ProSecCo: Progressive Sequence Mining with Convergence Guarantees

Sacha Servan-Schreiber
Brown University
Dept. of Computer Science
115 Waterman St.
Providence, RI 02912, USA
aservans@cs.brown.edu

Matteo Riondato
Two Sigma Labs
100 Avenue of the Americas
New York, NY 10013, USA
matteo@twosigma.com

Emanuel Zgraggen
Brown University
Dept. of Computer Science
115 Waterman St.
Providence, RI 02912, USA
ez@cs.brown.edu

ABSTRACT

"Here growes the wine Pucinum, now called Prosecho, much celebrated by Pliny." – Fynes Moryson, *An Itinerary*, 1617.

We present ProSecCo, a progressive algorithm for mining the collection of frequent sequences from large transactional datasets: it processes the dataset in small blocks and outputs, after having analyzed each block, a high-quality approximation of the collection of frequent sequences. ProSecCo can be used for interactive data exploration: the intermediate results enable the user to make informed decisions as the computation proceeds.

The intermediate results have strong probabilistic approximation guarantees and the final output is the exact collection of frequent sequences. The correctness analysis uses the Vapnik-Chervonenkis (VC) dimension, a key concept from statistical learning theory.

The results of our experimental evaluation of ProSecCo on real and artificial datasets shows that it produces fast-converging high-quality results almost immediately. Its practical performances are even better than what is guaranteed by the theoretical analysis, and it can even be faster than non-progressive algorithms.

PVLDB Reference Format:
Sacha Servan-Schreiber, Matteo Riondato, and Emanuel Zgraggen. ProSecCo: Progressive Sequence Mining with Convergence Guarantees. *PVLDB*, 12(xxx): xxxx-yyyy, 2019.
DOI: https://doi.org/TBD

1. INTRODUCTION

One of the first steps of data analysis is *data exploration*. During this phase, the user performs a preliminary study of the dataset to get acquainted with the data and prepare for further deeper analysis. Systems for data explorations must be *interactive* in order to be useful: computer response times longer than one second negatively affect user productivity [14, 21]. In the specific case of data exploration, it has been shown that small (500ms [12]) as well as larger (6–12s [29]) delays between query and response significantly decrease the rate at which users discover insights. In other words, productive human-in-the-loop data exploration is only achievable when the part of the loop not involving the human requires less time to complete than the human needs to perform her task.

One way to achieve this level of interactivity is to output, as soon as possible, *intermediate results* to the query, and frequently update them as more data is processed.

The intermediate results must not mislead the user, or she will not be able to make informed decisions. To be trustworthy, intermediate results must have two important properties: *1)* they must be (probabilistically) *guaranteed to be high-quality approximations* of the final exact results, where the concept of approximation must be easy to interpret for the final user; and *2)* they must *converge* to the exact results quickly as they are updated, and correspond to the exact results of the query once all data has been processed.

Some data exploration tools, such as Vizdom [5], achieve interactivity by displaying intermediate results computed through *online aggregation* [8]. This technique allows to obtain intermediate results for relatively simple SQL queries, but does not currently support more complex knowledge discovery tasks that are a key part of data exploration.

Existing data mining algorithms are poor candidates for this phase of data analysis, as they can take many minutes to complete, therefore disrupting fluid user experiences. In this work we focus on the important task of *frequent sequence mining* [3, 17], which requires to find ordered lists of itemsets appearing in a large fraction of a dataset of transactions. Applications includes market basket analysis and web log analysis for recommendation systems. Our approach can be generalized to other pattern extraction tasks.

The bottom part of Figure 1 shows the lack of interactivity of existing frequent sequence mining algorithms. After having selected a dataset and a *minimum frequency threshold* to deem a sequence frequent, the user launches a *non-progressive* frequent sequence mining algorithm, such as PrefixSpan [17]. No response is given to the user until the algorithm has terminated, which may take many tens of seconds. Such a delay destroys the productivity of the data exploration session. New algorithms are needed to ensure that the human is involved in the loop of data analysis by giving them actionable information as frequently as possible.

Proceedings of the VLDB Endowment, Vol. 12, No. xxx

DOI: https://doi.org/TBD

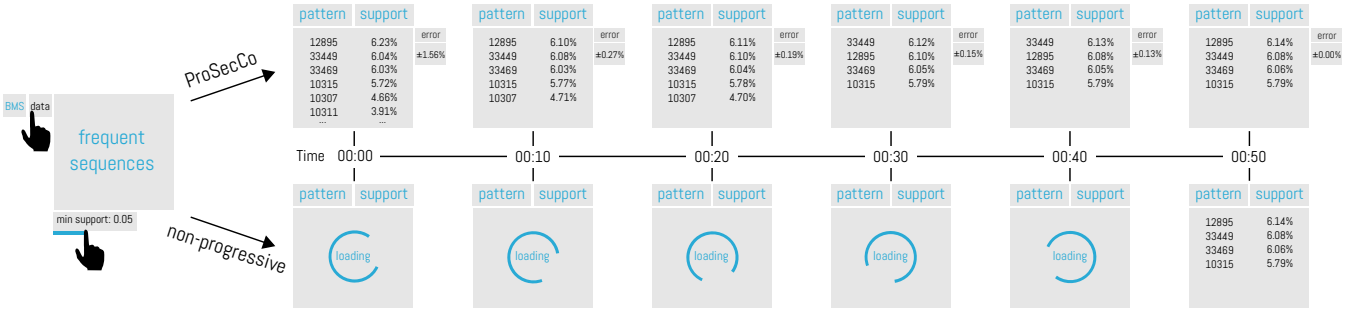


Figure 1: Illustration of an interactive data analysis tool where users can gesturally invoke a frequent sequences mining operation (left) by selecting a dataset and a minimum frequency threshold. The feedback displayed by the tool to the user varies greatly depending on whether a progressive or a non-progressive algorithm is used to compute the answer to such a query. In the case of a non-progressive algorithm (bottom) the tool shows a loading animation until the exact answer is computed after 40 seconds. With ProSecCo, the tool can show (top) progressively-refined results to the user immediately and at various points in time. Data and times for this example are taken from actual experiments.

Contributions. We present ProSecCo, a progressive frequent sequence mining algorithm with high-quality intermediate results, suitable for interactive data exploratory.

- ProSecCo periodically returns to the user high-quality approximations of the collection of interest (see the top part of Figure 1). This progressive behavior is achieved by analyzing the dataset incrementally in small blocks of user specified size. ProSecCo extracts a set of candidate frequent sequences from the first block by mining it at a lowered frequency threshold. ProSecCo can return the first set of results after less than a second, therefore keeping the user engaged in the data exploration process. The set of candidates is guaranteed to be *superset* of the exact collection of frequent sequences. It is progressively refined as more blocks are processed and corresponds to the exact collection of frequent sequences once the last block has been analyzed. We also present a variant ProSeK for extracting the *top-k* most frequent sequences.

- All the returned sets of candidate sequences come with strong probabilistic guarantees on its quality: each of them is an *ε-approximation* (see Definition 1) of the collection of frequent sequences. Such guarantees enable the user to decide whether to continue or stop the processing of additional blocks. Our analysis uses VC-dimension [25] and other fundamental sample-complexity results from statistical learning theory [11, 24]. This connection is novel for sequence mining. We show that the empirical VC-dimension of the task of frequent sequence mining is bounded above by a characteristic quantity of the dataset, which we call the *s-index* (see Definition 2), that can be computed in a streaming fashion as the blocks are read.

- We conducted an extensive experimental evaluation of ProSecCo on real and artificial datasets. The results show that ProSecCo produces approximations of the actual set of frequent sequences almost immediately, with even higher quality than our theoretical analysis suggests. It uses several orders of magnitude less memory when compared to PrefixSpan [17], and it is at times faster.

2. PRELIMINARIES

We now formally introduce the concepts and the basic results used throughout the paper.

2.1 Sequence mining

Let $\mathcal{I} = \{i_1, \ldots, i_n\}$ be a finite set. The elements of $\mathcal{I}$ are called *items* and non-empty subsets of $\mathcal{I}$ are known as *itemsets*. A *sequence* $\mathbf{s} = \langle S_1, S_2, \ldots, S_\ell \rangle$ is a *finite ordered list of itemsets*, with $S_i \subseteq \mathcal{I}$, $1 \leq i \leq \ell$. A sequence $\mathbf{a} = \langle A_1, A_2, \ldots, A_n \rangle$ is a *subsequence* of another sequence $\mathbf{b} = \langle B_1, B_2, \ldots, B_m \rangle$, denoted by $\mathbf{a} \sqsubseteq \mathbf{b}$, iff there exist integers $1 \leq j_1 < j_2 < \ldots j_n \leq m$ such that $A_1 \subseteq B_{j_1}, A_2 \subseteq B_{j_2}, \ldots, A_n \subseteq B_{j_n}$. If $\mathbf{a} \sqsubseteq \mathbf{b}$, we also say that $\mathbf{b}$ is a *super-sequence* of $\mathbf{a}$. The *length* $|\mathbf{s}|$ *of a sequence* $\mathbf{s} = \langle S_1, S_2, \ldots, S_\ell \rangle$ is the number of itemsets in it. The *item-length* $\|\mathbf{s}\|$ of $\mathbf{s}$ is the sum of the sizes of the itemsets in it, i.e.,

$$\|\mathbf{s}\| = \sum_{i=1}^{|\mathbf{s}|} |S_i|,$$

where the size $|S_i|$ of an itemset S_i is the number of items in it (e.g., $|\{a, b, c\}| = 3$).

A *dataset* $\mathcal{D}$ is a finite bag of sequences. When referring to them as members of the dataset, the elements of $\mathcal{D}$ are known as *transactions*. A sequence $\mathbf{s}$ *belongs* to a transaction $\tau \in \mathcal{D}$ iff $\mathbf{s}$ is a subsequence of τ i.e., $\mathbf{s} \sqsubseteq \tau$.

