

MIT Open Access Articles

ProSecCo: progressive sequence mining with convergence guarantees

The MIT Faculty has made this article openly available. **Please share** how this access benefits you. Your story matters.

As Published: <https://doi.org/10.1007/s10115-019-01393-8>

Publisher: Springer London

Persistent URL: <https://hdl.handle.net/1721.1/131797>

Version: Author's final manuscript: final author's manuscript post peer review, without publisher's formatting or copy editing

Terms of Use: Article is made available in accordance with the publisher's policy and may be subject to US copyright law. Please refer to the publisher's site for terms of use.



ProSecCo: progressive sequence mining with convergence guarantees

Cite this article as: Sacha Servan-Schreiber, Matteo Riondato and Emanuel Zgraggen, ProSecCo: progressive sequence mining with convergence guarantees, Knowledge and Information Systems <https://doi.org/10.1007/s10115-019-01393-8>

This Author Accepted Manuscript is a PDF file of an unedited peer-reviewed manuscript that has been accepted for publication but has not been copyedited or corrected. The official version of record that is published in the journal is kept up to date and so may therefore differ from this version.

Terms of use and reuse: academic research for non-commercial purposes, see here for full terms. <https://www.springer.com/aam-terms-v1>

Author accepted manuscript

Knowledge and Information Systems manuscript No.
(will be inserted by the editor)

ProSecCo: Progressive Sequence Mining with Convergence Guarantees

Sacha Servan-Schreiber · Matteo
Riondato · Emanuel Zraggen

Received: date / Accepted: date

Abstract We present PROSECCO, an algorithm for the *progressive mining* of frequent sequences from large transactional datasets: it processes the dataset in *blocks* and it outputs, after having analyzed each block, a high-quality approximation of the collection of frequent sequences. PROSECCO can be used for interactive data exploration, as the intermediate results enable the user to make informed decisions as the computation proceeds. These intermediate results have strong probabilistic approximation guarantees and the final output is the exact collection of frequent sequences. Our 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 show that it produces fast-converging high-quality results almost immediately. Its practical performance is even better than what is guaranteed by the theoretical analysis, and PROSECCO can even be faster than existing state-of-the-art non-progressive algorithms. Additionally, our experimental results show that PROSECCO uses a constant amount of memory, and orders of magnitude less than other standard, non-progressive, sequential pattern mining algorithms.

A preliminary version of this work appeared in the proceedings of IEEE ICDM'18 [28], where it was deemed the runner-up for the Best Student Paper Award.

S. Servan-Schreiber
MIT CSAIL, Cambridge, MA, USA
E-mail: sachass@mit.edu

M. Riondato
Amherst College Department of Computer Science, Amherst, MA, USA
E-mail: mriondato@amherst.edu

E. Zraggen
MIT CSAIL, Cambridge, MA, USA
E-mail: emzg@mit.edu

“Here grows the wine *Pucinum*, now called *Prosecho*, much celebrated by *Pliny*.”

–Fynes Moryson, *An Itinerary*, 1617

1 Introduction

Data exploration is one of the first steps of data analysis: the user performs a preliminary study of the dataset to get acquainted with it prior to performing deeper analysis. To be useful, systems for data explorations must be *interactive*: small (500ms [16]) and large (6–12s [37]) delays between query and response decrease the rate at which users discover insights.

Data exploration tools, such as Vizdom [7], achieve interactivity by displaying *intermediate results* as soon as possible after the query has been submitted, and frequently update them as more data is processed, using *online aggregation* [10].

The intermediate results must be *trustworthy*, i.e., not mislead the user, otherwise she will not be able to make informed decisions. To be trustworthy, intermediate results must have two important properties: 1) they must be, with high probability, *high-quality approximations* of the exact results; and 2) they must *quickly converge* to the exact results, and correspond to them once all data has been processed.

Online aggregation produces trustworthy 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 since most such algorithms are resource intensive, incurring delays between queries and results. “Batch” algorithms that analyze the whole dataset in one shot can take many minutes to complete, thereby disrupting fluid user experiences. Streaming algorithms often do not offer sufficient guarantees on the quality of intermediate results for them to be trustworthy, and are thus poor candidates for exploratory tasks.

In this work we focus on the important task of *frequent sequence mining* [3, 21], which requires finding ordered lists of itemsets appearing in a large fraction of a dataset of transactions. Applications include web log analysis, finance modeling, and market basket analysis. Our approach can be easily generalized to other pattern extraction tasks such as frequent itemset mining, which is an often simpler task compared to sequential pattern mining.

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 [21]. 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 providing them actionable information as frequently as possible.

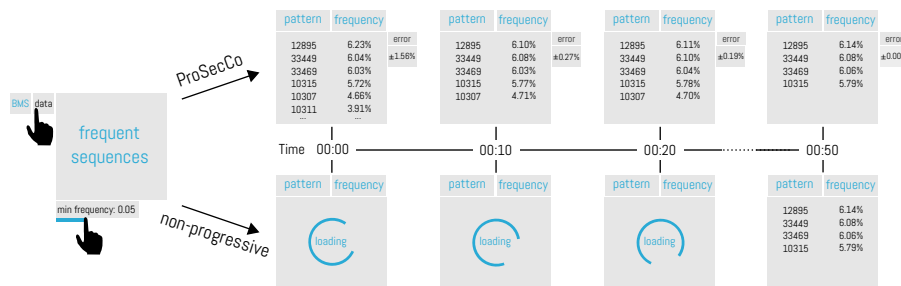


Fig. 1: Illustration of an interactive data analysis tool where users can gesturally invoke a frequent sequences mining operation (left) by selecting a dataset and a minimum frequency threshold. The feedback displayed by the tool to the user varies greatly depending on whether a progressive or a non-progressive 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 PROSECOCO, 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 describe PROSECOCO, a progressive frequent sequence mining algorithm with trustworthy intermediate results, suitable for interactive data exploration.

- PROSECOCO 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 blocks of user-specified size. PROSECOCO extracts a set of candidate frequent sequences from the first block by mining it at a lowered frequency threshold that depends on properties of the block. PROSECOCO often returns 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 probabilistically guaranteed to be a *superset* of the exact collection of frequent sequences. It is progressively refined as more blocks are processed, with each refinement output as an intermediate result. Once the last block has been analyzed, the candidate sets corresponds, with high probability, to the exact collection of frequent sequences. We also present a variant PROSEK for extracting the *top-k* most frequent sequences.
- All the returned sets of candidate sequences come with strong explicit probabilistic guarantees on their quality. Such guarantees enable the user to decide whether to continue or stop the processing of additional blocks. Our analysis uses VC-dimension [33] and fundamental sample-complexity results from statistical learning theory [15, 32]. We show that the empirical VC-dimension of the task of frequent sequence mining is tightly bounded above by a characteristic quantity of the dataset, which we call the *s-index*

(Definition 2), that can be computed in a streaming fashion as the blocks are read (Algorithm 2). A key ingredient of our analysis, that we deem of independent interest, is a method to efficiently compute an *upper bound* to the *capacity* of a sequence, i.e., to the number of distinct subsequences it contains. Obtaining this bound is sufficient for our purposes and faster than computing the exact capacity of a sequence [8].

- We conducted an extensive experimental evaluation of PROSECCO on real and artificial datasets. Our results show that PROSECCO produces approximations of the actual set of frequent sequences almost immediately, with even higher quality than our theoretical analysis guarantees: for example, all the temporary results it produced had the required quality guarantees *every time*, not just with probability $1 - \delta$. We conclude, therefore, that our theoretical analysis provides a generous upper-bound on the *worst case* performance which is not likely to be seen in real-world scenarios. Furthermore, PROSECCO uses a near-constant amount of memory and is, in almost all cases, up to two times faster when compared to the current state-of-the-art sequent mining algorithms PrefixSpan [21] and SPAM [5].

Outline In the next section we discuss the relationship of PROSECCO with related work. We then present preliminary concepts for frequent sequences mining and VC-dimension in Section 3, introduce PROSECCO (Section 4.2) and show the guarantees it offers (Section 4.3). After that, we discuss the variant for mining the top-k frequent sequences, the extension to other kind of patterns, and additional aspect of PROSECCO. The results of our experimental evaluation are presented in Section 5.

2 Related work

Online aggregation [10] 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, 13, 6, 11, 12, 20, 35, 36] 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 they do not cover knowledge discovery tasks that are a major component of data exploration. We focus on online aggregation for one knowledge discovery task: frequent sequences mining.

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 [30], to prefix-based approaches [21], to works focusing on the closed frequent sequences [34]. 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 [17] process the dataset in blocks, similarly to PROSECCO. The intermediate results they output are not trustworthy as they may miss many of the “true” frequent sequences. This limitation is due to the fact that the algorithms employ a *fixed, user-specified* lower frequency threshold to mine the blocks. This quantity is hard for the user to fix, and 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 on the quality of the intermediate and final results.

The use of sampling to speed up the mining phase has been successful in sequence mining [23] and in other variants of pattern discovery, such as frequent itemsets mining [24, 25, 31], to obtain *approximations* of the collection of interesting patterns. We do not use sampling, but we use techniques based on empirical VC-dimension to derive the lowered frequency threshold at which to mine the frequent sequences. Our bound to the empirical VC-dimension is specific to this task, and differs from bounds to the VC-dimension used to obtain approximations of other collection of patterns [24, 26]. Also, we analyze the whole dataset, although in blocks of transactions in random order, to obtain the *exact* collection of frequent sequences.

