ProSecCo: Progressive Sequence Mining with Convergence Guarantees

Sacha Servan-Schreiber
Brown University
Providence, RI, USA
Email: aservans@cs.brown.edu

Matteo Riondato
Two Sigma Investments, LP
New York, NY, USA
Email: matteo@twosigma.com

Emanuel Zgraggen
MIT CSAIL
Cambridge, MA, USA
Email: emzg@mit.edu

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

***Abstract*—We present PROSECCO, an algorithm for the progressive mining of frequent sequences from large transactional datasets: it processes the dataset in blocks and outputs, after having analyzed each block, a high-quality approximation of the collection of frequent sequences. 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 it can even be faster than existing state-of-the-art non-progressive algorithms.

I. 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 **LiuH14**) and large (6–12s **ZgraggenGCFK17**) delays between query and response decrease the rate at which users discover insights.

Data exploration tools, such as Vizdom **CrottyGZBK15** 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* **HellersteinHW97**

The intermediate results must be *trustworthy*, i.e., not mislead the user, otherwise she will not be able to make informed decisions. Specifically, *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.

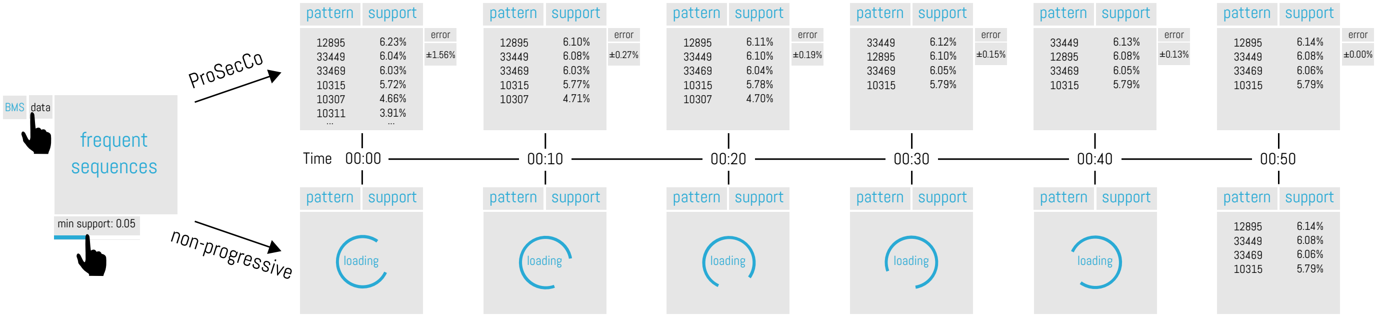
Existing data mining algorithms are poor candidates for this phase of data analysis. "Batch" algorithms that analyze the whole dataset in one shot can take many minutes to complete, thereby disrupting fluid user experiences. Streaming algorithms often do not offer sufficient guarantees on the quality of intermediate results for them to be trustworthy.

In this work we focus on the important task of *frequent sequence mining* **AgrawalS95**, **PeiHMWPCDH04** 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.

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 **PeiHMWPCDH04** 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 actionable information as frequently as possible.

Contributions: We describe PROSECCO, a progressive frequent sequence mining algorithm with trustworthy intermediate results, suitable for interactive data exploration.

- 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 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 that depends on properties of the block. PROSECCO 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 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 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.



pattern	support	error
12895	6.23%	±1.56%
33449	6.04%	
33469	6.03%	
10315	5.72%	
10307	4.66%	
10311	3.91%	
...	...	

pattern	support	error
12895	6.10%	±0.27%
33449	6.08%	
33469	6.03%	
10315	5.77%	
10307	4.71%	

pattern	support	error
12895	6.11%	±0.19%
33449	6.10%	
33469	6.04%	
10315	5.78%	
10307	4.70%	

pattern	support	error
33449	6.12%	±0.15%
12895	6.10%	
33469	6.05%	
10315	5.79%	

pattern	support	error
33449	6.13%	±0.13%
12895	6.08%	
33469	6.05%	
10315	5.79%	

pattern	support	error
12895	6.14%	±0.00%
33449	6.08%	
33469	6.06%	
10315	5.79%	

pattern	support
12895	6.14%
33449	6.08%
33469	6.06%
10315	5.79%

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 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.

Our analysis uses VC-dimension **VapnikC71** and fundamental sample-complexity results from statistical learning theory **LiLS01**, **Vapnik98** 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* (Definition 2), that can be computed in a streaming fashion as the blocks are read (Algorithm 2).

- 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. Furthermore, ProSecCo uses several orders of magnitude less memory when compared to the current state-of-the-art sequent mining algorithm PrefixSpan **PeiHMWPCDH04** and in some cases it is even faster.