For any sequence $\mathbf{s}$, the *frequency* of $\mathbf{s}$ in $\mathcal{D}$ is the *fraction* of transactions of $\mathcal{D}$ to which $\mathbf{s}$ belong:

$$\mathrm{f}_{\mathcal{D}}(\mathbf{s}) = \frac{|\{\tau \in \mathcal{D} \ : \ \mathbf{s} \sqsubseteq \tau\}|}{|\mathcal{D}|} \ . \tag{1}$$

For example, suppose we have a dataset $\mathcal{D}$ with the following four transactions:[1]

$$\begin{array}{l} \langle \{a\}, \{a, b, c\}, \{a, c\}, \{d\}, \{c, f\} \rangle \\ \langle \{a, d\}, \{c\}, \{b, c\}, \{a, e\} \rangle \\ \langle \{e, f\}, \{a, b\}, \{d, f\}, \{c\}, \{b\} \rangle \\ \langle \{e\}, \{g\}, \{e, f\}, \{c\}, \{b\}, \{c\} \rangle \end{array}$$

The first transaction is a *sequence*

$$\mathbf{s} = \langle \{a\}, \{a, b, c\}, \{a, c\}, \{d\}, \{c, f\} \rangle$$

[1]Example from [17].

with five *itemsets*. Its item-length $\|\mathbf{s}\|$ is 9. While the sequence $\langle\{a\}\rangle$ occurs three times as a subsequence of $\mathbf{s}$, it is only counted once to compute the frequency of $\langle\{a\}\rangle$. Thus, the frequency of $\langle\{a\}\rangle$ in the above dataset is 1/2. The sequence $\langle\{a,b\},\{f\},\{d\}\rangle$ is *not* a subsequence of $\mathbf{s}$, because the order of the itemsets in the sequences matters.

Frequent sequences mining. Let $\mathbb{S}$ denote the set of all sequences built with itemsets containing items from $\mathcal{I}$. Given a *minimum frequency threshold* $\theta \in (0,1]$, the collection $\mathsf{FS}(\mathcal{D},\theta)$ of *frequent sequences in $\mathcal{D}$ w.r.t. θ* contains all and only the sequences with frequency at least θ in $\mathcal{D}$:

$$\mathsf{FS}(\mathcal{D},\theta) = \{\mathbf{s} \in \mathbb{S} \ : \ \mathsf{f}_{\mathcal{D}}(\mathbf{s}) \geq \theta\} \ .$$

In this work we make heavy use of *ε-approximations of* $\mathsf{FS}(\mathcal{D},\theta)$, for $\varepsilon \in (0,1)$. Formally, they are defined as follows.

Definition 1. *Let $\varepsilon \in (0,1)$. An* ε-approximation to $\mathsf{FS}(\mathcal{D},\theta)$ *is a set $\mathcal{B}$ of pairs $(\mathbf{s}, f_{\mathbf{s}})$, where $\mathbf{s} \in \mathbb{S}$ and $f_{\mathbf{s}} \in [0,1]$, with the following properties:*

1. *$\mathcal{B}$ contains a pair $(\mathbf{s}, f_{\mathbf{s}})$ for every $\mathbf{s} \in \mathsf{FS}(\mathcal{D},\theta)$;*
2. *$\mathcal{B}$ contains no pair $(\mathbf{s}, f_{\mathbf{s}})$ such that $\mathsf{f}_{\mathcal{D}}(\mathbf{s}) < \theta - \varepsilon$;*
3. *Every $(\mathbf{s}, f_{\mathbf{s}}) \in \mathcal{B}$ is such that $|f_{\mathbf{s}} - \mathsf{f}_{\mathcal{D}}(\mathbf{s})| \leq \varepsilon/2$.*

An ϵ-approximation $\mathcal{B}$ is a superset of $\mathsf{FS}(\mathcal{D},\theta)$ (Property 1) but the "false positives" it contains, i.e., the sequences appearing in a pair of $\mathcal{B}$ but not appearing in $\mathsf{FS}(\mathcal{D},\theta)$, are "almost" frequent, in the sense that their frequency in $\mathcal{D}$ cannot be lower than $\theta - \varepsilon$ (Property 2). Additionally, the estimations of the frequencies for the sequences in $\mathcal{B}$ are all simultaneously up to $\varepsilon/2$ far from their exact values (Property 3). In this work we focus on the absolute error but an extension to relative error is possible.

2.2 VC-dimension and sampling

The (empirical) Vapnik-Chervonenkis (VC) dimension [25] is a fundamental concept from statistical learning theory [24]. We give here the most basic definitions and results, tailored to our settings, and refer the reader to the textbook by Shalev-Shwartz and Ben-David [20] for a detailed presentation of this topic.

Let $\mathcal{H}$ be a finite discrete domain and $\mathcal{R} \subseteq 2^{\mathcal{H}}$ be a set of subsets of $\mathcal{H}$. We call the elements of $\mathcal{R}$ *ranges*, and call $(\mathcal{H},\mathcal{R})$ a *rangeset*. Given $\mathcal{W} \subseteq \mathcal{H}$, we say that $A \subseteq \mathcal{W}$ is *shattered by* $\mathcal{R}$ if for every subset $B \subseteq A$ of A, there is a range $R_B \in \mathcal{R}$ such that $A \cap R_B = B$, i.e., if

$$\{R \cap A \ : \ R \in \mathcal{R}\} = 2^A \ .$$

The *empirical VC-dimension* $\mathsf{EVC}(\mathcal{H},\mathcal{R},\mathcal{W})$ *of* $(\mathcal{H},\mathcal{R})$ *on* $\mathcal{W}$ is the size of the largest subset of $\mathcal{W}$ that can be shattered by $\mathcal{R}$.

For example, let $\mathcal{H}$ to be the integers from 0 to 100, and let $\mathcal{R}$ be the collection of all sets of *consecutive* integers from 0 to 100, i.e.,

$$\mathcal{R} = \{\{a, a+1, \ldots, b\} \ : \ a,b \in \mathcal{H} \text{ s.t. } a \leq b\} \ .$$

Let $\mathcal{W}$ be the set of integers from 10 to 25. The empirical VC-dimension $\mathsf{EVC}(\mathcal{H},\mathcal{R},\mathcal{W})$ of $(\mathcal{H},\mathcal{R})$ on $\mathcal{W}$ is 2, because for any set $A = \{a,b,c\}$ with, w.l.o.g., $a < b < c$ of three distinct integers in $\mathcal{W}$, it is impossible to find a range R in $\mathcal{R}$ such that $R \cap A = \{a,c\}$, thus no such set of size three can be shattered by $\mathcal{R}$, while it is trivial to shatter a set of size two.

In practice, the *relative sizes* of the ranges, i.e., the quantities

$$\left\{\frac{|R|}{|\mathcal{H}|} \ : \ R \in \mathcal{R}\right\}$$

are *unknown*. One is interested in estimating all of them simultaneously with guaranteed accuracy from a subset $\mathcal{W}$ of ℓ elements of the domain $\mathcal{H}$. Let $\phi \in (0,1)$. We say that the set $\mathcal{W}$ is an *ϕ-sample* when is such that

$$\left|\frac{|R \cap \mathcal{W}|}{|\mathcal{W}|} - \frac{|R|}{|\mathcal{H}|}\right| < \phi \text{ for every } R \in \mathcal{R} \ . \qquad (2)$$

The use of the term *ϕ-sample* to denote such a set is motivated by the fact that if

1. $\mathcal{W}$ is a *uniform random sample* of ℓ elements from $\mathcal{H}$; and
2. we can compute an *upper bound to the empirical VC-dimension* of $(\mathcal{H},\mathcal{R})$ on $\mathcal{W}$,

then we can obtain a value ϕ such that, with high probability over the choice of $\mathcal{W}$, $\mathcal{W}$ is a ϕ-sample.

Theorem 1 ([11]). *Let $\mathcal{W}$ be a uniform random sample of ℓ elements from $\mathcal{H}$, and let $d \geq \mathsf{EVC}(\mathcal{H},\mathcal{R},\mathcal{W})$. Let $\eta \in (0,1)$ and*

$$\phi = \sqrt{\frac{d + \ln(1/\eta)}{2\ell}} \ .$$

Then with probability at least $1-\eta$ (over the choice of $\mathcal{W}$), $\mathcal{W}$ is a ϕ-sample.

We use this theorem in the analysis of ProSecCo (see Section 3.3) to ensure that the intermediate results it outputs have strong quality guarantees and converge to $\mathsf{FS}(\mathcal{D},\theta)$.

3. ALGORITHM

In this section we present ProSecCo, our *progressive* algorithm for computing the set of frequent sequences in a dataset.

3.1 Intuition and Motivation

The intuition behind ProSecCo is the following. It processes the dataset in blocks of b transactions each, where b is specified by the user. After having analyzed each block, the algorithm outputs an *intermediate result*, i.e., a high-quality approximation of the collection of frequent sequences (specifically an ε-approximation (Definition 1), for ε computed by ProSecCo), together with frequency estimations and error bounds around these estimates.

The intermediate results have *progressively increasing quality*: the difference between the set of sequences produced as output (and their frequency estimations) and the "true" frequent sequences, decreases as ProSecCo processes more blocks. Specifically,

1. for any two blocks B_i and B_j such that B_j is processed by ProSecCo *after* B_i, the intermediate results output by the algorithm after having processed B_j will contain a *subset* of the sequences output after having processed B_i; and

2. the error bounds associated to the estimations of the frequencies of these sequences will be *smaller* than the corresponding error bounds output after having processed B_i.

Once the last block has been analyzed, the final output of the algorithm is, with high probability, the exact collection of frequent sequences (see Theorem 2).

Extreme care is needed when processing the blocks, in order to ensure that each intermediate result is indeed a high-quality approximation of the collection of frequent sequences. This requirement is absolutely crucial otherwise the intermediate results cannot be *trusted*: the user is not able to decide whether to continue or interrupt the processing of the data because the intermediate results have already shown what they were interested in. It is the combination of frequently-updated intermediate results and the *trustworthiness* (i.e., the progressively-refined high-quality) of these results that enables interactive data exploration.

Among these two properties, the latter is the most difficult to achieve. For example, one may think that a successful strategy could involve returning the collection of frequent sequences w.r.t. the original user-specified minimum frequency threshold θ in the first block as the first intermediate result and then augmenting or refining this collection using the frequent sequences in each of the successive blocks. This strategy would not achieve the desired property of trustworthiness for two reasons:

1. there is no guarantee that a sequence that is frequent w.r.t. θ in a single block or even in a small number of blocks would actually be frequent w.r.t. θ win the whole dataset; and

2. while the "true" frequent sequences will definitively be frequent w.r.t. θ in some of the blocks, they may have frequency strictly less than θ in other blocks, and therefore they may be missing from the intermediate results if one is not careful.