This version of our work differs in many ways from the preliminary one that appeared in the proceedings of IEEE ICDM’18 [28]. The major changes are the following, listed approximately in order of importance:

- we present a new variant PROSEK, for the interactive mining of the top- k most frequent sequences (Section 4.5);
- we optimize the implementation of PROSECCO, and compare its performance, in addition to that of PrefixSpan [21], to that of SPAM [5], using existing optimized implementations. At the same time, we also use more datasets in the experimental evaluation.
- We show formally that the upper bound to the empirical VC-dimension that we present is tight, in the sense that there are datasets attaining the bound (Lemma 4);
- We discuss how to extend the general approach taken by PROSECCO to other kinds of patterns, specifically itemsets [4] and subgroups [14] (Section 4.6);
- We added running examples for all the concepts that we define, in order to make it easier for the reader to follow our reasoning.

3 Preliminaries

In this section we introduce the concepts and results used throughout the paper. Table 1 reports the main notation for reference.

Table 1: Main notation used in this work.

Symbol	Notes
\mathcal{I}	finite set of items
S	generic itemset, subset of \mathcal{I}
\mathbf{s}	generic sequence, ordered list of itemsets
$ \mathbf{s} $	number of itemsets in \mathbf{s}
$\ \mathbf{s}\ $	item-length of \mathbf{s}
$c(\mathbf{s})$	capacity of \mathbf{s}
$\tilde{c}(\mathbf{s})$	upper bound to the capacity of \mathbf{s}
$\mathbf{a} \sqsubseteq \mathbf{b}$	\mathbf{a} is a subsequence of \mathbf{b}
\mathcal{D}	dataset
$f_{\mathcal{D}}(\mathbf{s})$	frequency of \mathbf{s} on \mathcal{D}
$FS(\mathcal{D}, \theta)$	frequent sequences in \mathcal{D} w.r.t. θ
\mathcal{H}	a finite domain
\mathcal{R}	a collection of subsets from \mathcal{H}
\mathcal{W}	a subset of \mathcal{H}
$EVC(\mathcal{H}, \mathcal{R}, \mathcal{W})$	empirical VC-dimension of $(\mathcal{H}, \mathcal{R})$ on \mathcal{W}
B_i	block of transactions from \mathcal{D}
$TOPK(\mathcal{D}, k)$	top- k frequent sequences in \mathcal{D}

3.1 Sequence mining

Let $\mathcal{I} = \{i_1, \dots, i_n\}$ 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* $\mathbf{s} = \langle S_1, S_2, \dots, S_\ell \rangle$ is a *finite ordered list of itemsets*, with $S_i \subseteq \mathcal{I}$, $1 \leq i \leq \ell$.

The *length* $|\mathbf{s}|$ of \mathbf{s} is the number of itemsets in it, i.e., $|\mathbf{s}| = \ell$. 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., $|\{a, b, c\}| = 3$). For example, $\|\langle \{a, b\}, \{a\}, \{c, d\} \rangle\| = 5$.

A sequence $\mathbf{a} = \langle A_1, A_2, \dots, A_\ell \rangle$ is a *subsequence* of another sequence $\mathbf{b} = \langle B_1, B_2, \dots, B_m \rangle$, if and only if there exist integers $1 \leq j_1 < j_2 < \dots < j_\ell \leq m$ such that $A_1 \subseteq B_{j_1}, A_2 \subseteq B_{j_2}, \dots, A_\ell \subseteq B_{j_\ell}$. We denote that \mathbf{a} is a subsequence of \mathbf{b} as $\mathbf{a} \sqsubseteq \mathbf{b}$. As an example,

$$\langle \{a, b\}, \{a\}, \{c, d\} \rangle \sqsubseteq \langle \{a, b\}, \{b\}, \{a, e\}, \{a, c, d\} \rangle$$

The *capacity* $c(\mathbf{s})$ of \mathbf{s} is the number of *distinct* subsequences of \mathbf{s} :

$$c(\mathbf{s}) = |\{\mathbf{a} \sqsubseteq \mathbf{s}\}|. \tag{1}$$

Egho et al. [8, Sect. 5] present an ingenious method to compute the capacity of a sequence, that is nevertheless quite expensive. The quantity $2^{\|\mathbf{s}\|} - 1$ is an *upper bound* to $c(\mathbf{s})$. PROSECCO uses a stronger upper bound which we introduce in Sect. 4.2.

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 .

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} belongs:

$$f_{\mathcal{D}}(\mathbf{s}) = \frac{|\{\tau \in \mathcal{D} : \mathbf{s} \sqsubseteq \tau\}|}{|\mathcal{D}|} . \quad (2)$$

For example, the following dataset \mathcal{D} has five transactions:

$$\begin{aligned} & \langle \{a\}, \{b, c\}, \{c, d, e\} \rangle \\ & \langle \{a\}, \{d, e\}, \{c, d\} \rangle \\ & \langle \{b, d, e\}, \{a, b\} \rangle \\ & \langle \{b\}, \{c\}, \{d, e\} \rangle \\ & \langle \{a\}, \{a, c\}, \{b\} \rangle . \end{aligned} \quad (3)$$

The last transaction τ is a sequence with length $|\tau| = 3$. Its item-length $\|\tau\|$ is 4. Its capacity $c(\tau)$ is 13 (not $2^4 - 1 = 15$ because there are two ways to get $\langle \{a\} \rangle$ and $\langle \{a\}, \{b\} \rangle$). While the sequence $\langle \{a\} \rangle$ occurs twice as a subsequence of τ , τ is only counted once to compute the frequency of $\langle \{a\} \rangle$ in \mathcal{D} , which is $f_{\mathcal{D}}(\langle \{a\} \rangle) = 4/5$. The sequence $\langle \{a\}, \{b\}, \{c\} \rangle$ is *not* a subsequence of τ because the order of the itemsets in the sequence 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 $\text{FS}(\mathcal{D}, \theta)$ of *frequent sequences in* \mathcal{D} *w.r.t.* θ contains all and only the sequences with frequency at least θ in \mathcal{D} :

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

PROSECCO computes ε -*approximations* of $\text{FS}(\mathcal{D}, \theta)$, for $\varepsilon \in (0, 1)$. Formally, they are defined as follows.

Definition 1 Let $\varepsilon \in (0, 1)$. An ε -*approximation* to $\text{FS}(\mathcal{D}, \theta)$ 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 $\mathbf{s} \in \text{FS}(\mathcal{D}, \theta)$;
2. \mathcal{B} contains no pair $(\mathbf{s}, f_{\mathbf{s}})$ such that $f_{\mathcal{D}}(\mathbf{s}) < \theta - \varepsilon$;
3. Every $(\mathbf{s}, f_{\mathbf{s}}) \in \mathcal{B}$ is such that $|f_{\mathbf{s}} - f_{\mathcal{D}}(\mathbf{s})| \leq \varepsilon/2$.

An ε -*approximation* \mathcal{B} is a *superset* of $\text{FS}(\mathcal{D}, \theta)$ (Property 1) and the “false positives” it contains, i.e., the sequences appearing in a pair of \mathcal{B} but not appearing in $\text{FS}(\mathcal{D}, \theta)$, 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). We focus on the absolute error in the frequency estimation but an extension to relative error is possible.

3.2 VC-dimension and sampling

The (empirical) Vapnik-Chervonenkis (VC) dimension [33] is a fundamental concept from statistical learning theory [32]. 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 [29] for a detailed presentation.

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 and only 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 and only if

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

where 2^A denotes the powerset of A .

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

For example, let \mathcal{H} 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.,

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

Let \mathcal{W} be the set of integers from 10 to 25. The empirical VC-dimension $\text{EVC}(\mathcal{H}, \mathcal{R}, \mathcal{W})$ of $(\mathcal{H}, \mathcal{R})$ on \mathcal{W} is 2. Indeed consider any set $A = \{a, b, c\}$ of three distinct integers in \mathcal{W} , and assume, w.l.o.g., that $a < b < c$. 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 is shattered by \mathcal{R} , while it is trivial to shatter a set of size two.

An important application of empirical VC-dimension is estimating the *relative sizes* of the ranges, which are in practice unknown. Specifically, one is interested in estimating *all* the quantities

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

simultaneously and with guaranteed accuracy, from a subset \mathcal{W} of ℓ elements of the domain \mathcal{H} . A set \mathcal{W} that allows to estimate all these quantities to within an additive error ϕ is called a ϕ -*sample*. Formally, let $\phi \in (0, 1)$. The set \mathcal{W} is a ϕ -*sample* if and only if

$$\left| \frac{|R \cap \mathcal{W}|}{|\mathcal{W}|} - \frac{|R|}{|\mathcal{H}|} \right| < \phi \text{ for every } R \in \mathcal{R} . \tag{4}$$

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. It is important to remark that ϕ is a *sample-dependent* quantity, i.e., it is a property of \mathcal{W} , and it is not fixed beforehand by the user.

Theorem 1 ([15]) *Let \mathcal{W} be a uniform random sample of ℓ elements from \mathcal{H} , and let $d \geq \text{EVC}(\mathcal{H}, \mathcal{R}, \mathcal{W})$. 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 PROSEC_{CO} (see Section 4.3) to ensure that the intermediate results it outputs have strong quality guarantees and converge to $\text{FS}(\mathcal{D}, \theta)$.

4 Algorithms

We now present PROSEC_{CO}, our *progressive* algorithm for computing the set of frequent sequences in a dataset, and some variants for related problems.

4.1 Intuition and motivation

PROSEC_{CO} processes the dataset in blocks $B_1, \dots, B_{\lceil |\mathcal{D}|/b \rceil}$ of b transactions each,¹ for a user-specified b . After having analyzed the i -th block B_i , it outputs an *intermediate result*, which is an ε_i -approximation for an ε_i computed by PROSEC_{CO}.

