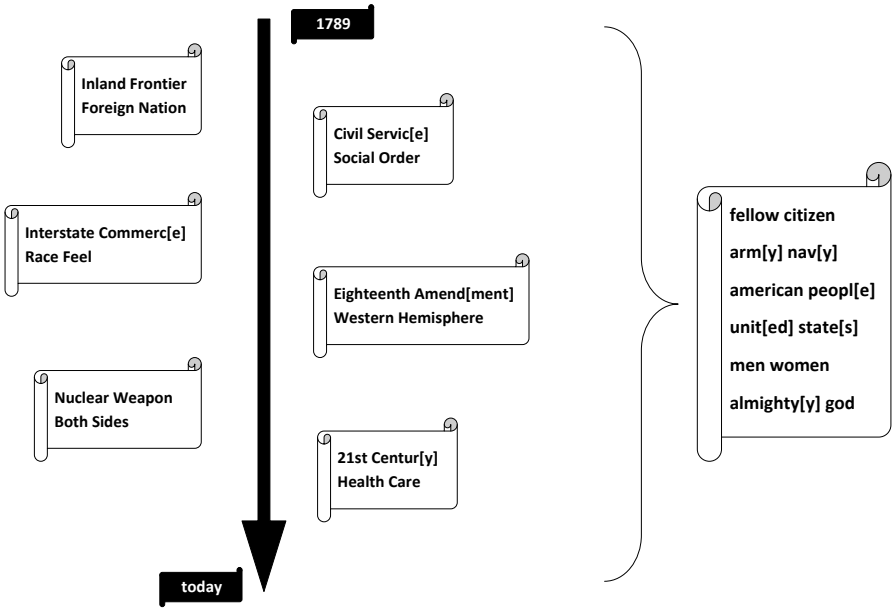

Fig. 2: Results of the *Addresses* dataset when divided by era. Left: patterns clustered by era they are typical for. Right: typical patterns for all eras

Songs To evaluate SqsNorm on a somewhat different scale, we finally test it on a collection of four songs taken from the top of the charts around Christmas. Each song is understood as a database containing a single sequence. Instead of comparing them word by word, we choose to analyze them letter by letter, with $\mathcal{I} = \{a, \ldots, z\}$ as our alphabet. This results on the one hand in global stopwords, like *and* and *this*, which do not contribute much interesting information. We find one non-stopword pattern in S_Ω: *Love* is a common pattern for all chosen songs. On the other hand we find long pieces of refrains (up to 32 letters) in the local pattern sets, often catching the most memorable bits of the song.

7 Discussion

The experiments show that SqsNorm correctly returns good sets of informative patterns. By analyzing synthetic datasets we could show that true patterns are correctly found up to very high gap levels. Furthermore, global patterns are correctly identified as such while noise is not picked up at all. Results on real datasets are interesting and plausible, and we discover universally used phrases and themes alike. By experimenting with the *Songs* dataset we have seen that SqsNorm can also be applied on a character-wise level. It picks up large parts of the songs' main refrains, as well as frequently used stopwords and even identifies *love* as a common theme they share.

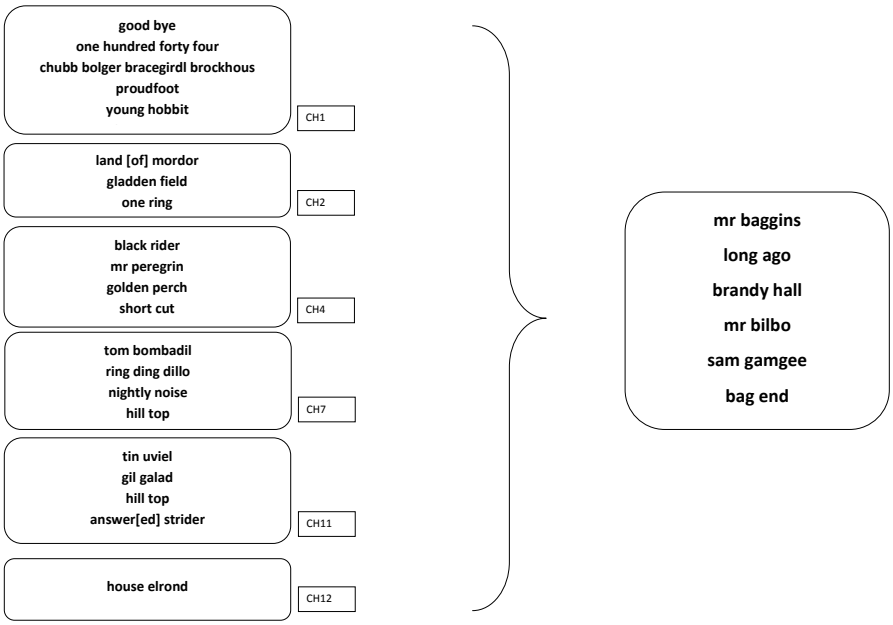

Fig. 3: Results of the *LOTR1-12* dataset. Left: Typical patterns for chosen chapters. Right: typical expressions for all chapters