Thus, a strategy like the one just described may output intermediate results that include a high number of false positives and also may be missing a large number of the "true" frequent sequences. Such intermediate results would mislead the user, who might make a decision on further steps in the analysis (or stop the current computation) on the basis of wrong information.

ProSecCo avoids these issues by carefully mining the first block at a *lowered* frequency threshold $\xi < \theta$ computed using information obtained from the first block. Thus, the obtained collection $\mathcal{F}$ of "candidate" frequent sequences is a *superset* of $\mathsf{FS}(\mathcal{D}, \theta)$ (more specifically, it is an ε-approximation, for an ε computed by ProSecCo). ProSecCo then refines the candidate set $\mathcal{F}$ using the additional information obtained from mining each of the successive blocks at a *data-dependent, block-specific* lowered frequency threshold, improving the quality of the candidate set (i.e., decreasing ε progressively and including fewer false positives) and eventually converging exactly to $\mathsf{FS}(\mathcal{D}, \theta)$.

Using a block-specific lowered frequency threshold that is data-dependent and not fixed in advance by the user allows ProSecCo to avoid a serious shortcoming of *streaming* algorithms for frequent sequence mining [13], which use a *fixed, user-specified* lowered frequency threshold: the temporary results produced by these algorithms may be of low quality, as they may not contain any of the sequences that are frequent in the whole dataset, because they may have a frequency in the block lower than the user-specified threshold. Such results would not help the end user, as they are not trustworthy.

In the following sections, we first describe ProSecCo, and then analyze its correctness.

3.2 Algorithm description

The following concept is of crucial importance for ProSecCo.

Definition 2. *Given a set $\mathcal{W}$ of transactions, the* s-index *of $\mathcal{W}$ is the largest integer d such that $\mathcal{W}$ contains at least d transactions of item-length at least d and such that for any two distinct such transactions of item-length at least d, neither is a subsequence (proper or improper) of the other.*

In other words, the s-index of $\mathcal{W}$ is the number of transactions in the largest *antichain* of transactions in $\mathcal{W}$, where the partial order for the transaction is defined by the subsequence operator.

Consider, for example, the following set $\mathcal{W}$ of five transactions:

$$\begin{array}{l}\langle\{a\},\{a,b\},\{c,d,e\}\rangle\\ \langle\{a\},\{d,e\},\{c,d\}\rangle\\ \langle\{c\},\{b\},\{b,e\}\rangle\\ \langle\{b,d,e\},\{a,b\}\rangle\\ \langle\{c\},\{b\}\rangle\end{array} .$$

It has s-index equal to 4, because there are four transactions (the first four) with item-length at least 4, while the last transaction has item-length two.

Given $\mathcal{W}$, an upper bound to its s-index d can be computed in a streaming fashion by starting with $d = 0$ and increase it progressively by looking at the transactions in $\mathcal{W}$ one by one and maintaining the set D composed of $\ell \leq d$ transactions of length at least d and of $d - \ell$ transactions of length exactly d. The pseudocode for computing this upper bound is presented in Algorithm 1.

Algorithm 1: `computeSIndex`

input : transaction set $\mathcal{W}$
output: upper bound to the s-index of $\mathcal{W}$.

$\mathcal{T} \leftarrow \emptyset$
$d \leftarrow 1$
foreach $\tau \in \mathcal{W}$ **do**
  if $|\tau| > d$ **and** $\tau \neq \mathcal{I}$ **and** $\neg\exists \rho \in \mathcal{T}$ **s.t.** $\tau \subseteq \rho$ **then**
    $\mathcal{Z} \leftarrow \mathcal{T} \cup \{\tau\}$
    $d \leftarrow$ largest integer such that $\mathcal{Z}$ contains at least d transactions of length at least d
    $\mathcal{T} \leftarrow$ set of d longest transactions from $\mathcal{Z}$
return d

We are now ready to describe ProSecCo. Its pseudocode is presented in Algorithm 2. ProSecCo takes in input the following parameters: the dataset $\mathcal{D}$, a block size $b \in \mathbb{N}$, a minimum frequency threshold $\theta \in (0, 1]$, and a failure probability $\delta \in (0, 1)$.

The algorithm processes the dataset $\mathcal{D}$ in *blocks* $B_1, \ldots, B_\beta$ where $\beta = \lceil |\mathcal{D}|/b \rceil$, of b transactions each,[2] analyzing the dataset one block at a time. We assume to form the blocks by reading the transactions in the dataset in an order chosen *uniformly at random*, which can be achieved, e.g., using randomized index traversal [15].

ProSecCo keeps two running quantities:

1. a descriptive quantity d which is an upper bound to the s-index (see Definition 2) of the set of transactions seen by the algorithm until now;
2. a set $\mathcal{F}$ of pairs $(\mathbf{s}, f_\mathbf{s})$ where $\mathbf{s}$ is a sequence and $f_\mathbf{s} \in (0, 1]$.

The quantities d and $\mathcal{F}$ are initialized while processing the first block B_1. The quantity d is initialized with an upper bound to the s-index of B_1, computed in a streaming fashion using `computeSIndex` (Algorithm 1) as B_1 is read (line 2 of Algorithm 2). $\mathcal{F}$ is populated with the frequent sequences in B_1 w.r.t. to a lowered minimum frequency threshold $\xi = \theta - \frac{\varepsilon}{2}$ and their corresponding frequencies in B_1 (lines 4 and 5 of Algorithm 2). Any frequent sequence mining algorithm, e.g., PrefixSpan [17], can be used to obtain this set. We explain the expression for ε (line 3) in Section 3.3.

After having analyzed B_1, ProSecCo processes the remaining blocks $B_2, \ldots, B_\beta$. While reading each block B_i, the algorithm updates d appropriately so that d is an upper bound to the s-index of the collection

$$\mathcal{W}_i = \bigcup_{j=1}^{i} B_j$$

of transactions in the blocks $B_1, \ldots, B_i$. The updating of d is straightforward thanks to the fact that `computeSIndex` (Algorithm 1) is a *streaming* algorithm, so by keeping in memory the set $\mathcal{Z}$ (line 5 of Algorithm 1) it is possible to update d as more transactions are read. At this point, ProSecCo updates $\mathcal{F}$ in two steps (both implemented in the function `updateRunningSet`, line 10 of Algorithm 2) as follows:

1. for each pair $(\mathbf{s}, f_\mathbf{s}) \in \mathcal{F}$, ProSecCo updates $f_\mathbf{s}$ as
$$f_\mathbf{s} \leftarrow \frac{f_\mathbf{s}(i-1)b + |\{\tau \in B_i \ : \ \mathbf{s} \sqsubseteq \tau\}|}{i \cdot b}, \qquad (3)$$
so that it is equal to the frequency of $\mathbf{s}$ in $\mathcal{W}_i$.
2. it removes from $\mathcal{F}$ all pairs $(\mathbf{s}, f_\mathbf{s})$ s.t. $f_\mathbf{s} < \theta - \frac{\varepsilon}{2}$, where ε is computed using d as explained in Section 3.3. When processing the last block B_β, ProSecCo uses $\varepsilon = 0$.

Notice that no pairs are ever added to $\mathcal{F}$ after its initialization. The intuition behind removing some pairs from $\mathcal{F}$ is that the corresponding sequences cannot have frequency in $\mathcal{D}$ at least θ. We formalize this intuition in the analysis in Section 3.3.

After each block is processed, ProSecCo outputs an *intermediate result* composed by set $\mathcal{F}$ together with ε (line 11 of Algorithm 2). The properties of the output are presented in the following section.

[2]With the possible exception of the last block $B_{\lceil |\mathcal{D}|/b \rceil}$, which may contain fewer than b transactions. For ease of presentation, we assume that all the blocks have size b.

Algorithm 2: ProSecCo

input : dataset $\mathcal{D}$, block size b, minimum frequency threshold θ, failure probability δ.
output: a set $\mathcal{F}$ which, with probability at least $1 - \delta$, equals $\mathsf{FS}(\mathcal{D}, \theta)$.

1 $\beta \leftarrow \lceil |\mathcal{D}|/b \rceil$ `// Number of blocks`
2 $(B_1, d) \leftarrow$ `readBlockAndUpdateSIndex`$(b, 1)$
3 $\varepsilon \leftarrow 2\sqrt{\frac{d - \ln(\delta) + \ln(\beta - 1)}{2b}}$
4 $\xi \leftarrow \theta - \frac{\varepsilon}{2}$
5 $\mathcal{F} \leftarrow$ `getFS`(B_1, ξ) `// Computes` $\mathsf{FS}(B_1, \xi)$
6 **progressiveOutput** $(\mathcal{F}, \varepsilon)$
7 **foreach** $i \leftarrow 2, \ldots, \beta - 1$ **do**
8 $\quad (B_i, d) \leftarrow$ `readBlockAndUpdateSIndex`(b, i)
9 $\quad \varepsilon \leftarrow 2\sqrt{\frac{d - \ln(\delta) + \ln(\beta - 1)}{2i \cdot b}}$
10 $\quad \mathcal{F} \leftarrow$ `updateRunningSet`$(\mathcal{F}, B_i, \varepsilon)$
11 $\quad$ **progressiveOutput** $(\mathcal{F}, \varepsilon)$
12 $(B_\beta, s) \leftarrow$ `readBlockAndUpdateSIndex`(b, β)
13 $\mathcal{F} \leftarrow$ `updateRunningSet`$(\mathcal{F}, B_\beta, 0)$
14 **return** $(\mathcal{F}, 0)$

3.3 Correctness analysis

We show the following result on the properties of ProSecCo.