It is the combination of *frequently-updated* intermediate results and their *trustworthiness* that enables interactive data exploration: *each* intermediate result *must be* a high-quality approximation of the collection of frequent sequences, otherwise 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. Achieving this goal is not straightforward.

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 within 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 (and final) results.

¹ The last block may have fewer than b transactions.

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, thereby misleading the user, who might make a decision on further steps in the analysis (or stop the current computation) on the basis of wrong information.

Streaming algorithms for frequent sequence mining [17] use a *fixed, user-specified, lowered* frequency threshold $\xi < \theta$ to mine all the blocks (the same ξ is used for all blocks). This strategy is not sufficient to guarantee trustworthy intermediate results, as they may not contain many of the sequences that are frequent in the whole dataset, because these sequences may have frequency in a block lower than ξ , and therefore be missing from the intermediate result for that block. Such results would mislead the user.

PROSECCO avoids these pitfalls by carefully mining the initial block at a lowered frequency threshold $\xi < \theta$ computed using *information obtained from the block*.² By doing so, the mined collection \mathcal{F} of “candidate” frequent sequences is a *superset* of $\text{FS}(\mathcal{D}, \theta)$ (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 $\text{FS}(\mathcal{D}, \theta)$. Making the lowered threshold ξ *dynamic and dependent on block-specific information computed by the algorithm* enables PROSECCO to output trustworthy intermediate results.

Algorithm 1: getCapBound: Compute $\tilde{c}(\tau) \geq c(\tau)$.

```

input : transaction  $\tau = \langle A_1, \dots, A_\ell \rangle$ , with the  $A_i$ 's labeled as described in the
         text.
output: upper bound  $\tilde{c}(\tau)$  to  $c(\tau)$ .
1  $c \leftarrow 2^{\|\tau\|} - 1$ 
2  $L \leftarrow \tau$  // Linked list
3 while  $|L| > 1$  do
4    $A \leftarrow \text{popFrontElement}(L)$ 
5   foreach  $B \in L$  and s.t.  $B \subseteq A$  do
6      $c \leftarrow c - (2^{|B|} - 1)$ 
7     erase  $B$  from  $L$ 
8 return  $c$ 

```

4.2 Algorithm description

We first need some preliminary definitions and results.

² Some additional care is needed when handling the initial block. See Section 4.4.

4.2.1 Upper bound to the capacity of a sequence

PROSECOCO relies on a descriptive property of sets of transactions which is a function of the distribution of the *capacities* (see (1)) of the transactions in the sets. Obtaining the exact capacity $c(\tau)$ of a transaction τ is possible thanks to an ingenious formula by Egho et al. [8, Sect. 5], but expensive. PROSECOCO instead computes an *upper bound* $\tilde{c}(\tau) \geq c(\tau)$. The intuition behind our upper capacity bound is the following. Consider the quantity $2^{|\tau|} - 1 \geq c(\tau)$. This quantity may be a *loose* upper bound because it is obtained by considering all subsets of the *bag-union* $\cup_{A \in \tau} A$ of the itemsets in τ as distinct subsequences, but that may not be the case. For example, when τ contains (among others) two itemsets A and B s.t. $A \subseteq B$, sequences of the form $\mathbf{s} = \langle C \rangle$ with $C \subseteq A$ are considered *twice* when obtaining $2^{|\tau|} - 1$, once as “generated” from A and once from B . For example, the subsequence $\langle \{a\} \rangle$ can be “generated” by both the first and the second itemset in the last transaction from (3), but it should not be counted twice.

Our goal in developing a better upper bound to $c(\tau)$ is to avoid over-counting the $2^{|A|} - 1$ sub-sequences of τ in the form of \mathbf{s} above. At an intuitive level, this goal can be achieved by ensuring that such subsequences are only counted once, i.e., as “generated” by the longest itemset that can generate them.

Formally, let $\tau = \langle Z_1, \dots, Z_\ell \rangle$ be a transaction and assume to *re-label* the itemsets in τ by *decreasing size*, ties broken arbitrarily, as A_1, \dots, A_ℓ , so that $|A_i| \geq |A_{i+1}|$. We compute the *capacity upper bound* $\tilde{c}(\tau)$ as follows (pseudocode in Algorithm 1). First, a variable c is set to $2^{|\tau|} - 1$, then we insert the A_i 's in a list L in the order of labeling. As long as the list L contains more than one itemset, we pop the first itemset A from the list, and look for any itemset B still in L such that $B \subseteq A$. For each such B , we decrease c by $2^{|B|} - 1$ and remove B from L . We define the *capacity upper bound* $\tilde{c}(\tau)$ for τ as the value c returned by `getCapBound` (Algorithm 1) with input τ . The following result is then straightforward from this description of `getCapBound` and the intuition given above.

Lemma 1 *For any sequence τ , it holds that $\tilde{c}(\tau) \geq c(\tau)$.*

There are many other types of sub-sequences of τ that may be over-counted, but we have to strike the right trade-off between the time it takes to identify the over-counted subsequences and the gain in the upper bound to the capacity. Investigating better bounds to the capacity of a transaction that can still be computed efficiently is an interesting direction for future work.

4.2.2 The s -index of a set of transactions

Given a set \mathcal{W} of transactions, we use the capacity upper bounds of the transactions in \mathcal{W} to define a characteristic quantity of \mathcal{W} , which we call the *s -index* of \mathcal{W} . This concept is of crucial importance for PROSECOCO.

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 with capacity upper bounds at least $2^d - 1$, and such that for any two distinct such transactions, neither is a subsequence (proper or improper) of the other.

Consider, for example, the set of five transactions from (3). It has s-index equal to 4 because the first four transactions have capacity upper bounds at least $2^4 - 1 = 15$ (each τ of the first four has $\tilde{c}(\tau) = 2^{\|\tau\|} - 1$), while the last transaction τ has $\tilde{c}(\tau) = 14$.

Because it uses the capacity upper bounds, the s-index is tailored for the task of frequent sequence mining. It is in particular different from the d-index of a transactional dataset used for mining approximations of the frequent itemsets through sampling [24].

Given \mathcal{W} , an *upper bound* to its s-index d can be computed in a streaming fashion as follows (pseudocode in Algorithm 2). We maintain a min-priority queue \mathcal{T} of transactions where the priority of an element τ is its capacity upper bound $\tilde{c}(\tau)$. The priority queue is initially empty. At any point during the execution of the algorithm, the priority queue contains

- all ℓ (for some $\ell \leq |\mathcal{T}|$) transactions seen so far with capacity upper bound *strictly greater* than $2^{\ell} - 1$; and
- $|\mathcal{T}| - \ell$ of the transactions seen so far with capacity upper bound *exactly* $2^{\ell} - 1$.

This property of \mathcal{T} is the invariant maintained by the algorithm. The transactions in \mathcal{W} are processed one by one. For each transaction τ , its capacity upper bound $\tilde{c}(\tau)$ is computed. If it is larger than $2^{|\mathcal{T}|} - 1$ and if \mathcal{T} does not contain any transaction of which τ is a subsequence, then τ is inserted in \mathcal{T} with priority $\tilde{c}(\tau)$. We then peek at the top element in \mathcal{T} , and pop it if its capacity upper bound is less than $2^{|\mathcal{T}|} - 1$. Once all transactions of \mathcal{W} have been processed, the number $|\mathcal{T}|$ of elements in the priority queue \mathcal{T} is an upper bound to the s-index of \mathcal{W} .

Algorithm 2: getSIndexBound

```

input : transaction set  $\mathcal{W}$ 
output: upper bound to the s-index of  $\mathcal{W}$ .
1  $\mathcal{T} \leftarrow$  empty min-priority queue
2 foreach  $\tau \in \mathcal{W}$  do
3    $\tilde{c}(\tau) \leftarrow$  getCapBound( $\tau$ ) // See Alg. 1
4   if  $\tilde{c}(\tau) > 2^{|\mathcal{T}|} - 1$  and  $\neg \exists \rho \in \mathcal{T}$  s.t.  $\tau \sqsubseteq \rho$  then
5      $\mathcal{T}$ .insertWithPriority( $\tau, \tilde{c}(\tau)$ )
6     if  $\mathcal{T}$ .peek().priority()  $< 2^{|\mathcal{T}|} - 1$  then
7        $\mathcal{T}$ .pop()
8 return  $|\mathcal{T}|$ 

```

4.2.3 PROSECCO, the algorithm

We are now ready to describe PROSECCO. Its pseudocode is presented in Algorithm 3. PROSECCO takes in input the following parameters: a 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, \dots, B_β where $\beta = \lceil |\mathcal{D}|/b \rceil$, of b transactions each,³ analyzing 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 [19]. This requirement is crucial for the correctness of the algorithm.

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 quantity d is initialized with an upper bound to the s-index of B_1 , computed in a streaming fashion using `getSIndexBound` (Algorithm 2) as B_1 is read (line 2 of Algorithm 3). The second quantity \mathcal{F} is populated with the frequent sequences in B_1 w.r.t. a lowered minimum frequency threshold $\xi = \theta - \frac{\varepsilon}{2}$ and their corresponding frequencies in B_i (lines 4 and 5 of Algorithm 3). Any frequent sequence mining algorithm, e.g., PrefixSpan [21], can be used to obtain this set. We explain the expression for ε (line 3) in Section 4.3.