II. Related work

Online aggregation **HellersteinHW97** 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 **AcharyaGPR99b**, **AgarwalMPMMS13**, **KamatJTN14**, **CondieCAHES10**, **HellersteinACHORRH99**, **JermaineAPD08**, **PansareBorkarJC11**, **ZengADAS15**, **ZengAS16** 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 **AgrawalS95** 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 **SrikantA96** to prefix-based approaches **PeiHMWPCDH04** to works focusing on the closed frequent sequences **WangHL07** 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 **MendesDH08** 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 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 the mining phase has been successful in sequence mining **RaissiP07** and in other variants of pattern discovery, such as frequent itemsets mining **RiondatoU14**, **RiondatoU15**, **Toivonen96** 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 we actually analyze the whole dataset, although in blocks of transactions in random order, to obtain the *exact* collection of frequent sequences.

III. Preliminaries

Here are the concepts and results used throughout the paper.

A. 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 $S_i \subseteq \mathcal{I}$, $1 \le i \le \ell$.

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., $|\{a,b,c\}| = 3$).

A sequence $\mathbf{a} = \langle A_1, A_2, \ldots, A_\ell \rangle$ 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 $1 \le j_1 < j_2 < \ldots j_\ell \le m$ such that $A_1 \subseteq B_{j_1}, A_2 \subseteq B_{j_2}, \ldots, A_\ell \subseteq B_{j_\ell}$.

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

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

The quantity $2^{\|\mathbf{s}\|} - 1$ is an *upper bound* to $\mathsf{c}(\mathbf{s})$. ProSecCo uses a stronger upper bound introduced later.

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 $\mathbf{s}$ belong:

$$\mathsf{f}_{\mathcal{D}}(\mathbf{s}) = \frac{|\{\tau \in \mathcal{D} \ : \ \mathbf{s} \sqsubseteq \tau\}|}{|\mathcal{D}|} \ . \tag{2}$$

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

$$\begin{gathered}\langle \{a\}, \{b,c\}, \{c,d,e\} \rangle \\ \langle \{a\}, \{d,e\}, \{c,d\} \rangle \\ \langle \{b,d,e\}, \{a,b\} \rangle \\ \langle \{b\}, \{c\}, \{d,e\} \rangle \\ \langle \{a\}, \{a,c\}, \{b\} \rangle \ .\end{gathered} \tag{3}$$

The last transaction is a sequence with length $|\tau| = 3$. Its item-length $\|\tau\|$ is 4. Its capacity $\mathsf{c}(\tau)$ 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 $\langle \{a\} \rangle$, which is 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 $\mathsf{FS}(\mathcal{D},\theta)$ of *frequent sequences in* $\mathcal{D}$ *w.r.t.* θ contains all and only the sequences with frequency at least θ in $\mathcal{D}$:

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

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 $|f_{\mathbf{s}} - \mathsf{f}_{\mathcal{D}}(\mathbf{s})| \le \varepsilon/2$.

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

B. VC-dimension and sampling

The (empirical) Vapnik-Chervonenkis (VC) dimension **VapnikC71** is a fundamental concept from statistical learning theory **Vapnik98** 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 **ShalevSBD14** 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 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 $\mathcal{W}$ 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.,

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

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 is 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 $\phi \in (0,1)$. The set $\mathcal{W}$ is a *ϕ-sample* iff

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

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.

*Theorem 1 (***LiLS01***):* 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 IV-C) to ensure that the intermediate results it outputs have strong quality guarantees and converge to $\mathsf{FS}(\mathcal{D}, \theta)$.

IV. Algorithm

We now present ProSecCo, our *progressive* algorithm for computing the set of frequent sequences in a dataset.

A. Intuition and Motivation

ProSecCo processes the dataset in blocks $B_1, \ldots, B_{\lceil|\mathcal{D}|/b\rceil}$ 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 ProSecCo.

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. Streaming algorithms for frequent sequence mining **MendesDH08** 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 $\mathsf{FS}(\mathcal{D}, \theta)$ (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 $\mathsf{FS}(\mathcal{D}, \theta)$. 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 $\tilde{\mathsf{c}}(\tau) \geq \mathsf{c}(\tau)$.

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

B. Algorithm description

We first need some preliminary definitions and results.

ProSecCo 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 $\mathsf{c}(\tau)$ of a transaction τ is expensive. We instead compute an *upper bound* $\tilde{\mathsf{c}}(\tau) \geq \mathsf{c}(\tau)$ as follows. Consider the quantity $2^{\|\tau\|} - 1 \geq \mathsf{c}(\tau)$. 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 $\mathsf{s} = \langle C \rangle$ 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 $\mathsf{c}(\tau)$ is to avoid over-counting the $2^{|A|} - 1$ sub-sequences of τ in the form of 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 $\tau = \langle Z_1, \ldots, Z_\ell \rangle$ be a transaction and assume to *re-label* the itemsets in τ by *decreasing size*, ties broken arbitrarily, as $A_1, \ldots, A_\ell$, so that $|A_i| \geq |A_{i+1}|$. We compute the upper bound $\tilde{\mathsf{c}}(\tau)$ as follows (pseudocode in Algorithm 1). First, $\tilde{\mathsf{c}}(\tau)$ is set to $2^{\|\tau\|} - 1$, then we put 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 $\tilde{\mathsf{c}}(\tau)$ by $2^{|B|} - 1$ and remove B from L. The following result is then straightforward.