Theorem 2. *Let $(\mathcal{F}_i, \varepsilon_i)$ be the i-th pair produced in output by* ProSecCo,[3] *$1 \le i \le \beta$. With probability at least $1 - \delta$ (over the runs of the algorithm), it holds that, for all i, $\mathcal{F}_i$ is an ε_i-approximation to $\mathsf{FS}(\mathcal{D}, \theta)$, i.e.:*

$$\Pr(\exists i, 1 \le i \le \beta, \text{s.t. } \mathcal{F}_i \text{ is not an } \varepsilon_i\text{-approximation}) < \delta \ .$$

Before proving the theorem we need some definitions and preliminary results. Consider the range set $(\mathcal{D}, \mathcal{R})$, where $\mathcal{R}$ contains one set $R_\mathbf{s}$ for any sequence $\mathbf{s} \in \mathbb{S}$ defined as the set of transactions of $\mathcal{D}$ that $\mathbf{s}$ belongs to:

$$R_\mathbf{s} = \{\tau \in \mathcal{D} \ : \ \mathbf{s} \sqsubseteq \tau\} \ . \qquad (4)$$

From (1) it is easy to see that for any sequence $\mathbf{s} \in \mathbb{S}$, the relative size of the range $R_\mathbf{s}$ equals the frequency of $\mathbf{s}$ in $\mathcal{D}$:

$$\frac{|R_\mathbf{s}|}{|\mathcal{D}|} = \mathsf{f}_\mathcal{D}(\mathbf{s}) \ . \qquad (5)$$

Given a subset $\mathcal{W}$ of $\mathcal{D}$, it holds

$$\frac{|R_\mathbf{s} \cap \mathcal{W}|}{|\mathcal{W}|} = \mathsf{f}_\mathcal{W}(\mathbf{s}) \ . \qquad (6)$$

Lemma 1. *Let $\mathcal{W}$ be a subset of $\mathcal{D}$ that is a ϕ-sample of $(\mathcal{D}, \mathcal{R})$ for some $\phi \in (0, 1)$. Then the set*

$$\mathcal{B} = \{(\mathbf{s}, \mathsf{f}_\mathcal{W}(\mathbf{s})) \ : \ \mathbf{s} \in \mathsf{FS}(\mathcal{W}, \theta - \phi)\}$$

is a 2ϕ-approximation for $\mathsf{FS}(\mathcal{D}, \theta)$.

Proof. Property 3 from Definition 1 follows immediately from the definition of ϕ-sample (see (2)) and from (5) and (6), as for *all* sequences (not just those in the first components of the pairs in $\mathcal{B}$) it holds

$$|\mathsf{f}_\mathcal{W}(\mathbf{s}) - \mathsf{f}_\mathcal{D}(\mathbf{s})| \le \phi \ .$$

Property 1 from Definition 1 follows from the fact that any sequence $\mathbf{s} \in \mathsf{FS}(\mathcal{D}, \theta)$ has frequency in $\mathcal{W}$ greater than $\theta - \phi$, so the pair $(\mathbf{s}, \mathsf{f}_\mathcal{W}(\mathbf{s}))$ is in $\mathcal{B}$.

[3]I.e., the i-th intermediate result.

Finally, Property 2 from Definition 1 follows from the fact that any sequence $\mathbf{s}$ with frequency in $\mathcal{D}$ *strictly smaller than* $\theta - 2\phi$ has frequency in $\mathcal{W}$ *strictly smaller* than $\theta - \phi$, so the pair $(\mathbf{s}, \mathsf{f}_{\mathcal{W}}(\mathbf{s}))$ is *not* in $\mathcal{B}$. □

The following lemma connects the task of frequent sequence mining with the concepts from statistical learning theory.

LEMMA 2. *For any subset $\mathcal{W} \subseteq \mathcal{D}$ of transactions of $\mathcal{D}$, the s-index d of $\mathcal{W}$ is an upper bound to the empirical VC-dimension of $(\mathcal{D}, \mathcal{R})$ on $\mathcal{W}$: $d \leq \mathsf{EVC}(\mathcal{D}, \mathcal{R}, \mathcal{W})$.*

PROOF. Assume that there is a subset $\mathcal{S} \subseteq \mathcal{W}$ of $z > d$ transactions that can be shattered by $\mathcal{R}$. From the definition of d, $\mathcal{S}$ must contain a transaction τ of item-length d. The transaction τ belongs to 2^{z-1} subsets of $\mathcal{S}$. We label these subsets arbitrarily as A_i, $1 \leq i \leq 2^{z-1}$. Since $\mathcal{S}$ is shattered by $\mathcal{R}$, for each A_i there must be a range $R_i \in \mathcal{R}$ such that

$$A_i = \mathcal{S} \cap R_i, \text{for each } 1 \leq i \leq 2^{z-1} \ .$$

Since all the A_i's are different, so must be the R_i's. The transaction τ belongs to every A_i so it must belong to every R_i as well. From the definition of $\mathcal{R}$, there must be, for every $1 \leq i \leq 2^{z-i}$, a sequence $\mathbf{s}_i$ such that $R_i = R_{\mathbf{s}_i}$ (see (4)). Thus, all the $\mathbf{s}_i$'s must be different. From (4) it holds that τ belongs to all and only the ranges $R_{\mathbf{q}}$ such that $\mathbf{q} \sqsubseteq \tau$. There are at most $2^d - 1$ distinct non-empty sequences that are subsequences of τ. But from the definition of z it holds that $2^{z-1} > 2^d - 1$, so τ cannot belong to all the ranges $R_{\mathbf{s}_i}$, thus we reach a contradiction, and it is impossible that $\mathcal{S}$ is shattered. □

We conjecture that the bound is tight, in the sense that it is possible to build a dataset and a set $\mathcal{W}$ of transactions such that the empirical VC-dimension on $\mathcal{W}$ equals the s-index of $\mathcal{W}$.

PROOF OF THEOREM 2. Recall that $\mathcal{W}_i = \bigcup_{j=1}^{i} B_j$ is the set of transactions seen by PROSECCO up to the point that $(\mathcal{F}_i, \varepsilon_i)$ is sent in output. The number of transactions in $\mathcal{W}_i$ is $|\mathcal{W}_i| = b \cdot i$. For any i, $1 \leq i \leq \beta$ and for any pair $(\mathbf{s}, f_{\mathbf{s}}) \in \mathcal{F}_i$, it holds

$$f_{\mathbf{s}} = \mathsf{f}_{\mathcal{W}_i}(\mathbf{s}) \tag{7}$$

by definition of $f_{\mathbf{s}}$ (see (3)). Consider the event

$$\mathsf{E} = \text{“Every } \mathcal{W}_i,\ 1 \leq i < \beta \text{ is an } \varepsilon_i\text{/2-sample”}$$

and let $\bar{\mathsf{E}}$ be its complementary event. Using the union bound, we can write

$$\Pr\left(\bar{\mathsf{E}}\right) \leq \sum_{i=1}^{\beta-1} \Pr(\mathcal{W}_i \text{ is not a } \varepsilon_i\text{/2-sample}) \ . \tag{8}$$

By construction, each $\mathcal{W}_i$ is an uniform random sample of $\mathcal{D}$ of size $b \cdot i$, $1 \leq i < \beta$. The fact that $\mathcal{W}_i \subset \mathcal{W}_j$ for $j > i$ is irrelevant, because of the definition of uniform random sample. Using Lemma 2, Theorem 1 and the definition of ε_i (from lines 3 and 9 of Algorithm 2), it holds

$$\Pr(\mathcal{W}_i \text{ is not a } \varepsilon_i\text{/2-sample}) \leq \frac{\delta}{\beta - 1}, \text{ for } 1 \leq i < \beta \ .$$

Plugging the above in (8), it follows that the event E then happens with probability at least $1 - \delta$. When E happens, the thesis follows from Lemma 1 for all $1 \leq i < \beta$ and from (7) for $i = \beta$. □

3.4 Top-k Sequence Mining

A variant of the frequent sequence mining task requires to find the *top-k* most frequent sequences: instead of specifying the minimum frequency threshold θ, the user specifies a desired output size k. The collection of sequence to return is defined as follows. Assume to sort the sequences in $\mathbb{S}$ according to their frequency in $\mathcal{D}$, ties broken arbitrarily. Let $\mathsf{f}_{\mathcal{D}}^{(k)}$ be the frequency in $\mathcal{D}$ of the k-th sequence in this order. The set of top-k frequent sequences is the set

$$\mathsf{TOPK}(\mathcal{D}, k) = \{\mathbf{s} \in \mathbb{S} \ : \ \mathsf{f}_{\mathcal{D}}(\mathbf{s}) \geq \mathsf{f}_{\mathcal{D}}^{(k)}\} \ .$$

This collection may contain more than k sequences. The use of k as a parameter in place of the minimum frequency threshold θ is often more intuitive for the user and more appropriate for interactive visualization tools, where the human user can only handle a limited number of output sequences.

Since

$$\mathsf{TOPK}(\mathcal{D}, k) = \mathsf{FS}(\mathcal{D}, \mathsf{f}_{\mathcal{D}}^{(k)})$$

the concept of ε-approximation (Definition 1) is valid also for this collection.

PROSECCO can be modified as follows to return progressive results for the top-k frequent sequences. We denote this modified algorithm as PROSEK, and in the following describe how it differs from PROSECCO by referencing the pseudocode in Algorithm 2.

First of all, PROSEK takes k as input parameter instead of θ. A major difference is in the definition of ε on lines 3 and 9 of Algorithm 2. PROSEK uses a factor 4 (instead of 2) before the square root to compute the values for this variable:

$$\varepsilon \leftarrow 4\sqrt{\frac{d - \ln(\delta) + \ln(\beta - 1)}{2i \cdot b}} \ .$$

Another difference is in the initialization of $\mathcal{F}$ (line 5): instead of θ, PROSEK uses $\mathsf{f}_{B_1}^{(k)}$, the frequency in B_1 of the k-th most frequent sequence in B_1:

$$\mathcal{F} \leftarrow \texttt{getFS}(B_1, \xi) \ .$$

The quantity $\mathsf{f}_{B_1}^{(k)}$ can be computed using a straightforward variant of PrefixSpan for top-k frequent sequence mining. The last difference between PROSECCO and PROSEK is in the function `updateRunningSet`: while the second component of the pairs in $\mathcal{F}$ is still updated using (3), PROSEK removes from $\mathcal{F}$ all pairs with updated second component strictly less than $\mathsf{f}_{\mathcal{W}_i}^{(k)} - \frac{\varepsilon}{2}$, the frequency of the k-th most frequent sequence in $\mathcal{W}_i$.