After having analyzed B_1 , PROSECCO processes the remaining blocks B_2, \dots, B_β . 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, \dots, B_i . The updating of d is straightforward thanks to the fact that `getSIndexBound` (Algorithm 2) is a *streaming* algorithm, so by keeping in memory the priority queue \mathcal{T} (line 1 of Algorithm 2) 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 11 of Algorithm 3) as follows:

1. for each pair $(\mathbf{s}, f_{\mathbf{s}}) \in \mathcal{F}$, PROSECCO updates $f_{\mathbf{s}}$ as

$$f_{\mathbf{s}} \leftarrow \frac{f_{\mathbf{s}} \cdot (i-1) \cdot b + |\{\tau \in B_i : \mathbf{s} \sqsubseteq \tau\}|}{i \cdot b}, \quad (5)$$

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 4.3. When processing the last block B_β , PROSECCO uses $\varepsilon = 0$.

³ The last block $B_{\lceil |\mathcal{D}|/b \rceil}$ may contain fewer than b transactions. For ease of presentation, we assume that all blocks have size b .

No pairs are ever *added* to \mathcal{F} after the initial block B_1 has been processed. 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 4.3.

After each block is processed, PROSECCO outputs an *intermediate result* composed by the set \mathcal{F} together with ε (line 12 of Algorithm 3).

Algorithm 3: 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 $\text{FS}(\mathcal{D}, \theta)$.

- 1 $\beta \leftarrow \lceil |\mathcal{D}|/b \rceil$ // Number of blocks
- 2 $(B_1, d) \leftarrow \text{readBlockAndUpdateSIndex}(b, 1)$
- 3 $\varepsilon \leftarrow 2\sqrt{\frac{d - \ln(\delta) + \ln(\beta - 1)}{2b}}$
- 4 $\xi \leftarrow \theta - \frac{\varepsilon}{2}$ // Computes lowered threshold
- 5 $\mathcal{F} \leftarrow \text{getFS}(B_1, \xi)$ // Computes $\text{FS}(B_1, \xi)$
- 6 **returnIntermediateResult** (\mathcal{F}, ε)
- 7 **foreach** $i \leftarrow 2, \dots, \beta - 1$ **do**
- 8 $(B_i, d) \leftarrow \text{readBlockAndUpdateSIndex}(b, i)$
- 9 $\varepsilon \leftarrow 2\sqrt{\frac{d - \ln(\delta) + \ln(\beta - 1)}{2i \cdot b}}$
- 10 $\xi \leftarrow \theta - \frac{\varepsilon}{2}$
- 11 $\mathcal{F} \leftarrow \text{updateRunningSet}(\mathcal{F}, B_i, \xi)$
- 12 **returnIntermediateResult** (\mathcal{F}, ε)
- 13 $(B_\beta, d) \leftarrow \text{readBlockAndUpdateSIndex}(b, \beta)$
- 14 $\mathcal{F} \leftarrow \text{updateRunningSet}(\mathcal{F}, B_\beta, \theta)$
- 15 **return** ($\mathcal{F}, 0$)

4.3 Correctness analysis

We show the following property of PROSECCO's outputs.

Theorem 2 *Let $(\mathcal{F}_i, \varepsilon_i)$ be the i -th pair produced in output by PROSECCO,⁴ $1 \leq i \leq \beta$. It holds that*

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

The theorem says that, with probability at least $1 - \delta$ (over the runs of the algorithm), for every $1 \leq i \leq \beta$, each intermediate result \mathcal{F}_i is an ε_i -approximation, and since $\varepsilon_\beta = 0$, the last result corresponds to the *exact* collection $\text{FS}(\mathcal{D}, \theta)$.

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

$$R_{\mathbf{s}} = \{\tau \in \mathcal{D} : \mathbf{s} \sqsubseteq \tau\} . \tag{6}$$

⁴ I.e., the i -th intermediate result.

From (2) 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} :

$$\frac{|R_{\mathbf{s}}|}{|\mathcal{D}|} = f_{\mathcal{D}}(\mathbf{s}) . \tag{7}$$

Also, given a subset \mathcal{W} of \mathcal{D} , it holds that

$$\frac{|R_{\mathbf{s}} \cap \mathcal{W}|}{|\mathcal{W}|} = f_{\mathcal{W}}(\mathbf{s}) . \tag{8}$$

The following results connects the concepts of ϕ -sample and ε -approximation.

Lemma 2 *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*

$$\mathcal{B} = \{(\mathbf{s}, f_{\mathcal{W}}(\mathbf{s})) : \mathbf{s} \in \text{FS}(\mathcal{W}, \theta - \phi)\}$$

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

Proof Property 3 from Definition 1 follows immediately from the definition of ϕ -sample (see (4)) and from (7) and (8), as for *every* sequence \mathbf{s} in \mathbb{S} (not just those in the first components of the pairs in \mathcal{B}) it holds that

$$|f_{\mathcal{W}}(\mathbf{s}) - f_{\mathcal{D}}(\mathbf{s})| \leq \phi .$$

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

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 $(\mathbf{s}, f_{\mathcal{W}}(\mathbf{s}))$ is *not* in \mathcal{B} . □

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

Lemma 3 *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} : $d \leq \text{EVC}(\mathcal{D}, \mathcal{R}, \mathcal{W})$.*

Proof Assume that there is a subset $\mathcal{S} \subseteq \mathcal{W}$ of $z > d$ transactions shattered by \mathcal{R} . From the definition of d , \mathcal{S} must contain a transaction τ of with $\check{c}(\tau) \leq 2^d - 1$. 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 $1 \leq i \leq 2^{z-1}$, a sequence \mathbf{s}_i such that $R_i = R_{\mathbf{s}_i}$ (see (6)). Thus, all the \mathbf{s}_i 's must be different. From (6) it holds that τ belongs

to all and only the ranges $R_{\mathbf{q}}$ such that $\mathbf{q} \sqsubseteq \tau$. Since $\tilde{c}(\tau) \leq 2^d - 1$, it follows from Lemma 1, that 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. \square

We can now prove Theorem 2.

Proof (for 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 when $(\mathcal{F}_i, \varepsilon_i)$ 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 that

$$f_{\mathbf{s}} = f_{\mathcal{W}_i}(\mathbf{s}) \tag{9}$$

by definition of $f_{\mathbf{s}}$ (see (5)). Consider the event

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

and let $\bar{\mathbf{E}}$ be its complementary event. Using the union bound [18, Lemma 1.2], we can write

$$\Pr(\bar{\mathbf{E}}) \leq \sum_{i=1}^{\beta-1} \Pr(\mathcal{W}_i \text{ is not a } \varepsilon_i/2\text{-sample}) . \tag{10}$$

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 $\mathcal{W}_i \subset \mathcal{W}_z$ for $z > i$ is irrelevant, because of the definition of uniform random sample. Using Lemma 3, Theorem 1 and the definition of ε_i (from lines 3 and 9 of Algorithm 3), it holds that

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

Plugging the above in (10), it follows that the event \mathbf{E} then happens with probability at least $1 - \delta$. When \mathbf{E} happens, the thesis follows from Lemma 2 for all $1 \leq i < \beta$ and from (9) for $i = \beta$. \square

Tightness of the bound The bound to the empirical VC-dimension shown in Lemma 3 is tight, as shown in the following Lemma, whose proof is similar to the one for [24, Theorem 4.6].

Lemma 4 *Let d be a positive integer. There is a dataset \mathcal{D} and a set $\mathcal{W} \subseteq \mathcal{D}$ with s -index d such that $\text{EVC}(\mathcal{D}, \mathcal{R}, \mathcal{W}) = d$.*

Proof For $d = 1$, let \mathcal{D} be any dataset containing at least two distinct transactions $t_1 = \langle \{a\} \rangle$ and $t_2 = \langle \{b\} \rangle$, for $a \neq b \in \mathcal{I}$. Let $\mathcal{W} = \{t_1, t_2\}$. It holds $\tilde{c}(t_1) = \tilde{c}(t_2) = 1$, so the s -index of \mathcal{W} is 1. It is straightforward to shatter the subset $\{t_1\} \subseteq \mathcal{W}$, so the empirical VC-dimension is at least 1, and the thesis follows from Lemma 3.

Without loss of generality let $\mathcal{I} = \mathbb{N}$. For $d > 1$, let \mathcal{D} be any dataset containing the set \mathcal{W} defined as follows. The set \mathcal{W} contains:

– The set $\mathcal{K} = \{t_1, \dots, t_d\}$ where

$$t_i = \langle \{0, 1, 2, \dots, i-1, i+1, \dots, d\} \rangle, 1 \leq i \leq d;$$

– Any number of arbitrary transactions with capacity upper bound less than $2^d - 1$.

It holds $\bar{c}(t_i) = c(t_i) = 2^d - 1$ and for no pair t_i, t_j with $i \neq j$ it holds either $t_i \sqsubseteq t_j$ or $t_j \sqsubseteq t_i$, so the s-index of \mathcal{W} is d .