[1] With the possible exception of the last block, which may have fewer than b transactions.

[2] Some additional care is needed when handling the initial block. See Section IV-D.

Lemma 1: It holds that $\tilde{\mathsf{c}}(\tau) \geq \mathsf{c}(\tau)$.

There are many other types of sub-sequences of τ that may be over-counted, but one has to strike the right trade-off between the time it takes to identify the over-counted features 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.

Given a set $\mathcal{W}$ of transactions, we use the upper bounds $\tilde{\mathsf{c}}$ to define a characteristic quantity of $\mathcal{W}$, which we call the *s-index of* $\mathcal{W}$.

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 upper bound $\tilde{\mathsf{c}}$ to their capacities at least $2^d - 1$, 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.

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

Because of its use of $\tilde{\mathsf{c}}$, 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 frequent itemsets mining **RiondatoU14**

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 start with $d = 0$ and increase it progressively by looking at the transactions in $\mathcal{W}$ one by one, maintaining the set $\mathcal{T}$ of $\ell \leq d$ transactions with $\tilde{\mathsf{c}}$ greater than $2^d - 1$ and of $d - \ell$ transactions with $\tilde{\mathsf{c}}$ exactly $2^d - 1$.

```
Algorithm 2: getSIndexBound
input : transaction set $\mathcal{W}$
output: upper bound to the s-index of $\mathcal{W}$.
1 $\mathcal{T} \leftarrow \emptyset$
2 $d \leftarrow 1$
3 foreach $\tau \in \mathcal{W}$ do
4     $\tilde{\mathsf{c}}(\tau) \leftarrow$ getCapBound($\tau$) // See Alg. 1
5     if $\tilde{\mathsf{c}}(\tau) > 2^d - 1$ and $\neg\exists \rho \in \mathcal{T}$ s.t. $\tau \sqsubseteq \rho$ then
6         $\mathcal{Z} \leftarrow \mathcal{T} \cup \{\tau\}$
7         $d \leftarrow$ largest integer such that $\mathcal{Z}$ contains at least
          $d$ transactions of with $\tilde{\mathsf{c}}$ at least $2^d - 1$
8         $\mathcal{T} \leftarrow$ set of $d$ transactions from $\mathcal{Z}$ with $\tilde{\mathsf{c}}$ at least
          $2^d - 1$
9 return $d$
```

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* $B_1, \ldots, B_\beta$ where $\beta = \lceil |\mathcal{D}|/b \rceil$, of b transactions each,[3] 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 **Olken93** 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 **PeiHMWPCDH04** can be used to obtain this set. We explain the expression for ε (line 3) in Section IV-C.

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 `getSIndexBound` (Algorithm 2) is a *streaming* algorithm, so by keeping in memory the set $\mathcal{T}$ (line 8 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
$$f_{\mathbf{s}} \leftarrow \frac{f_{\mathbf{s}}(i-1)b + |\{\tau \in B_i \ : \ \mathbf{s} \sqsubseteq \tau\}|}{i \cdot b}, \tag{5}$$
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 IV-C. When processing the last block B_β, ProSecCo uses $\varepsilon = 0$.

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 IV-C.

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

[3] 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 3: ProSecCo

```
input : dataset $\mathcal{D}$, block size $b$, minimum frequency
        threshold $\theta$, failure probability $\delta$.
output: a set $\mathcal{F}$ which, with probability at least $1-\delta$,
        equals $\mathsf{FS}(\mathcal{D},\theta)$.
$\beta \leftarrow \lceil |\mathcal{D}|/b \rceil$ // Number of blocks
$(B_1, d) \leftarrow$ readBlockAndUpdateSIndex($b, i$)
$\varepsilon \leftarrow 2\sqrt{\frac{d-\ln(\delta)+\ln(\beta-1)}{2b}}$
$\xi \leftarrow \theta - \frac{\varepsilon}{2}$   // Computes lowered threshold
$\mathcal{F} \leftarrow$ getFS($B_1, \xi$) // Computes $\mathsf{FS}(B_i, \xi)$
returnIntermediateResult $(\mathcal{F}, \varepsilon)$
foreach $i \leftarrow 2, \ldots, \beta-1$ do
    $(B_i, d) \leftarrow$ readBlockAndUpdateSIndex($b, i$)
    $\varepsilon \leftarrow 2\sqrt{\frac{d-\ln(\delta)+\ln(\beta-1)}{2i\cdot b}}$
    $\xi \leftarrow \theta - \frac{\varepsilon}{2}$
    $\mathcal{F} \leftarrow$ updateRunningSet($\mathcal{F}, B_i, \xi$)
    returnIntermediateResult $(\mathcal{F}, \varepsilon)$
$(B_\beta, s) \leftarrow$ readBlockAndUpdateSIndex($b, \beta$)
$\mathcal{F} \leftarrow$ updateRunningSet($\mathcal{F}, B_\beta, \theta$)
return $(\mathcal{F}, 0)$
```