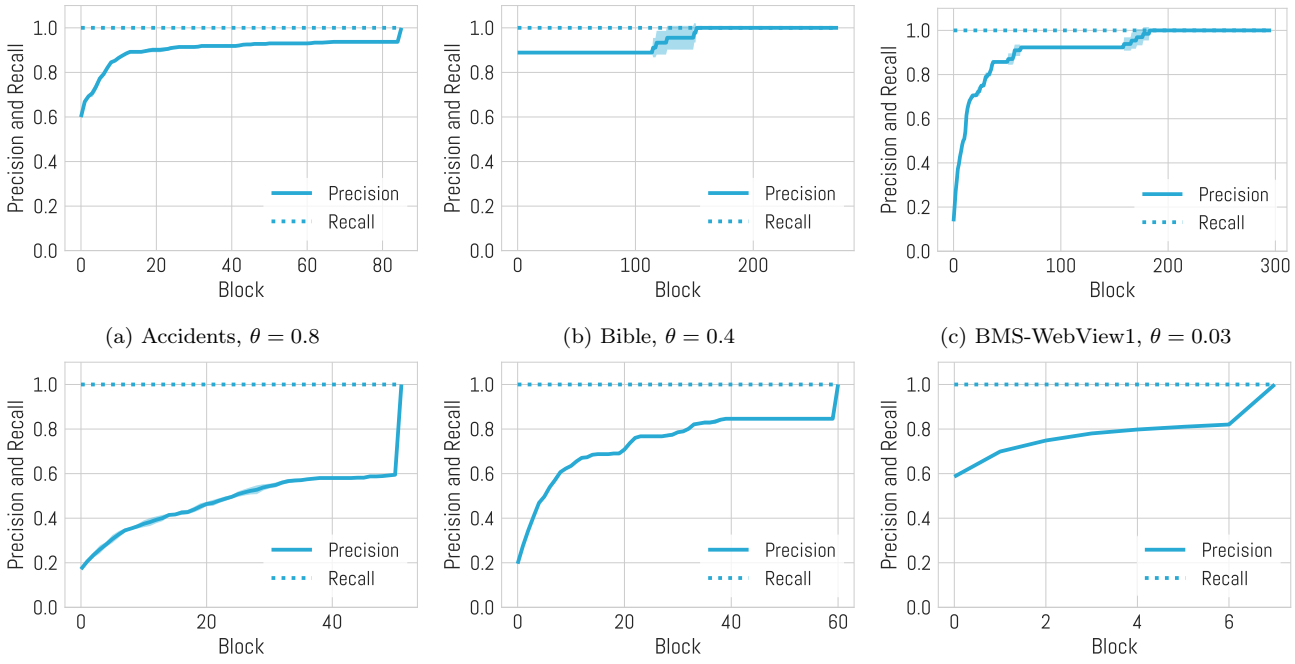
The output of PROSEK has the following properties.

THEOREM 3. *Let $(\mathcal{F}_i, \varepsilon_i)$ be the i-th pair sent in output by* PROSEK, *$1 \leq i \leq \beta$. With probability at least $1 - \delta$, it holds that, for all i, $\mathcal{F}_i$ is an ε_i-approximation to $\mathsf{TOPK}(\mathcal{D},)$.*

The proof follows essentially the same steps as the one for Theorem 2.

3.5 Discussion

We now comment on some important aspects of PROSECCO, including how to make it even more efficient in practice.



(a) Accidents, $\theta = 0.8$ (b) Bible, $\theta = 0.4$ (c) BMS-WebView1, $\theta = 0.03$

(d) FIFA, $\theta = 0.3$ (e) Kosarak, $\theta = 0.05$ (f) Sign, $\theta = 0.4$

Figure 2: Precision and Recall evolution as more blocks are processed.

Parallel processing. ProSecCo can be easily parallelized without sacrificing its progressive nature: after having processed the initial block, it can mine *all* subsequent blocks *in parallel*. The set $\mathcal{F}$ is updated (and sent in output) every time the mining of a block has completed, and can be done with minimal synchronization. This optimization can drastically increase the power of ProSecCo in practical settings: ProSecCo can now compute the *exact* collection of frequent sequences very fast, while *simultaneously* producing intermediate results. This kind of parallelization combined with interactivity can greatly benefit the responsiveness of data exploration tools.

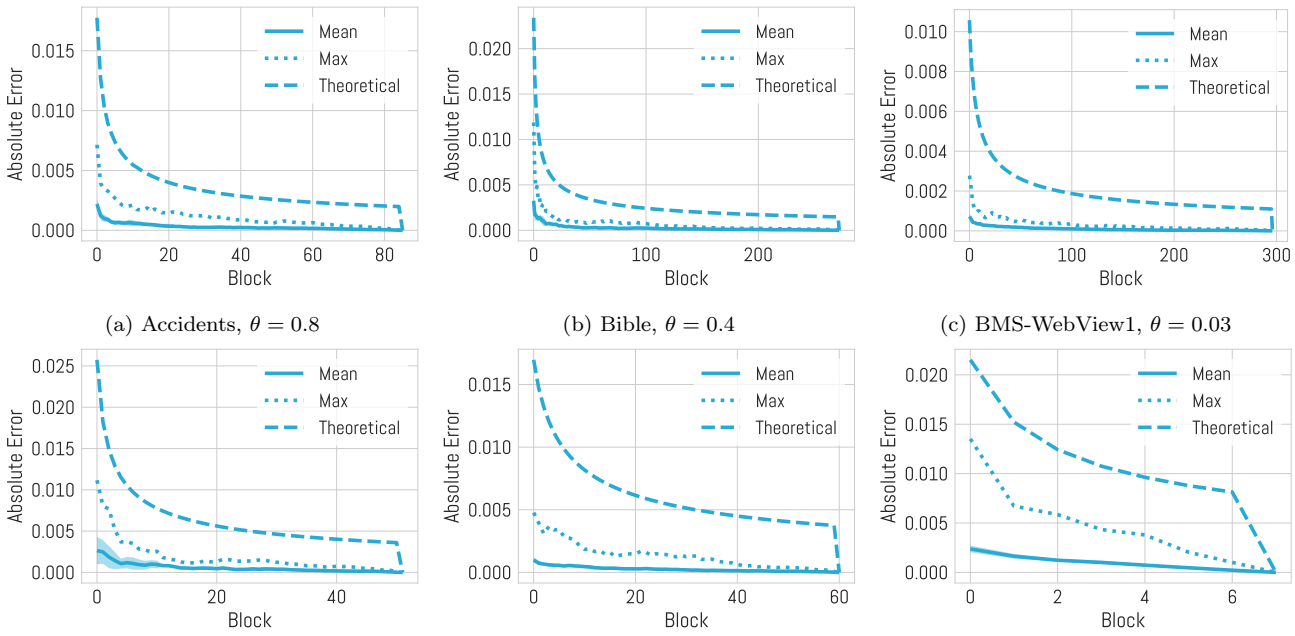
Memory considerations. Many current real-world datasets contain hundreds of millions of transactions. As a result, such datasets are impractical to store, let alone mine, locally on a single machine. Most existing algorithms are ill-suited for mining large datasets as they require enormous amounts of memory to operate (usually ranging in the GigaBytes, see also Section 4.3), even with relatively small datasets by today's standards. Existing workarounds involve expensive disk I/O operations to store and fetch from disk what does not fit into memory, leading to extreme runtime inefficiencies.

Thanks to the fact that ProSecCo only mines one block at a time, it incurs in minimal memory overhead, making it an ideal candidate for mining very large datasets (see also the results of our experimental evaluation in Section 4.3). Furthermore, this small space footprint means that ProSecCo can be used in low-memory settings *without* the need for expensive I/O swapping operations effectively bypassing the runtime increase faced by existing algorithms. We believe this is a major benefit of ProSecCo, given the impracticality of using existing sequence mining algorithms on huge sequence datasets.

Mining optimizations. The main goal of progressive algorithms is to be interactive. A key component of interactivity is to present the first intermediate results to the user as soon as possible. As described above, ProSecCo uses an exact, non-progressive algorithm such as PrefixSpan [17] to mine the first block with a frequency threshold ξ (line 4 of Algorithm 2). Because of the way ξ is computed, it could be very small, depending on the (upper bound to the) s-index of the first block and on the user-specified block size b. Mining the first block at a very low frequency threshold has two undesirable effects:

1. the mining may take a long time due to the very large number of patterns that are deemed frequent w.r.t. a very small threshold (the so-called *pattern explosion* phenomenon);
2. all these patterns would be shown to the user, effectively flooding them with too much information with diminishing return.

To counteract these drawbacks, the algorithm can hold before mining the first block if the frequency threshold ξ is too low, and instead continue on to read the second block (without discarding the first) and potentially additional blocks until the frequency threshold ξ computed using the upper bound to the s-index and the size of the set of all read transactions is large enough for this set of transactions to be mined quickly by PrefixSpan at this threshold. A good starting point for how large ξ should be before mining is to wait until it is approximately $\theta/2$. Other heuristics are naturally possible and we are investigating a cost-model-based optimizer for the mining step to determine when ξ is large enough.



(a) Accidents, $\theta = 0.8$ (b) Bible, $\theta = 0.4$ (c) BMS-WebView1, $\theta = 0.03$

(d) FIFA, $\theta = 0.3$ (e) Kosarak, $\theta = 0.05$ (f) Sign, $\theta = 0.4$

Figure 3: Absolute error in the frequency estimation and its evolution as more blocks are processed.

4. EXPERIMENTAL EVALUATION

In this section we report the results of our experimental evaluation of ProSecCo on multiple datasets.

The goals of the evaluation are the following:

- Assess the accuracy of ProSecCo in terms of:
  1. the precision and the recall of the collections of sequences that are progressively output, and how these quantities change over time as more blocks are processed;
  2. the error in the estimations of the frequencies of the output sequences, and its behavior over time. Additionally, we compare the actual maximum frequency error obtained with its theoretical upper bound ε_i that is output by ProSecCo after having processed the i-th block.
- Measure the running time of ProSecCo both in terms of the time needed to produce the first intermediate result, the successive ones, and the last one. We also compare the latter with the running time of PrefixSpan [17].
- Evaluate the memory usage of ProSecCo over time and compare it with that of PrefixSpan, especially as function of the size of the dataset.