We now show that \mathcal{K} is shattered by \mathcal{R} . For each $\mathcal{A} \in 2^{\mathcal{K}} \setminus \{\mathcal{K}, \emptyset\}$, let $Y_{\mathcal{A}}$ be the sequence

$$Y_{\mathcal{A}} = \langle \{1, \dots, d\} \setminus \{i : t_i \in \mathcal{A}\} \rangle .$$

Let $Y_{\mathcal{K}} = \langle \{0\} \rangle$ and $Y_{\emptyset} = \langle \{d+1\} \rangle$. It holds by construction that

$$R_{Y_{\mathcal{A}}} \cap \mathcal{K} = \mathcal{A}, \text{ for each } \mathcal{A} \subseteq \mathcal{K} .$$

Thus, it holds

$$\mathcal{A} \in \{R \cap \mathcal{K}, R \in \mathcal{R}\} \text{ for each } \mathcal{A} \subseteq \mathcal{K},$$

which is equivalent to say that

$$\{R \cap \mathcal{K}, R \in \mathcal{R}\} = 2^{\mathcal{K}},$$

i.e., that \mathcal{K} is shattered. Since $|\mathcal{K}| = d$, it follows that $\text{EVC}(\mathcal{D}, \mathcal{R}, \mathcal{W}) \geq d$. We obtain the thesis by combining this fact with Lemma 3. \square

4.4 Handling the initial block

A major goal for PROSECCO is to be interactive. Interactivity requires 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 [21] to mine the first block with a frequency threshold ξ (line 4 of Algorithm 3). 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 low threshold (*pattern explosion*);
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. Doing so has no effect on the correctness of the algorithm: the proof of Theorem 2

can be amended to take this change into consideration. A good starting point for how large ξ should be before mining is to wait until it is approximately $\theta/2$. Other heuristics are possible and we are investigating a cost-model-based optimizer for the mining step to determine when ξ is large enough.

4.5 Top- k Sequences 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} in decreasing order according to their frequency in \mathcal{D} , ties broken arbitrarily. Let $f_{\mathcal{D}}^{(k)}$ be the frequency in \mathcal{D} of the k -th sequence in this order. The set of top- k frequent sequences is the set

$$\text{TOPK}(\mathcal{D}, k) = \left\{ \mathbf{s} \in \mathbb{S} : f_{\mathcal{D}}(\mathbf{s}) \geq f_{\mathcal{D}}^{(k)} \right\} .$$

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

Since

$$\text{TOPK}(\mathcal{D}, k) = \text{FS} \left(\mathcal{D}, f_{\mathcal{D}}^{(k)} \right),$$

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 3.

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 3. 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 ξ (line 4): instead of θ , PROSEK uses $f_{B_1}^{(k)}$, the frequency in B_1 of the k -th most frequent sequence in B_1 :

$$\xi \leftarrow f_{B_1}^{(k)} - \frac{\varepsilon}{2} .$$

The quantity $f_{B_1}^{(k)}$ 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 (5), PROSEK removes from

\mathcal{F} all pairs with updated second component strictly less than $f_{\mathcal{W}_i}^{(k)} - \frac{\varepsilon}{2}$, 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 $\text{TOPK}(\mathcal{D}, k)$.*

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

4.6 Extension to other patterns

Our presentation of PROSECCO has been focused on sequences, but a similar approach can be used to enable the interactive mining of other kinds of patterns, provided that it is possible, given a random sample \mathcal{S} of transactions from the dataset and a failure probability λ , to determine a $\phi \in (0, 1)$ such that \mathcal{S} is a ϕ -sample, with probability at least $1 - \lambda$ (over the choice of \mathcal{S}).

We are aware of two kinds of patterns for which methods to compute ϕ have been developed: itemsets [4] and subgroups [14]. For itemsets, Riondato and Vandin [26] showed how to use an upper bound to the empirical VC-dimension of the task of itemsets mining to compute ϕ . The same authors used *pseudodimension* [22], a different concept from statistical learning theory, to compute ϕ for subgroups [27].

Developing methods to compute ϕ for other kinds of patterns is an interesting direction for future research.

4.7 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 (usually ranging in the GigaBytes, see also Section 5.3), 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 far beyond what can be tolerated in an interactive setting.

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 5.3). Furthermore, this small resource 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 small memory usage is a major benefit of PROSECCO, given the impracticality of using existing sequence mining algorithms on huge datasets.

5 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 intermediate results, 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 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 [21] and SPAM [5].
- Evaluate the memory usage of PROSECCO over time and compare it with that of PrefixSpan and SPAM, especially as function of the size of the dataset and frequency threshold parameter.

Summary Our experimental results show that PROSECCO is *faster* than PrefixSpan and SPAM while simultaneously capable of producing within a few milliseconds high-quality results that are updated quickly and rapidly converge to the exact collection of frequent sequences. PROSECCO uses a constant amount of memory (at most 2 GigaBytes in our experiments) which was consistently far less than the amount of memory used by PrefixSpan and SPAM, which often required over 10 GigaBytes of memory (SPAM even requiring over 400 GigaBytes at times). Table 2 shows a high-level summary of the comparison, reporting average runtime and memory usage over all the datasets and parameter values that we tested. More than for the actual values, the summary highlights the superiority of PROSECCO and its added flexibility thanks to the fact that it produces intermediate results. All of these advantages come with a very low price in terms of correctness: while theoretically there is a small probability that some of PROSECCO’s intermediate outputs are not ε -approximations, this event never happened in the thousands of runs that we performed.

Implementation and Environment We implement PROSECCO in Java 11.0.1. Our implementation of PROSECCO uses PrefixSpan as the black-box non-progressive algorithm to mine the first set \mathcal{F} from the initial block (line 5 of Algorithm 3) and for updating this set when processing the successive blocks (line 11). We use the PrefixSpan and SPAM implementations from the SPMF [9] repository, which we adapt and optimize for use in PROSECCO. Our open-source implementation is included in the SPMF package.

All experiments are conducted on a machine with Intel Xeon E5 v3 @ 2.30GHz processors, with 128GB of RAM in total, running Ubuntu 18.04 LTS.

Table 2: Summary of experimental comparisons to existing algorithms in terms of average runtime, memory usage, and correctness, over all tested parameter values and datasets

Algorithm	Avg. Runtime	Avg. Memory	Intermediate Results	Correctness
PROSECOCO	32 seconds	743 MB	Yes	$\geq 95\%^*$
PrefixSpan [21]	46 seconds	10 GB	No	100%
SPAM [5]	3 minutes	72 GB	No	100%

*as guaranteed by theoretical analysis; 100% in experimental evaluation.

Unless otherwise stated, each reported 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 from the SPMF Data Mining Repository [9].

- **Accidents:** Dataset of (anonymized) traffic accidents;
- **Bible:** Sequence of sentences in the Bible. Each word is an item;
- **BMSWebView1:** Click-stream dataset from the Gazelle e-commerce website. Each webpage is an item;
- **FIFA:** Click-stream sequences of the FIFA World Cup ‘98 website. Each item in a sequence represents a web page;
- **Kosarak:** Click-stream dataset from a Hungarian on-line news portal;
- **MSNBC:** Dataset of click-stream data consisting of user browsing patterns on the MSNBC website. Each item represents a web page.

The characteristics of the datasets are reported in Table 3. To make the datasets more representative of the very large datasets that are frequently available in company environments (and sadly not publicly available), we replicate each dataset a number of times (between 5 and 100). The replication preserves the original distribution of sequence frequencies and transaction lengths, so it does not advantage PROSECOCO in any way, nor disadvantages any other sequence mining algorithm.

Parameters We test PROSECOCO using a number of different minimum frequency thresholds on each dataset. We report, for each dataset, the results for two to three selected frequency thresholds. We vary the frequency thresholds across the datasets due to the unique characteristics of each dataset, using thresholds which produce an amount of frequent sequences likely to be of interest in an interactive setting. 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

Table 3: Dataset characteristics

Dataset	Size ($ \mathcal{D} $)	Repl. Factor	$ \mathcal{I} $	Avg. trans. size
Accidents	3401830	10x	481	34.8
Bible	7273800	200x	14442	22.6
BMS-WebView1	17880001	300x	938	3.5
FIFA	2045000	100x	4153	37.2
Kosarak	6999801	100x	16428	9.0
MSNBC	3179000	100x	17.0	13.33

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 $b = 10,000$ transactions unless stated otherwise. This value was found to guarantee the best interactivity (see also Section 5.2 for a comparison of different blocks sizes).

5.1 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. Figure 2 shows the results for recall and precision, while Figure 3 and Figure 4 show the ones for the frequency errors.

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 of *each* intermediate result is also 1.0. In summary, we can say that PROSECCO always produces intermediate results that are supersets of $FS(\mathcal{D}, \theta)$. This improvement over the theoretical results can be explained by the inevitable looseness in the sample complexity bounds.

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 θ would never be included in any intermediate result (see Property 2 of Definition 1). This property is very strong but does not prevent false positives from occurring. We can see from the results in Figure 2 that the

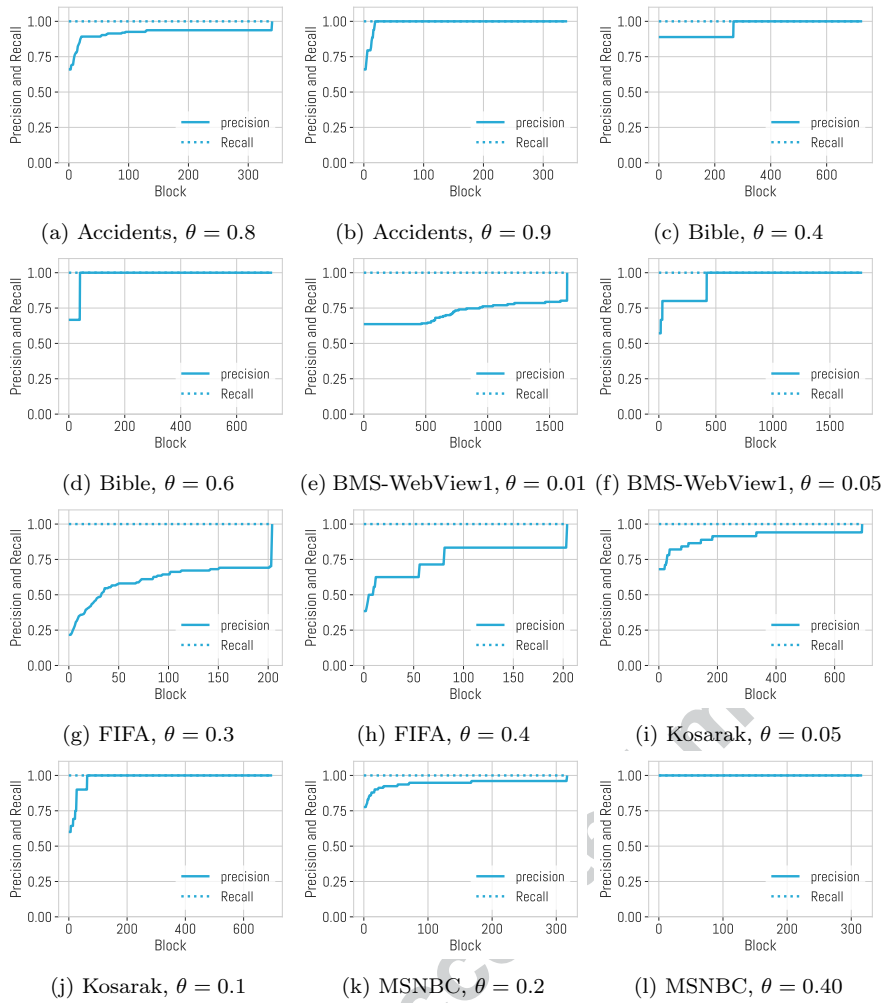


Fig. 2: Precision and recall evolution as more blocks are processed.