C. 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] $1 \le i \le \beta$. It holds that

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

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

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:

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

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 $\mathcal{D}$:

$$\frac{|R_\mathbf{s}|}{|\mathcal{D}|} = \mathsf{f}_\mathcal{D}(\mathbf{s}) \enspace . \tag{7}$$

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

$$\frac{|R_\mathbf{s} \cap \mathcal{W}|}{|\mathcal{W}|} = \mathsf{f}_\mathcal{W}(\mathbf{s}) \enspace . \tag{8}$$

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

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

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 (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

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

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 $\mathcal{B}$.

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 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 $\mathcal{W}$: $d \le \mathsf{EVC}(\mathcal{D},\mathcal{R},\mathcal{W})$.

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 $\tilde{\mathsf{c}}(\tau) \le 2^d - 1$. The transaction τ belongs to 2^{z-1} subsets of $\mathcal{S}$. We label these subsets arbitrarily as A_i, $1 \le i \le 2^{z-1}$. Since $\mathcal{S}$ is shattered by $\mathcal{R}$, for each A_i there must be a range $R_i \in \mathcal{R}$ such that

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

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 $1 \le i \le 2^{z-i}$, 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 $\tilde{\mathsf{c}}(\tau) \le 2^d - 1$, 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. ■

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

Proof of Theorem 2: Recall that $\mathcal{W}_i = \bigcup_{j=1}^{i} B_i$ 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, $1 \le i \le \beta$ and for any pair $(\mathbf{s}, f_\mathbf{s}) \in \mathcal{F}_i$, it holds that

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

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

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

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

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

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

By construction, each $\mathcal{W}_i$ is an uniform random sample of $\mathcal{D}$ of size $b \cdot i$, $1 \le i < \beta$. 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

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

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

D. 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 **PeiHMWPCDH04** 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.

E. 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

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

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 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 $\mathsf{f}_{B_1}^{(k)}$, the frequency in B_1 of the k-th most frequent sequence in B_1:

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

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 (5), 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.