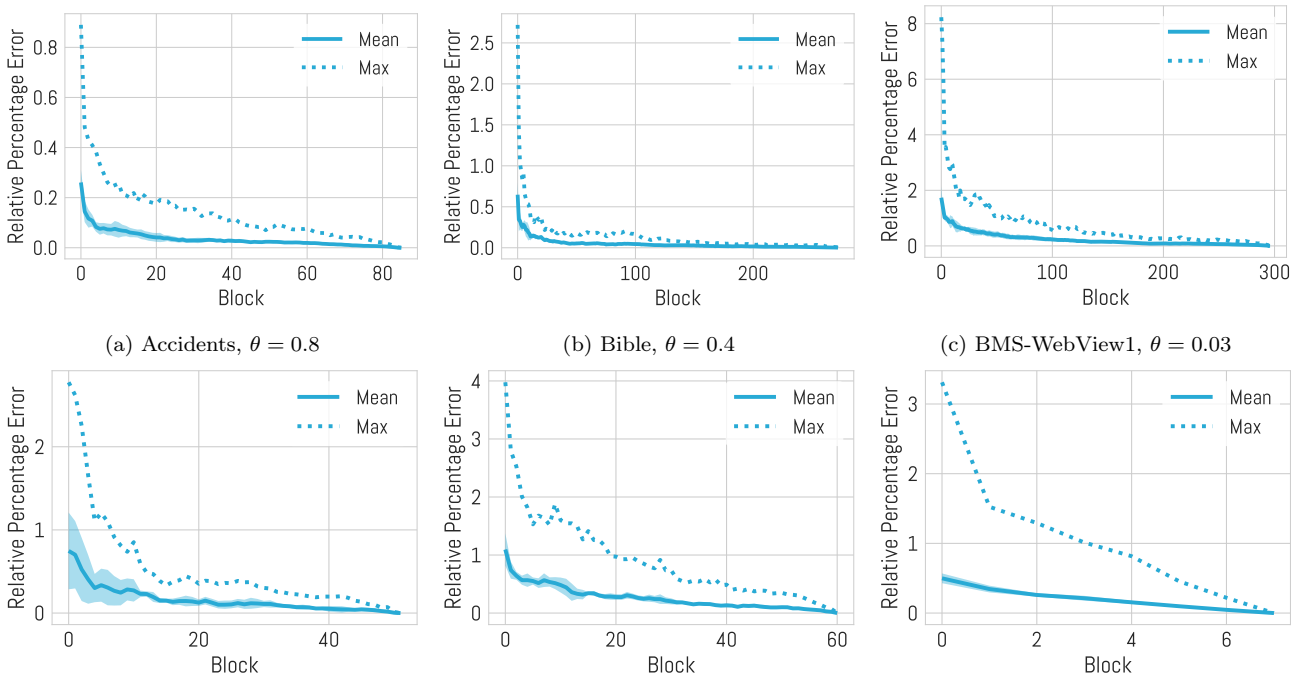
Summary. Overall, our experiments show that ProSecCo is capable of producing high-quality results within a few seconds and quickly converge to the exact collection of frequent sequences. Furthermore, ProSecCo is often *faster* than PrefixSapn while simultaneously capable of producing many progressive results along the way. Finally, ProSecCo uses a constant amount of memory (up to 800 MegaBytes) which was orders of magnitude less when compared to the PrefixSpan algorithm which consistently required several GigaBytes memory in our experiments.

Implementation and Environment. We implement ProSecCo and PrefixSpan in C#. Our implementation of ProSecCo uses PrefixSpan as the black-box non-progressive algorithm to mine the first set $\mathcal{F}$ from the first block (line 5 of Algorithm 2) and for updating this set when processing the successive blocks (line 10). Our open-source implementation can be found at `https://github.com/sachaservan/ProSecCo`.

All experiments are conducted on a machine with *Intel(R) Xeon(R) CPU E5-2660 v2 @ 2.20GHz processors* and 256GB of RAM, running Ubuntu 16.04 LTS. Unless otherwise stated, each result is the average over five trial runs (for each combination of parameters). In most cases the variance across runs was minimal, but we also report 95%-confidence regions (under a normal approximation assumption). These regions are shown in the figures as a shaded areas around the curves.

Datasets. We used six sequence mining datasets obtained from the SPMF Data Mining Repository [6]:

- **Accidents:** Dataset of (anonymized) traffic accidents;
- **Bible:** Conversion of the Bible into a sequence dataset where each word is an item;
- **BMSWebView1:** Click-stream dataset from the Gazelle e-commerce website;
- **FIFA:** Click-stream dataset of the FIFA World Cup '98 website. Each item represents a web page;



(a) Accidents, $\theta = 0.8$ (b) Bible, $\theta = 0.4$ (c) BMS-WebView1, $\theta = 0.03$

(d) FIFA, $\theta = 0.3$ (e) Kosarak, $\theta = 0.05$ (f) Sign, $\theta = 0.4$

Figure 4: Relative percentage error in the frequency estimation and its evolution as more blocks are processed.

- **Kosarak:** Click-stream dataset from a Hungarian online news portal;
- **Sign:** Dataset of sign language utterances;

The characteristics of the datasets are reported in Table 1. To make the datasets more representative of the huge datasets that are frequently available in company environments (and sadly not publicly available), we replicate each dataset a number of times (between 5 and 200). The replication preserves the original distribution of sequence frequencies and transaction lengths, so it does not advantage ProSecCo in any way, or it disadvantages any other algorithm.

Table 1: Dataset characteristics

Dataset	Size ($\lvert\mathcal{D}\rvert$)	Repl. Factor	$\lvert\mathcal{I}\rvert$	Avg. trans. size
Accidents	1700915	5x	481	34.8
Bible	5455350	200x	14442	22.6
BMS-WebView1	5960001	100x	938	3.5
FIFA	1022500	50x	4153	37.2
Kosarak	1249951	50x	16428	9.0
Sign	146000	200x	291	53.0

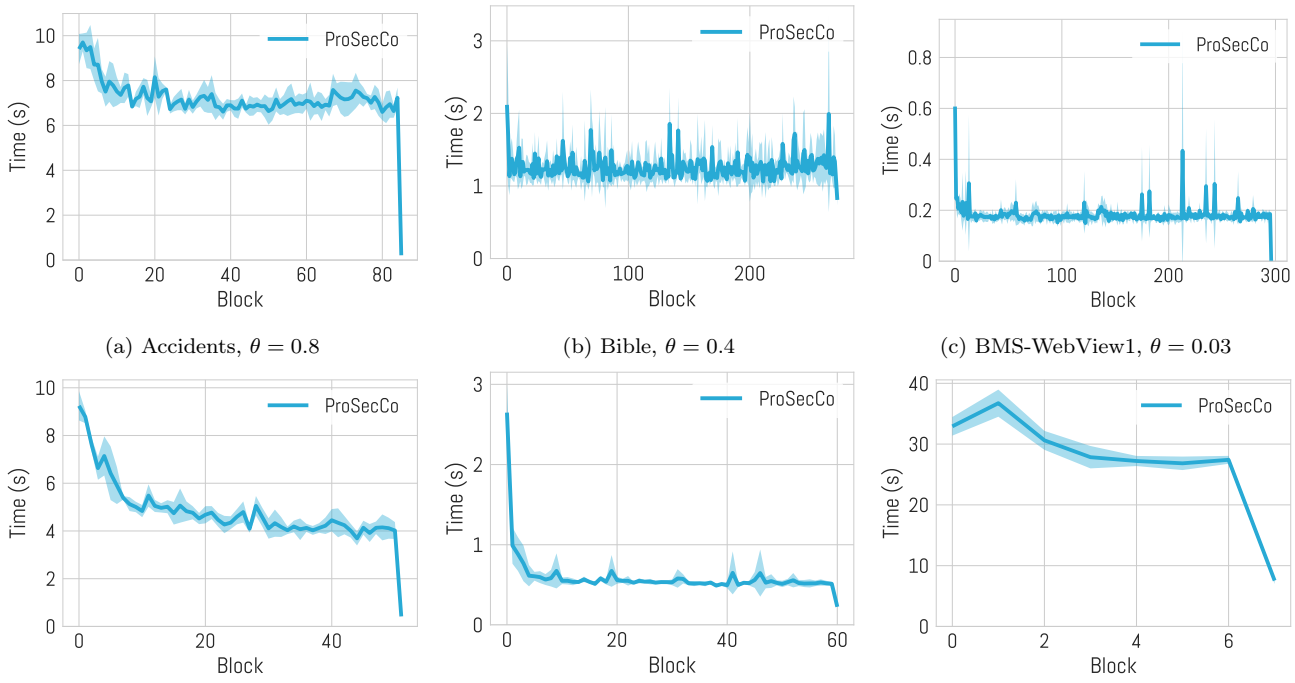
Parameters. We test ProSecCo using a number of different minimum frequency thresholds on each dataset. We report, for each dataset, the results for a single threshold. Additional results are available in the online extended version at `https://github.com/sachaservan/ProSecCo`. We vary the frequency thresholds across the datasets due to the unique characteristics of each dataset. While some datasets have only a few sequences which are common to the majority of transactions, other datasets have sequences common to almost all transactions leading to *pattern explosion* when mining at low thresholds. For example, the Kosarak dataset mined at $\theta = 0.05$ yields 33 frequent sequences, while the Accidents dataset mined at $\theta = 0.85$ produces 71 frequent sequences. This stark variation led us to experiment with frequency thresholds which produce an amount of frequent sequences likely to be of interest in an interactive setting (less than 500 sequences in the final output).

We set $\delta = 0.05$ and do not vary the value of this parameter because the algorithm has only a limited logarithmic (and under square root) dependency on it. We also use a constant block size $b = 20,000$ transactions. This value was found to guarantee the best interactivity of the results.

4.1 Accuracy

We measure the accuracy of ProSecCo in terms of *recall*, *precision* and *frequency error* of the collection of sequences output in each intermediate result. The results for recall and precision are presented in Figure 2 while the ones for the frequency errors can be found in Figure 3 and Figure 4.