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. It rapidly increases in all cases as more blocks are analyzed. Due to the randomized nature of the algorithm, different runs of PROSECOCO may perform slightly differently but the absence of a visible shaded region around the precision curve implies that the difference is insignificant. The precision tends to plateau after a few blocks: this effect is due to the fact that, before having processed the whole dataset, 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 . Only after the last block has been analyzed it becomes evident that these sequences do not belong to $\text{FS}(\mathcal{D}, \theta)$ and they can

be safely expunged from \mathcal{F} . Indeed the final output is always exactly $\text{FS}(\mathcal{D}, \theta)$, i.e., the precision of the final output is 1.0.

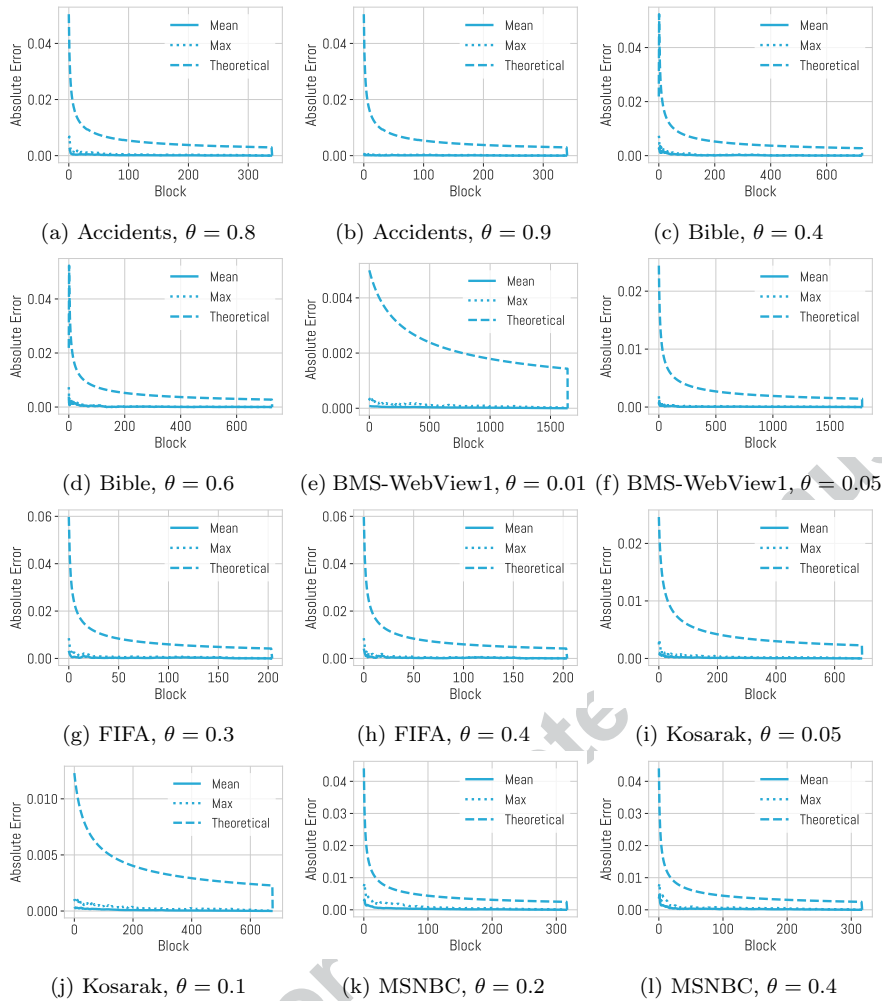


Fig. 3: Absolute error in the frequency estimation and its evolution as more blocks are processed.

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} .

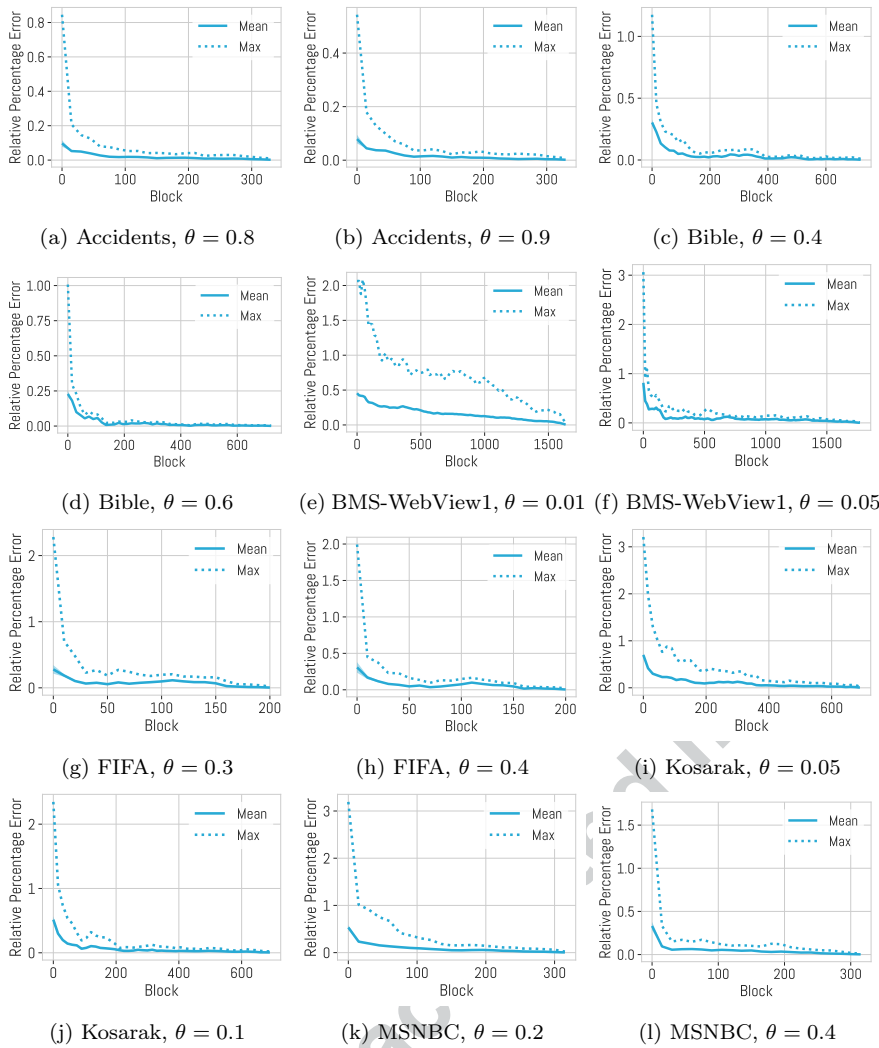


Fig. 4: Relative percentage error in the frequency estimation and its evolution as more blocks are processed.

- *relative percentage error* (RPE): we divide the absolute error by the true frequency in \mathcal{D} , and multiply the result by 100 to obtain a percentage.

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 0.01) 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 (extremely small or absent shaded region). Even the maximum error is only slightly larger than the mean. 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. A good research direction is to explore better bounds for the empirical VC-dimension and the use of improved results from statistical learning theory to study the large deviations.

In terms of the RPE, PROSECCO does not give any guarantees on this quantity (although extensions of PROSECCO that offer guarantees on the RPE are possible). Nevertheless, Figure 4 shows that the RPE is 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.

5.2 Runtime

We measure the time it takes for PROSECCO to produce each intermediate result, and compare its completion time with that of PrefixSpan and SPAM. Our experiments show (Figures 5 and 6) that PROSECCO provides a progressive output every few milliseconds, producing many incrementally converging and useful results *before* PrefixSpan and SPAM 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 usually contains many fewer than b transactions.

We experimented with four different block sizes to analyze the overall effect that block size has on PROSECCO's performance. We stress that the block size only has an effect on the *runtime* required to produce an incremental output but does not impact the correctness of PROSECCO. Figure 5 displays the variation in the time required to produce an incremental output as a function of θ and the block size b . As expected, our experiments show that larger values of b increase the time required per progressive output since each block contains more transactions which need to be processed.