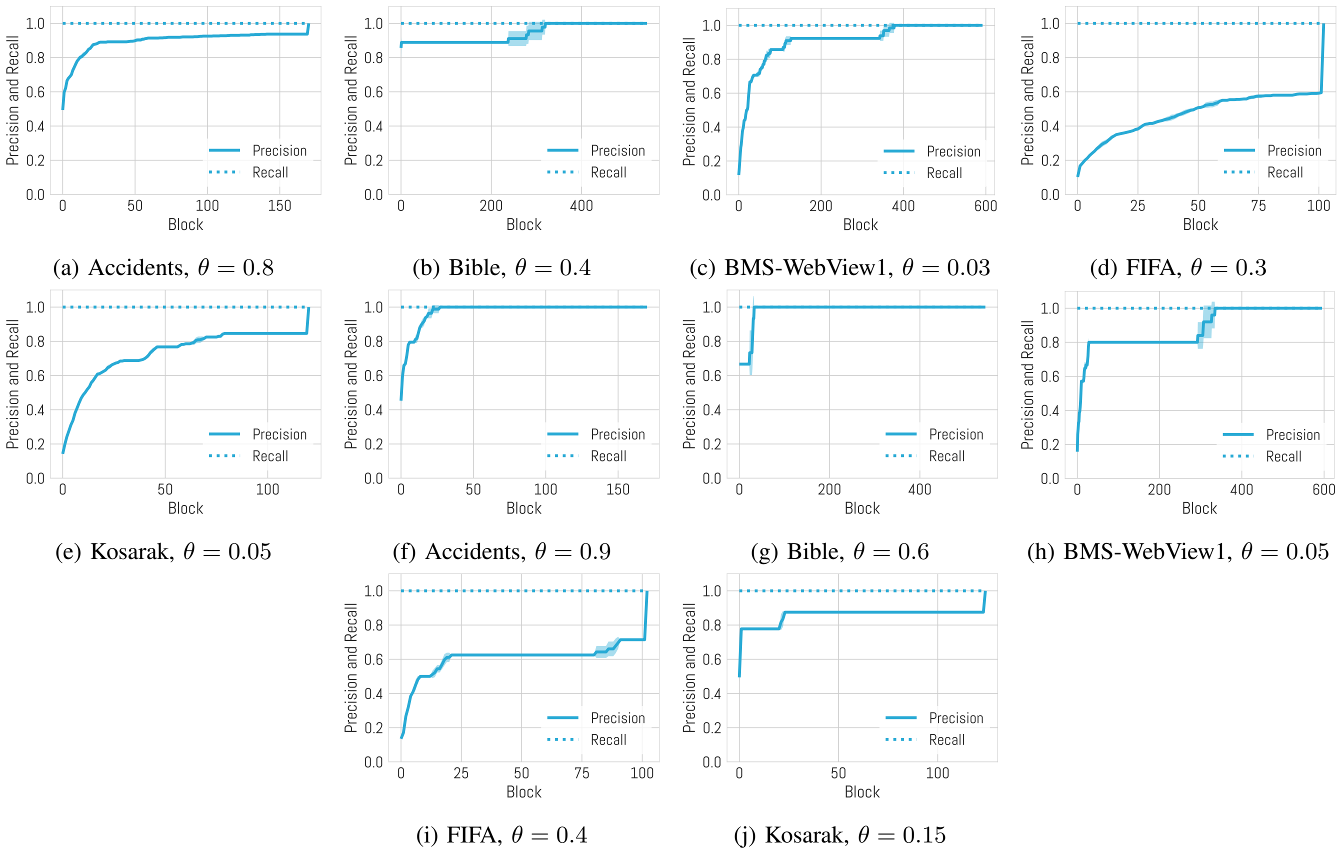
Theorem 3: Let $(\mathcal{F}_i, \varepsilon_i)$ be the i-th pair sent in output by ProSeK, $1 \le i \le \beta$. With probability at least $1 - \delta$, it holds that, for all i, $\mathcal{F}_i$ is an ε_i-approximation to $\mathsf{TOPK}(\mathcal{D}, k)$.

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

F. Discussion

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

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 V-C), 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.



(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) Accidents, $\theta = 0.9$

(g) Bible, $\theta = 0.6$

(h) BMS-WebView1, $\theta = 0.05$

(i) FIFA, $\theta = 0.4$

(j) Kosarak, $\theta = 0.15$

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

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 V-C). 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 is a major benefit of ProSecCo, given the impracticality of using existing sequence mining algorithms on huge sequence datasets.

V. Experimental evaluation

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

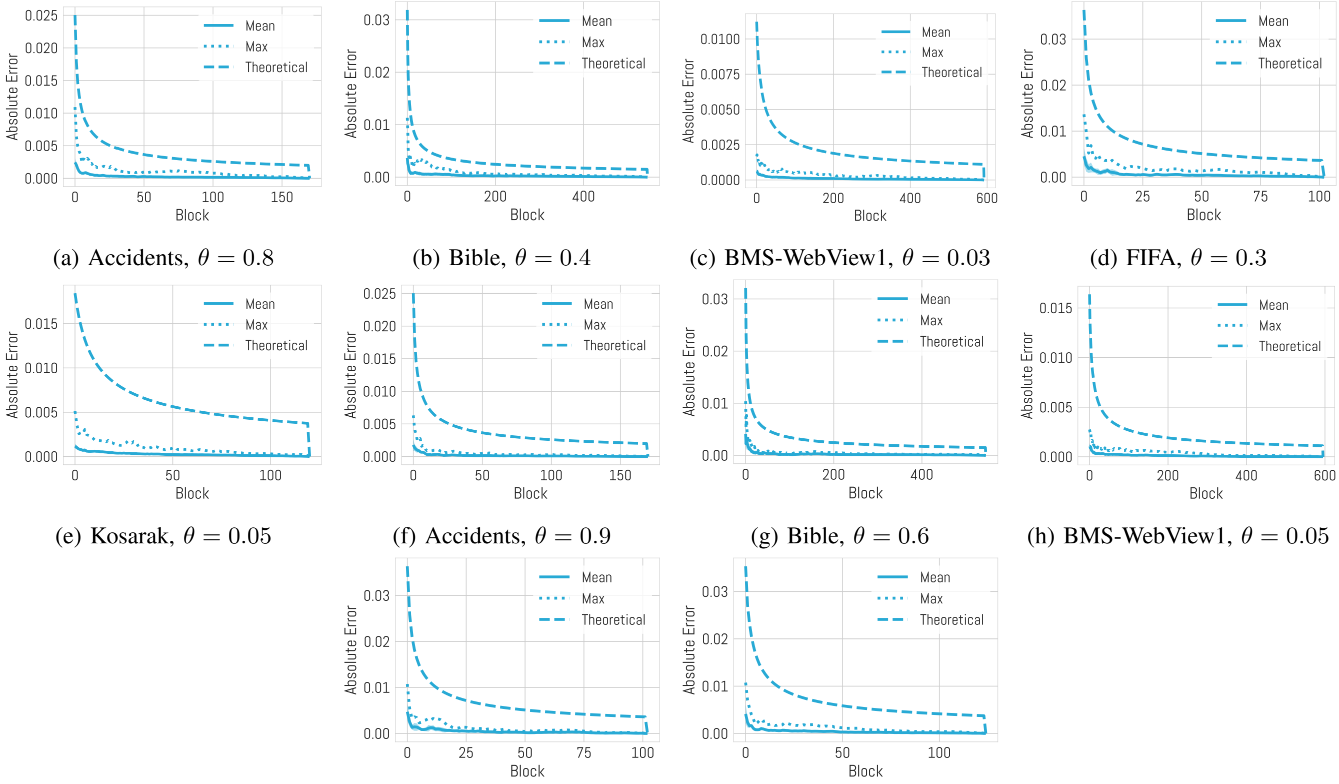
The goals of the evaluation are the following:

- 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 **PeiHMWPCDH04**
- Evaluate the memory usage of ProSecCo over time and compare it with that of PrefixSpan, especially as function of the size of the dataset.

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 initial block (line 5 of Algorithm 3) and for updating this set when processing the successive blocks (line 11). 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 five sequence mining datasets from the SPMF Data Mining Repository **SPMF**



(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) Accidents, $\theta = 0.9$

(g) Bible, $\theta = 0.6$

(h) BMS-WebView1, $\theta = 0.05$

(i) FIFA, $\theta = 0.4$

(j) Kosarak, $\theta = 0.15$

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

- **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;
- **Kosarak:** Click-stream dataset from a Hungarian on-line news portal;

The characteristics of the datasets are reported in Table I. 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 100). The replication preserves the original distribution of sequence frequencies and transaction lengths, so it does not advantage PROSECCO in any way, nor disadvantages any other sequence mining algorithm.

TABLE I: Dataset characteristics

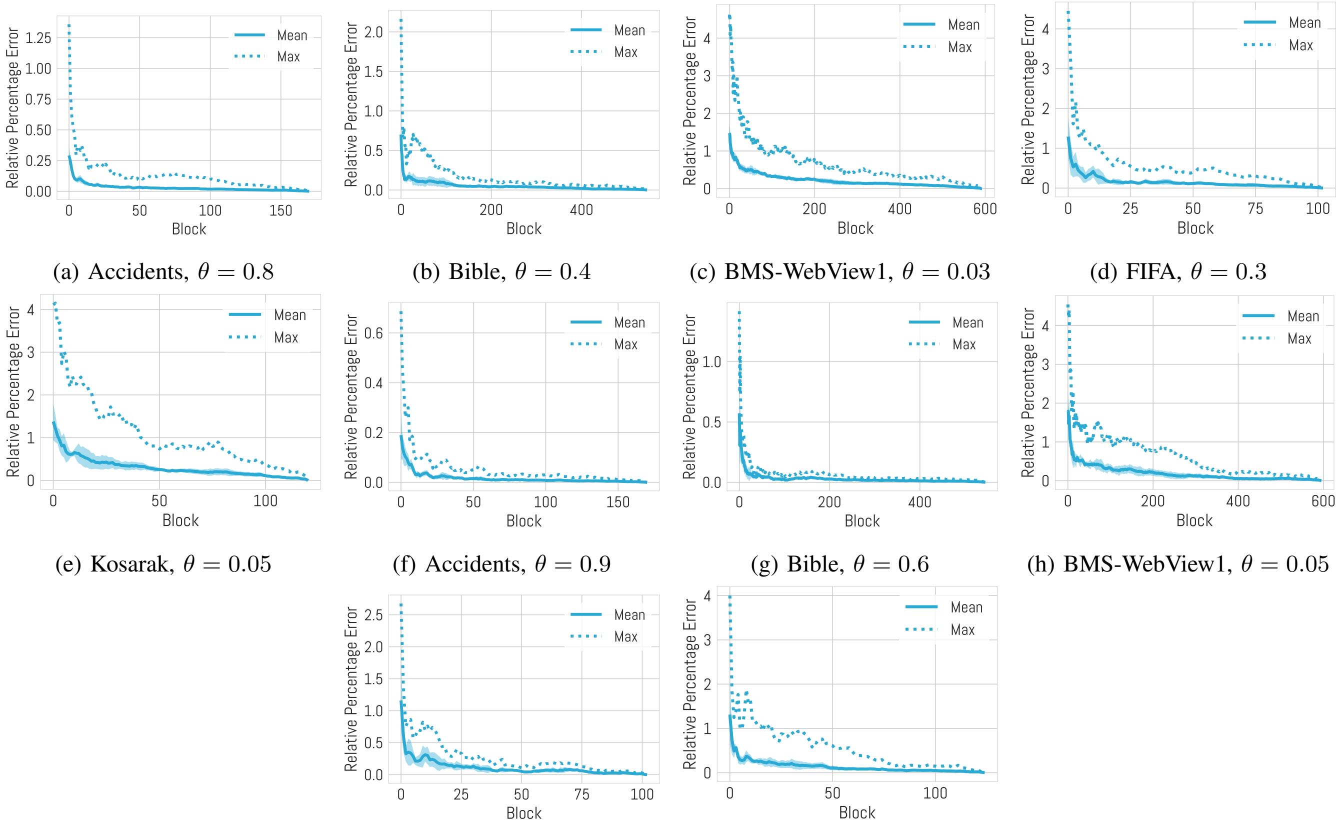
Dataset	Size ($\lvert\mathcal{D}\rvert$)	Repl. Factor	$\lvert\mathcal{I}\rvert$	Avg. trans. size
Accidents	1700915	5x	481	34.8
Bible	5455350	200x	14442	22.6
BMS-WebView1	5960001	100x	938	3.5
FIFA	1022500	50x	4153	37.2
Kosarak	1249951	50x	16428	9.0