Recall. The first result, which is common to *all* the experiments conducted, is that the final output of ProSecCo *always* contains the *exact* collection of frequent sequences, not just with probability $1-\delta$ which is what our theoretical analysis guarantees. In other words, the *recall* of our algorithm at the final iteration is always 1.0 in practice. Furthermore, in all our experiments, the recall *at each progressive output* is also 1.0. In summary, we can say that ProSecCo always



(a) Accidents, $\theta = 0.8$ (b) Bible, $\theta = 0.4$ (c) BMS-WebView1, $\theta = 0.03$

(d) FIFA, $\theta = 0.3$ (e) Kosarak, $\theta = 0.05$ (f) Sign, $\theta = 0.4$

Figure 5: Per-block runtime

produces a progressive output which contains the complete set of frequent sequences (i.e. there are no false negatives in either the progressive or the final output).

Precision. ProSecCo does not offer guarantees in terms of the precision: it only guarantees that any sequence much less frequent than the user-specified minimum threshold would never be included in any intermediate output (see Property 2 of Definition 1). This property is very strong but does not prevent false positives to occur. We can see from the results in Figure 2 that the precision after having processed the first block is around 0.20 for some datasets but it can be much higher (0.6–0.8) or even perfect(see the extended online version). In all cases, it rapidly increases as more blocks are analyzed. Due to the randomized nature of the algorithm, different runs of ProSecCo may show slightly different performances (shaded region around the precision curve) but still show relatively high precision across the board. After a few blocks, the precision tends to plateau: this effect is due to the fact that it is hard for the algorithm to discard from the set $\mathcal{F}$ the sequences with a frequency in $\mathcal{D}$ just slightly lower than θ before ProSecCo has processed the whole dataset. Only after the last block has been analyzed it becomes evident that these sequences do not belong to $\mathsf{FS}(\mathcal{D}, \theta)$ and they can be safely removed from $\mathcal{F}$. Indeed the final output is always exactly $\mathsf{FS}(\mathcal{D}, \theta)$, i.e., the precision of the final output is 1.0.

Frequency Error. We measure the error in the estimation of the frequencies in each progressive output in two ways:

- *absolute error*: the absolute value of the difference between the estimation and the true frequency in $\mathcal{D}$.
- *relative percentage error*: we divide the absolute error by the true frequency in $\mathcal{D}$, and multiply by 100 to obtain a percentage. This quantity measures the percentage difference between the estimated frequencies of the sequences and their actual frequencies.

Results for the absolute error are reported in Figure 3 and those for the relative percentage error are in Figure 4.

Beginning with the absolute error, we can see from the plots that on average over the sequences in the intermediate results for each block, the error is very small (never more than 0.0025) and quickly converges to zero. The error goes to exactly zero after the algorithm has processed the last block. The results are very stable across runs (small or absent shaded region). Even the maximum error is only slightly larger than the mean, which highlights that the error is concentrated around its average. We also report the theoretical upper bound to the maximum error, i.e., the quantity ε_i that is output by ProSecCo after each block has been processed. This quantity is zero after having processed the last block (the single point is not clearly visible in some of the figures). We can see that this bound is larger than the actual maximum error observed, which confirms our theoretical analysis. The fact that at times the bound is significantly larger than the observed error is due to the looseness of the large-deviation bounds used (Theorem 1) and that ProSecCo computes an *upper-bound* to the s-index which in turn is an *upper-bound* to the empirical VC-dimension, itself a worst-case quantity. In the near future, we plan to explore better bounds to the empirical VC-dimension and the use of improved results from statistical learning theory to study the large deviations.

In terms of the relative percentage error (RPE), we remark that ProSecCo does not give any guarantees on this

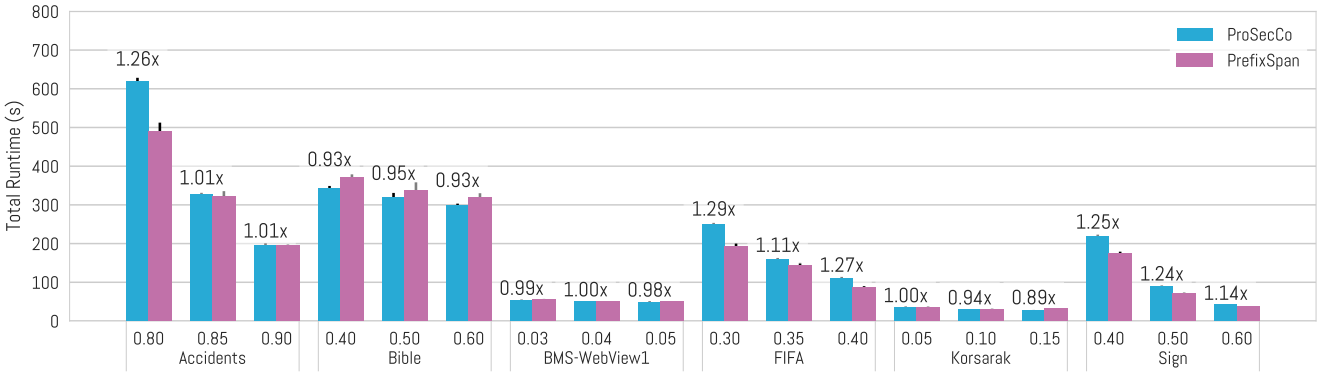


Figure 6: Total runtime comparision for all experiments between PrefixSpan and ProSecCo including 95% confidence intervals. Numbers on top of bars represent ProSecCo's average runtime as a factor of PrefixSpan's average runtime.

quantity (although extensions of ProSecCo that offer guarantees on the RPE and we will discuss them in an extended version of this work). Nevertheless, we can see from the plots in Figure 4 that the RPE is also generally small, and it converges rapidly to zero. The fact that ProSecCo behaves well even with respect to a measure which it was not designed to take into consideration testifies to its great practical usefulness and applicability.

4.2 Runtime

We measure the time it takes for ProSecCo to produce each intermediate result, and compare its completion time with that of PrefixSpan.

Our experiments show (Figures 5 and 6) that ProSecCo provides a progressive output every few seconds (sometimes even milliseconds) producing many incrementally converging and useful results *before* PrefixSpan completes. The variability in the processing time of a block is due to the slightly different thresholds used to mine different blocks. Processing the last block tends to take much less time than analyzing the others because this block is usually significantly smaller than the block size b.

The *overall runtime* of ProSecCo is almost identical (often even faster) to the runtime of PrefixSpan (Figure 6). Even if at times ProSecCo is slower, we stress that it has been producing high-quality results every few seconds, regardless of the overall size of the dataset, while PrefixSpan may require several minutes or more to produce any output.

We break down the total runtime into fractions for the major steps of the algorithm. We report the average percentage of time (relative to the total) for each step across all six datasets.

- 54% (standard deviation: 22%) of the overall runtime is spent reading and parsing the blocks. This step is so expensive because the algorithm must parse each row of the sequence dataset and convert it into an instance of a sequence object in our implementation. This step is not specific to ProSecCo and was equally slow in our PrefixSpan implementation.

- 9.5% (standard deviation: 5.5%) of the runtime was dedicated to updating the s-index as well as sorting and pruning the parsed sequences. After the initial block is processed, the algorithm sorts and prunes each sequence based on the items in the running set $\mathcal{F}$. Doing so allows for a more efficient frequent sequence extraction (see the next step) since the pruned sequences are guaranteed to only contain items which are part of a frequent sequence and it avoids computing the item frequencies from scratch.

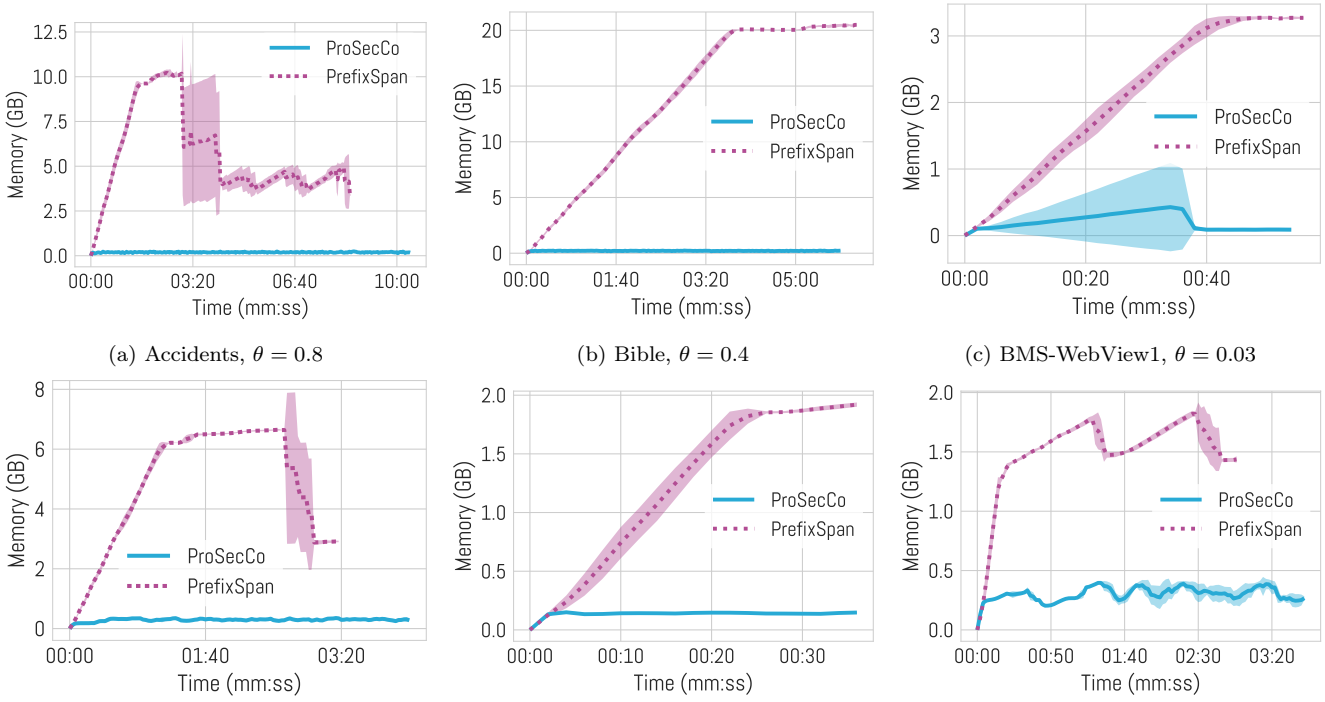
- 36.4% (standard deviation: 25%) of the total runtime involved obtaining the frequent sequences using PrefixSpan. We note that without the previous pruning step, this process would incur a much more significant overhead since the individual item frequencies would need to be computed and the sequences pruned and sorted accordingly.