The results suggest that using a "small" block size has the advantage of producing more incremental results, however, using too small a value for b can lead to higher values of ε when mining the blocks, which may slow-down overall performance due to the pattern-explosion phenomena at lowered frequency

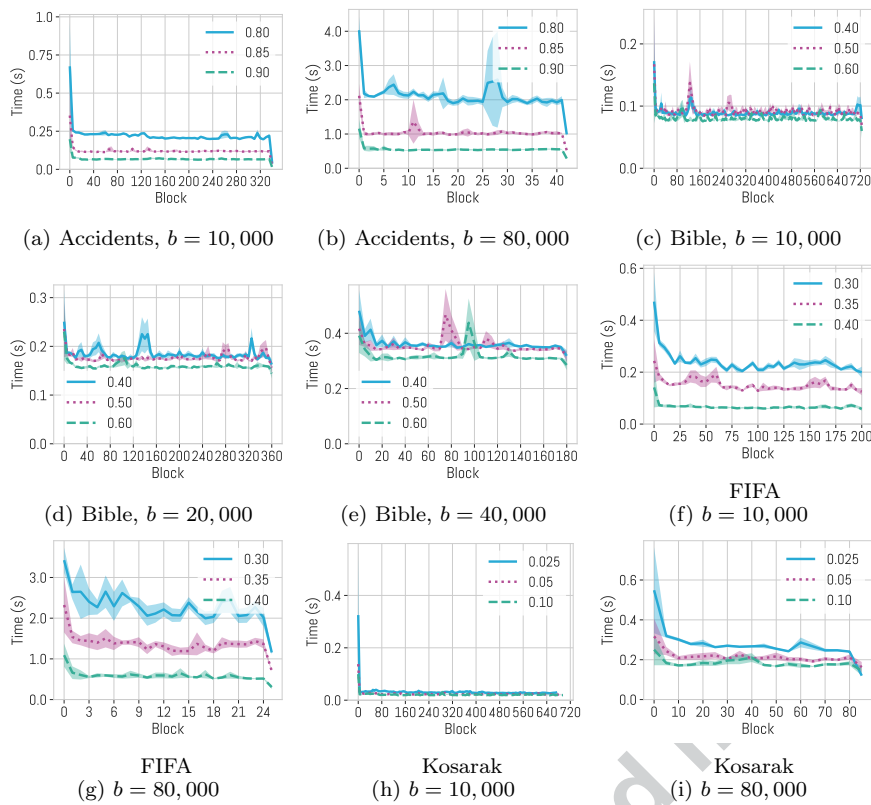


Fig. 5: Per-block runtime for different choices of block size b and different frequency thresholds θ . Results were common across datasets; we select a set of representative results from four datasets.

thresholds. A block size of $b = 10,000$ seems to be a good choice for interactive settings and large datasets.

The *overall runtime* of PROSECCO is almost always smaller than the runtimes of PrefixSpan and SPAM (Figure 6, where we omit plotting the SPAM runtime for clarity, as this algorithm was consistently 2-10x slower than PrefixSpan). On the Bible dataset, PROSECCO is slightly slower than PrefixSpan, but we stress that PROSECCO has been producing high-quality trustworthy results every few milliseconds, regardless of the overall size of the dataset, while PrefixSpan and SPAM may require several minutes or more to produce *any* output. The reason why PROSECCO is slower than PrefixSpan on the Bible dataset is that each transaction in this dataset consists of words in sentences and hence contains many repeated items) has a capacity bound $\tilde{c}(\tau) > 2^{|\mathcal{T}|-1}$, which causes the bound to be re-computed for each transaction (see line 4 of Algorithm 2).

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.

- Roughly 20-30% 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 the PrefixSpan implementation.
- 1-10% 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.
- 50-80% 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.
- 0-30% of the runtime was dedicated to subsequence matching, i.e., computing the frequency of the sequences which did not appear frequent in the current block. This step is computationally expensive but only requires a relatively small percentage of the total runtime since the majority of the truly frequent sequences are likely to be frequent in the i^{th} block and therefore found in the previous step.

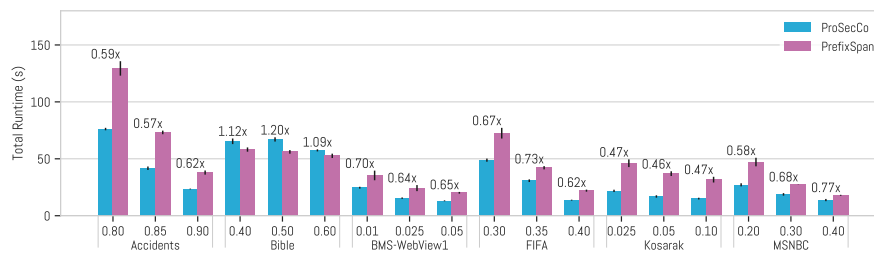


Fig. 6: Total runtime comparison for all experiments between PrefixSpan and PROSECCO including 95% confidence intervals. Numbers on top of bars represent PROSECCO's average runtime as a factor of PrefixSpan's average runtime.

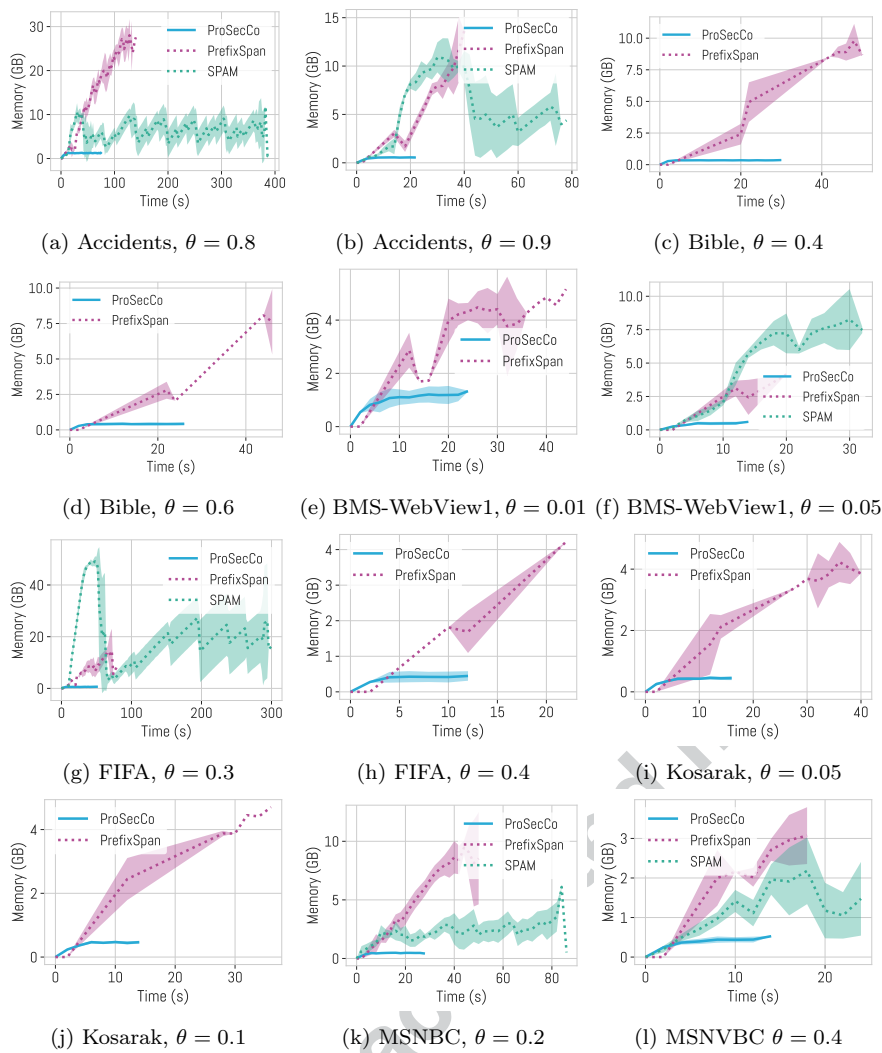


Fig. 7: Comparison of memory usage between PROSECCO and PrefixSpan.

5.3 Memory Usage

We measure the memory usage *over time* for PROSECCO, PrefixSpan, and SPAM. Our results (Figure 7) show that PROSECCO uses a constant amount of memory, 700 MegaBytes on average, regardless of the size of the dataset, while PrefixSpan and SPAM require a linear amount of memory (in the size of the dataset) which, in some experiments, exceeded 30 GigaBytes for PrefixSpan and 400 GigaBytes for SPAM. In fact, we were unable to accurately compare performance for several huge datasets due to memory constraints.

For this reason, we omit SPAM from several figures in order to provide a clearer comparison to PROSECCO. Such 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.

Although measuring memory usage in Java is not straightforward due to the JVM and the automatic garbage collection mechanisms involved, the SPMF code that we use for both PrefixSpan and SPAM, and indirectly for PROSECCO (which uses PrefixSpan at its core), is smart in explicitly invoking the garbage collector at key moments in the algorithms' execution to ensure accurate memory analysis. The SPMF implementation of these algorithms is considered to be of state-of-the-art quality and it is widely used in testing pattern mining algorithms (see, e.g., the Citations page on the SPMF webpage). In the preliminary version of this work [28] we used home-grown C# implementations of PrefixSpan and PROSECCO, and the memory usage patterns of the two algorithms were almost identical to those that we report here.

6 Conclusions

We present PROSECCO, an algorithm for progressive mining of frequent sequences from large transactional datasets. PROSECCO periodically outputs intermediate results that are approximations of the collection $\mathcal{FS}(\mathcal{D}, \theta)$ of frequent sequences, with increasingly high quality. Once all the dataset has been processed, the last result is exactly $\mathcal{FS}(\mathcal{D}, \theta)$.