Despite these very encouraging results, there are several possibilities to further improve SqsNorm. Candidate estimation can be trivially parallelized by concurrently evaluating candidates. Moreover, since estimates for each database are calculated independently, all of these computations can be parallelized on a database-level as well. Lastly, we are currently using prefix codes for our encoding. There is a more refined approach called prequential coding, which allows us to transmit the data without having to encode the usages of the patterns explicitly, and hence helps avoid any possible bias in the encoding of the model. The key idea of prequential coding is that, starting from a uniform distribution over the usages, we can update these counts with every code that we receive, and so rapidly approximate the true usage distribution. This has been successfully used in DiffNorm [2], and while our results show it is not necessary to obtain meaningful results, it will be interesting to see if these improve when we do.

8 Conclusion

In this paper we studied the problem of discovering difference and norm between sequence databases. In order to do so, we proposed SqsNorm, an algorithm that uses MDL to find sets of patterns that represent both local and global features of the data. Drawing inspiration from Sqs and DiffNorm, we defined an encoding for our data and proposed a way to mine good candidates directly from data. Evaluating SqsNorm, we have seen that our system works well in practice, does

not pick up noise and is fast. SqsNorm allows to extract potentially interesting information with regard to what is the norm, and what are the differences between sequences databases in general, and as we have shown, text data in particular. In the future we plan to further explore its application for exploratory analysis in computational linguistics.

Acknowledgements

The authors are supported by the Cluster of Excellence "Multimodal Computing and Interaction" within the Excellence Initiative of the German Federal Government.

References

1. A. Bhattacharyya and J. Vreeken. Efficiently summarising event sequences with rich interleaving patterns. In *SDM*. SIAM, 2017.
2. K. Budhathoki and J. Vreeken. The difference and the norm – characterising similarities and differences between databases. In *ECML PKDD*. Springer, 2015.
3. H. Cheng, X. Yan, J. Han, and P. S. Yu. Direct discriminative pattern mining for effective classification. In *ICDE*, pages 169–178, 2008.
4. T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley-Interscience New York, 2006.
5. J. Fowkes and C. Sutton. A subsequence interleaving model for sequential pattern mining. In *KDD*, 2016.
6. X. Ji, J. Bailey, and G. Dong. Mining minimal distinguishing subsequence patterns with gap constraints. *Knowl. Inf. Syst.*, 11(3):259–286, April 2007.
7. S. Laxman, P. S. Sastry, and K. P. Unnikrishnan. A fast algorithm for finding frequent episodes in event streams. In *KDD*, pages 410–419. ACM, 2007.
8. M. Li and P. Vitányi. *An Introduction to Kolmogorov Complexity and its Applications*. Springer, 1993.
9. H. Mannila, H. Toivonen, and A. Inkeri Verkamo. Discovery of frequent episodes in event sequences. *Data Min. Knowl. Discov.*, 1(3):259–289, January 1997.
10. S. Quiniou, P. Cellier, T. Charnois, and D. Legallois. What about sequential data mining techniques to identify linguistic patterns for stylistics? In *CICLing*, pages 166–177. Springer-Verlag, 2012.
11. J. Rissanen. Modeling by shortest data description. *Automatica*, 14(1):465–471, 1978.
12. J. Rissanen. A universal prior for integers and estimation by minimum description length. *Annals Stat.*, 11(2):416–431, 1983.
13. K. Smets and J. Vreeken. Slim: Directly mining descriptive patterns. In *SDM*, pages 236–247. SIAM, 2012.
14. N. Tatti and B. Cule. Mining closed episodes with simultaneous events. In *KDD*, pages 1172–1180, 2011.
15. N. Tatti and J. Vreeken. The long and the short of it: Summarizing event sequences with serial episodes. In *KDD*, pages 462–470. ACM, 2012.
16. J. Vreeken, M. van Leeuwen, and A. Siebes. Krimp: Mining itemsets that compress. *Data Min. Knowl. Disc.*, 23(1):169–214, 2011.