4.3 Memory Usage

We measure the memory usage over time for both ProSecCo and PrefixSpan. Our results (Figure 7) show that ProSecCo uses a constant amount of memory, approximately 100 to 800MB, regardless of the size of the dataset, while PrefixSpan requires a linear amount of memory which, in some experiments, exceeded 21 GigaBytes. In fact, we were unable to accurately compare performance for several very large datasets which required over 40 GigaBytes of memory to evaluate using existing algorithms, however, we had no issues obtaining results from ProSecCo which always required less than 1 GigaByte of memory to obtain results. Such huge difference of many orders of magnitude clearly shows the advantage of using ProSecCo over classical sequence mining algorithms, especially as datasets get larger and more complex.

5. RELATED WORK

Online aggregation [8] is a paradigm in DBMS operations where the user is presented with on-the-fly and constantly updated results for aggregation queries. A number of systems [1, 2, 10, 4, 7, 9, 16, 27, 28] have been proposed over the years, with increasing levels of sophistications and different trade-offs. One major limitations of most of these systems is their focus on SQL queries and do not cover knowledge discovery tasks that are a major component of data exploration. In this work, we focus on "online aggregation" for one such knowledge discovery task, the extraction of frequent sequences.



(a) Accidents, $\theta = 0.8$ (b) Bible, $\theta = 0.4$ (c) BMS-WebView1, $\theta = 0.03$

(d) FIFA, $\theta = 0.3$ (e) Kosarak, $\theta = 0.05$ (f) Sign, $\theta = 0.4$

Figure 7: Comparison of memory usage between ProSecCo and PrefixSpan.

Frequent sequences mining was introduced by Agrawal and Srikant [3]. A number of exact algorithms for this task have been proposed, ranging from multi-pass algorithms using the anti-monotonicity property of the frequency function [22] to prefix-based approaches [17], to works focusing on the closed frequent sequences [26]. In this work, we consider these algorithms as black-boxes and we run them on blocks of the dataset, without any modification. None of them can work in a progressive, interactive setting like the one we envision (see Figure 1), and in which ProSecCo shines. Additionally, they use a very large amount of memory, while ProSecCo uses an essentially constant amount of memory.

Streaming algorithms for frequent sequences mining [13] process the dataset in blocks, similarly to ProSecCo. The intermediate results they output are not trustworthy, as they may be very-low-quality approximations of the exact collection of frequent sequences. This limitation is due to the fact that the algorithms employ a *fixed, user-specified* lower frequency threshold to mine the blocks, and this quantity may not be small enough to ensure that all "true" frequent sequences are included in each intermediate result. ProSecCo solves this issue by using a *variable, data-dependent* lowered frequency threshold, which offers strong guarantees.

The use of sampling to speed up pattern mining has been successful in other variants of the task, such as frequent itemsets mining [18, 19, 23]. We use similar techniques based on VC-dimension to derive the lowered frequency threshold at which to mine the frequent sequences, but ProSecCo is the first application of random sampling to frequent sequences mining, and previous uses for other pattern mining tasks were not focused on interactive data exploration.

6. CONCLUSIONS

We present ProSecCo, an algorithm for progressive mining of frequent sequences from large transactional datasets.

ProSecCo enables interactive data exploration: it periodically returns to the user intermediate results that are approximations of the collection of frequent sequences with increasingly high quality. Once all the dataset has been processed, the last output result is the exact collection of frequent sequences.

We prove that each returned approximation comes with strong theoretical guarantees, thanks to the use of VC-dimension, a key concept from statistical learning theory: we show an upper bound to the VC-dimension of the task at hand. The bound can be easily obtained in a streaming fashion and allows the algorithm to compute the quality of the approximations it outputs.

Our experimental results show that ProSecCo is able to output a high-quality approximation to the collection of frequent sequences after less than a second, while non-progressive algorithms would take tens of seconds. Additionally, this first high-quality approximation is refined as more blocks of the dataset are processed, and the error progressively and quickly decreases. The last output of the algorithm is guaranteed, with high probability, to be the exact collection of frequent sequences. The estimations of the frequencies of the sequences in output is even better that what is guaranteed by the theoretical analysis.

Among interesting directions for future work, we highlight the need for progressive algorithms for many other knowledge discovery problems, with the goal of making interactive data exploration a reality for more and more complex tasks.

7. REFERENCES

[1] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. The AQUA approximate query answering system. In *Proceedings of the 1999 ACM SIGMOD international conference on Management of data*, SIGMOD '99, pages 574–576, New York, NY, USA, 1999. ACM.

[2] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica. BlinkDB: queries with bounded errors and bounded response times on very large data. In *Proceedings of the 8th ACM European Conference on Computer Systems*, pages 29–42. ACM, 2013.

[3] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proceedings of the Eleventh International Conference on Data Engineering,*, ICDE'95, pages 3–14. IEEE, 1995.

[4] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, K. Elmeleegy, and R. Sears. MapReduce online. In *NSDI*, pages 313–328, 2010.

[5] A. Crotty, A. Galakatos, E. Zgraggen, C. Binnig, and T. Kraska. Vizdom: interactive analytics through pen and touch. *Proceedings of the VLDB Endowment*, 8(12):2024–2027, 2015.

[6] P. Fournier-Viger, C. Lin, A. Gomariz, T. Gueniche, A. Soltani, Z. Deng, and H. T. Lam. The SPMF open-source data mining library version 2. In *Proc. 19th European Conference on Machine Learning and Principles and Practice of Knowledge Discovery and Data Mining (Part III)*, ECML PKDD '16, 2016. `http://www.philippe-fournier-viger.com/spmf/`.

[7] J. M. Hellerstein, R. Avnur, A. Chou, C. Hidber, C. Olston, V. Raman, T. Roth, and P. J. Haas. Interactive data analysis: The Control project. *Computer*, 32(8):51–59, 1999.

[8] J. M. Hellerstein, P. J. Haas, and H. J. Wang. Online aggregation. In *Proceedings of the 1997 ACM SIGMOD international conference on Management of data*, SIGMOD '97, pages 171–182, New York, NY, USA, 1997. ACM.

[9] C. Jermaine, S. Arumugam, A. Pol, and A. Dobra. Scalable approximate query processing with the DBO engine. *ACM Trans. Database Syst.*, 33:23:1–23:54, December 2008.

[10] N. Kamat, P. Jayachandran, K. Tunga, and A. Nandi. Distributed and interactive cube exploration. In *30th IEEE International Conference on Data Engineering*, ICDE '14, pages 472–483. IEEE, 2014.

[11] Y. Li, P. M. Long, and A. Srinivasan. Improved bounds on the sample complexity of learning. *Journal of Computer and System Sciences*, 62(3):516–527, 2001.

[12] Z. Liu and J. Heer. The effects of interactive latency on exploratory visual analysis. *IEEE transactions on visualization and computer graphics*, 20(12):2122–2131, 2014.

[13] L. F. Mendes, B. Ding, and J. Han. Stream sequential pattern mining with precise error bounds. In *Eighth IEEE International Conference on Data Mining*, ICDM'08, pages 941–946. IEEE, 2008.

[14] J. Nielsen. Powers of 10: Time scales in user experience. *Retrieved January*, 5:2015, 2009.

[15] F. Olken. *Random Sampling from Databases*. PhD thesis, University of California, Berkeley, 1993.

[16] N. Pansare, V. R. Borkar, C. Jermaine, and T. Condie. Online aggregation for large MapReduce jobs. *Proc. VLDB Endow*, 4(11):1135–1145, 2011.

[17] J. Pei, J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu. Mining sequential patterns by pattern-growth: The PrefixSpan approach. *IEEE Transactions on knowledge and data engineering*, 16(11):1424–1440, 2004.

[18] M. Riondato and E. Upfal. Efficient discovery of association rules and frequent itemsets through sampling with tight performance guarantees. *ACM Trans. Knowl. Disc. from Data*, 8(4):20, 2014.

[19] M. Riondato and E. Upfal. Mining frequent itemsets through progressive sampling with Rademacher averages. In *Proceedings of the 21st ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, pages 1005–1014. ACM, 2015.

[20] S. Shalev-Shwartz and S. Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014.

[21] B. Shneiderman. Response time and display rate in human performance with computers. *ACM Computing Surveys (CSUR)*, 16(3):265–285, 1984.

[22] R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *International Conference on Extending Database Technology*, ICDT '96, pages 1–17. Springer, 1996.

[23] H. Toivonen. Sampling large databases for association rules. In *Proc. 22nd Int. Conf. Very Large Data Bases*, VLDB '96, pages 134–145, San Francisco, CA, USA, 1996. Morgan Kaufmann Publishers Inc.

[24] V. N. Vapnik. *Statistical learning theory*. Wiley, 1998.

[25] V. N. Vapnik and A. J. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16(2):264–280, 1971.

[26] J. Wang, J. Han, and C. Li. Frequent closed sequence mining without candidate maintenance. *IEEE Transactions on Knowledge and Data Engineering*, 19(8):1042–1056, 2007.

[27] K. Zeng, S. Agarwal, A. Dave, M. Armbrust, and I. Stoica. G-OLA: Generalized on-line aggregation for interactive analysis on big data. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 913–918. ACM, 2015.

[28] K. Zeng, S. Agarwal, and I. Stoica. IOLAP: Managing uncertainty for efficient incremental OLAP. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, pages 1347–1361. ACM, 2016.

[29] E. Zgraggen, A. Galakatos, A. Crotty, J.-D. Fekete, and T. Kraska. How progressive visualizations affect exploratory analysis. *IEEE transactions on visualization and computer graphics*, 23(8):1977–1987, 2017.