Each returned approximation comes with strong probabilistic guarantees. The analysis uses VC-dimension, a key concept from statistical learning theory: we show an upper bound to the empirical VC-dimension of the task at hand, which can be easily obtained in a streaming fashion. The bounds allows PROSECCO to compute the quality of the approximations it produces.

Our experimental results show that PROSECCO outputs a high-quality approximation to the collection of frequent sequences after less than a second, while non-progressive algorithms would take tens of seconds. This first approximation is refined as more blocks of the dataset are processed, and the error quickly decreases. The estimations of the frequencies of the sequences in output is even better than 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.

References

1. Acharya S, Gibbons PB, Poosala V, Ramaswamy S (1999) The AQUA approximate query answering system. In: Proceedings of the 1999 ACM SIGMOD international conference on Management of data, ACM, New York, NY, USA, SIGMOD '99, pp 574–576, DOI <http://doi.acm.org/10.1145/304182.304581>, URL <http://doi.acm.org/10.1145/304182.304581>

2. Agarwal S, Mozafari B, Panda A, Milner H, Madden S, Stoica I (2013) BlinkDB: queries with bounded errors and bounded response times on very large data. In: Proceedings of the 8th ACM European Conference on Computer Systems, ACM, pp 29–42
3. Agrawal R, Srikant R (1995) Mining sequential patterns. In: Proceedings of the Eleventh International Conference on Data Engineering, IEEE, ICDE'95, pp 3–14
4. Agrawal R, Imieliński T, Swami A (1993) Mining association rules between sets of items in large databases. SIGMOD Rec 22:207–216, DOI 10.1145/170036.170072
5. Ayres J, Flannick J, Gehrke J, Yiu T (2002) Sequential PAttern Mining using a bitmap representation. In: Proc. 8th ACM SIGKDD Int. Conf. Knowl. Disc. Data Mining, ACM Press, KDD '02, DOI 10.1145/775047.775109, URL <http://dx.doi.org/10.1145/775047.775109>
6. Condie T, Conway N, Alvaro P, Hellerstein JM, Elmeleegy K, Sears R (2010) MapReduce online. In: NSDI, pp 313–328
7. Crotty A, Galakatos A, Zraggen E, Binnig C, Kraska T (2015) Vizdom: interactive analytics through pen and touch. Proceedings of the VLDB Endowment 8(12):2024–2027
8. Egho E, Raïssi C, Calders T, Jay N, Napoli A (2015) On measuring similarity for sequences of itemsets. Data Mining and Knowledge Discovery 29(3):732–764, DOI 10.1007/s10618-014-0362-1, URL <https://doi.org/10.1007/s10618-014-0362-1>
9. Fournier-Viger P, Lin C, Gomariz A, Gueniche T, Soltani A, Deng Z, Lam HT (2016) 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, <http://www.philippe-fournier-viger.com/spmf/>
10. Hellerstein JM, Haas PJ, Wang HJ (1997) Online aggregation. In: Proceedings of the 1997 ACM SIGMOD international conference on Management of data, ACM, New York, NY, USA, SIGMOD '97, pp 171–182, DOI <http://doi.acm.org/10.1145/253260.253291>, URL <http://doi.acm.org/10.1145/253260.253291>
11. Hellerstein JM, Avnur R, Chou A, Hidber C, Olston C, Raman V, Roth T, Haas PJ (1999) Interactive data analysis: The Control project. Computer 32(8):51–59
12. Jermaine C, Arumugam S, Pol A, Dobra A (2008) Scalable approximate query processing with the DBO engine. ACM Trans Database Syst 33:23:1–23:54, DOI <http://doi.acm.org/10.1145/1412331.1412335>, URL <http://doi.acm.org/10.1145/1412331.1412335>
13. Kamat N, Jayachandran P, Tunga K, Nandi A (2014) Distributed and interactive cube exploration. In: 30th IEEE International Conference on Data Engineering, IEEE, ICDE '14, pp 472–483
14. Klösgen W (1992) Problems for knowledge discovery in databases and their treatment in the Statistics Interpreter Explora. International Journal of Intelligent Systems 7:649–673
15. Li Y, Long PM, Srinivasan A (2001) Improved bounds on the sample complexity of learning. Journal of Computer and System Sciences 62(3):516–527
16. Liu Z, Heer J (2014) The effects of interactive latency on exploratory visual analysis. IEEE transactions on visualization and computer graphics 20(12):2122–2131
17. Mendes LF, Ding B, Han J (2008) Stream sequential pattern mining with precise error bounds. In: Eighth IEEE International Conference on Data Mining, IEEE, ICDM'08, pp 941–946
18. Mitzenmacher M, Upfal E (2005) Probability and Computing: Randomized Algorithms and Probabilistic Analysis. Cambridge University Press
19. Olken F (1993) Random sampling from databases. PhD thesis, University of California, Berkeley
20. Pansare N, Borkar VR, Jermaine C, Condie T (2011) Online aggregation for large MapReduce jobs. Proc VLDB Endow 4(11):1135–1145
21. Pei J, Han J, Mortazavi-Asl B, Wang J, Pinto H, Chen Q, Dayal U, Hsu MC (2004) Mining sequential patterns by pattern-growth: The PrefixSpan approach. IEEE Transactions on knowledge and data engineering 16(11):1424–1440
22. Pollard D (1984) Convergence of stochastic processes. Springer-Verlag

23. Raïssi C, Poncelet P (2007) Sampling for sequential pattern mining: From static databases to data streams. In: Seventh IEEE International Conference on Data Mining, IEEE, ICDM '07, pp 631–636
24. Riondato M, Upfal E (2014) Efficient discovery of association rules and frequent itemsets through sampling with tight performance guarantees. *ACM Trans Knowl Disc from Data* 8(4):20, DOI 10.1145/2629586, URL <http://doi.acm.org/10.1145/2629586>
25. Riondato M, Upfal E (2015) Mining frequent itemsets through progressive sampling with Rademacher averages. In: Proceedings of the 21st ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, KDD '15, pp 1005–1014
26. Riondato M, Vandin F (2014) Finding the true frequent itemsets. In: Zaki MJ, Obradovic Z, Tan P, Banerjee A, Kamath C, Parthasarathy S (eds) Proceedings of the 2014 SIAM International Conference on Data Mining, Philadelphia, Pennsylvania, USA, April 24–26, 2014, SIAM, pp 497–505, DOI 10.1137/1.9781611973440.57, URL <http://dx.doi.org/10.1137/1.9781611973440.57>
27. Riondato M, Vandin F (2018) MiSoSouP: Mining interesting subgroups with sampling and pseudodimension. In: Proc. 24th ACM SIGKDD Int. Conf. Knowl. Disc. and Data Mining, ACM, KDD '18, pp 2130–2139
28. Servan-Schreiber S, Riondato M, Zraggen E (2018) ProSecCo: Progressive sequence mining with convergence guarantees. In: Proceedings of the 18th IEEE International Conference on Data Mining, pp 417–426
29. Shalev-Shwartz S, Ben-David S (2014) *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press
30. Srikant R, Agrawal R (1996) Mining sequential patterns: Generalizations and performance improvements. In: International Conference on Extending Database Technology, Springer, ICDT '96, pp 1–17
31. Toivonen H (1996) Sampling large databases for association rules. In: Proc. 22nd Int. Conf. Very Large Data Bases, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, VLDB '96, pp 134–145
32. Vapnik VN (1998) *Statistical learning theory*. Wiley
33. Vapnik VN, Chervonenkis AJ (1971) On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications* 16(2):264–280, DOI 10.1137/1116025
34. Wang J, Han J, Li C (2007) Frequent closed sequence mining without candidate maintenance. *IEEE Transactions on Knowledge and Data Engineering* 19(8):1042–1056
35. Zeng K, Agarwal S, Dave A, Armbrust M, Stoica I (2015) G-OLA: Generalized online aggregation for interactive analysis on big data. In: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, ACM, pp 913–918
36. Zeng K, Agarwal S, Stoica I (2016) IOLAP: Managing uncertainty for efficient incremental OLAP. In: Proceedings of the 2016 International Conference on Management of Data, ACM, SIGMOD '16, pp 1347–1361
37. Zraggen E, Galakatos A, Crotty A, Fekete JD, Kraska T (2017) How progressive visualizations affect exploratory analysis. *IEEE transactions on visualization and computer graphics* 23(8):1977–1987

Author Biographies



Sacha Servan-Schreiber is a PhD student at MIT CSAIL where his research interests are in applied cryptography, machine learning, and data science. He received his Bachelor of Science with honors from Brown University in 2018 and his work received a best-of-conference award at the 2018 IEEE International Conference on Data Mining (ICDM).



Matteo Riondato is an assistant professor of computer science at Amherst College, and a visiting faculty at Brown University. Previously he was a research scientist at Two Sigma, and a postdoc at Stanford and Brown. He obtained his PhD in computer science from Brown. His research focuses on algorithms for knowledge discovery, data mining, and machine learning: he develops methods to analyze rich datasets, including graphs and time series, as fast as possible and in a statistically sound way. His works received best-of-conference awards at the 2014 SIAM International Conference on Data Mining (SDM), the 2016 ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), and the 2018 IEEE International Conference on Data Mining (ICDM).



Emanuel Zraggen is a postdoctoral associate at MIT CSAIL where he works on interactive systems for data science and automated machine learning. He received his Fachhochschuldiplom in Informatik from HSR Hochschule für Technik Rapperswil in Switzerland and his MS and PhD from Brown University. His main research areas are human-computer interaction, information visualization, and data science. His work received best-of-conference awards at the 2015 International Conference on Very Large Data Bases (VLDB) and the 2018 IEEE International Conference on Data Mining (ICDM).