Parameters: We test PROSECCO using a number of different minimum frequency thresholds on each dataset. We report, for each dataset, the results for two 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 (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 V-B for a comparison of different blocks sizes).

A. 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



(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) Accidents, $\theta = 0.9$

(g) Bible, $\theta = 0.6$

(h) BMS-WebView1, $\theta = 0.05$

(i) FIFA, $\theta = 0.4$

(j) Kosarak, $\theta = 0.15$

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

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 $\mathsf{FS}(\mathcal{D}, \theta)$.

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 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 ProSecCo may perform slightly differently (shaded region around the precision curve) but still show relatively high precision across the board. The precision tends to plateau after a few blocks: this effect is due to the fact that, before having process 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 $\mathsf{FS}(\mathcal{D}, \theta)$ and they can be safely expunged 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* (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 0.0025) 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. 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 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.

B. 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 it is 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.

Furthermore, 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 thresholds.

The *overall runtime* of ProSecCo is almost identical to (and often faster than) the runtime of PrefixSpan (Figure 6). At times ProSecCo is slower, but we stress that it has been producing high-quality trustworthy 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.

C. 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. 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.

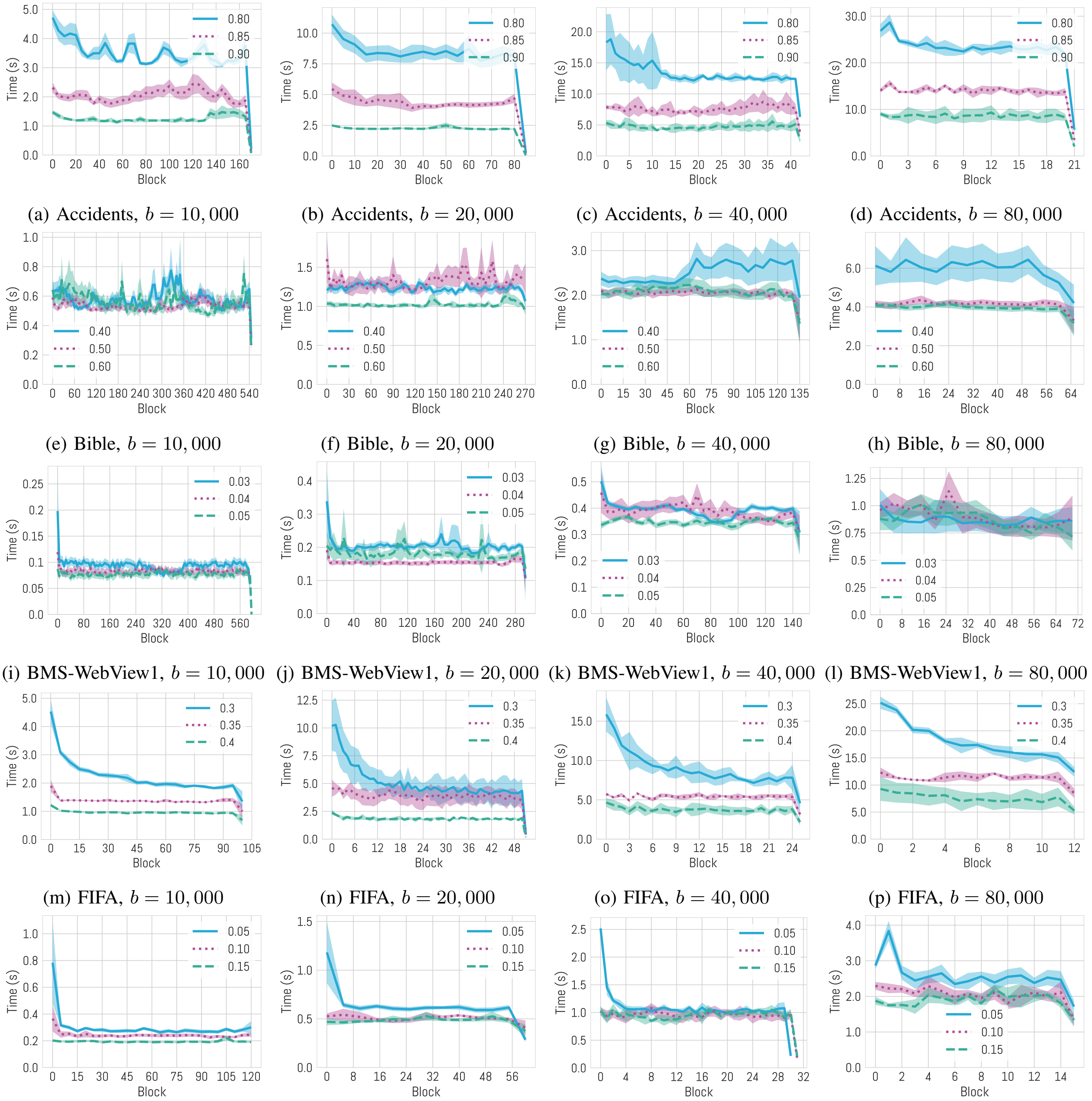
VI. Conclusions

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 $\mathsf{FS}(\mathcal{D}, \theta)$ of frequent sequences, with increasingly high quality. Once all the dataset has been processed, the last result is exactly $\mathsf{FS}(\mathcal{D}, \theta)$.

Each returned approximation comes with strong theoretical guarantees. The analysis uses VC-dimension, a key concept from statistical learning theory.

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 progressively and quickly decreases.

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.



(a) Accidents, $b = 10,000$

(b) Accidents, $b = 20,000$

(c) Accidents, $b = 40,000$

(d) Accidents, $b = 80,000$

(e) Bible, $b = 10,000$

(f) Bible, $b = 20,000$

(g) Bible, $b = 40,000$

(h) Bible, $b = 80,000$

(i) BMS-WebView1, $b = 10,000$

(j) BMS-WebView1, $b = 20,000$

(k) BMS-WebView1, $b = 40,000$

(l) BMS-WebView1, $b = 80,000$

(m) FIFA, $b = 10,000$

(n) FIFA, $b = 20,000$

(o) FIFA, $b = 40,000$

(p) FIFA, $b = 80,000$

(q) Kosarak, $b = 10,000$

(r) Kosarak, $b = 20,000$

(s) Kosarak, $b = 40,000$

(t) Kosarak, $b = 80,000$

Fig. 5: Per-block runtime for different choices of block size b and different frequency thresholds θ.

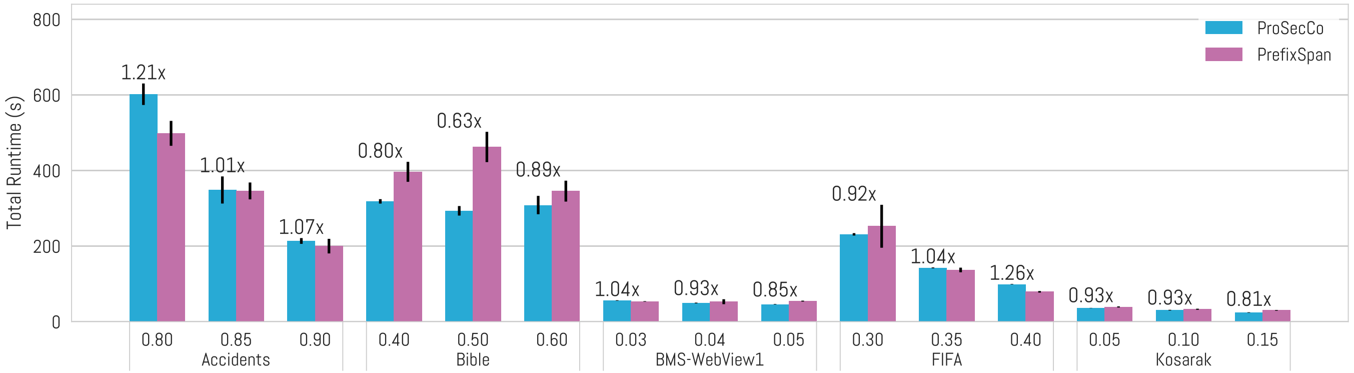
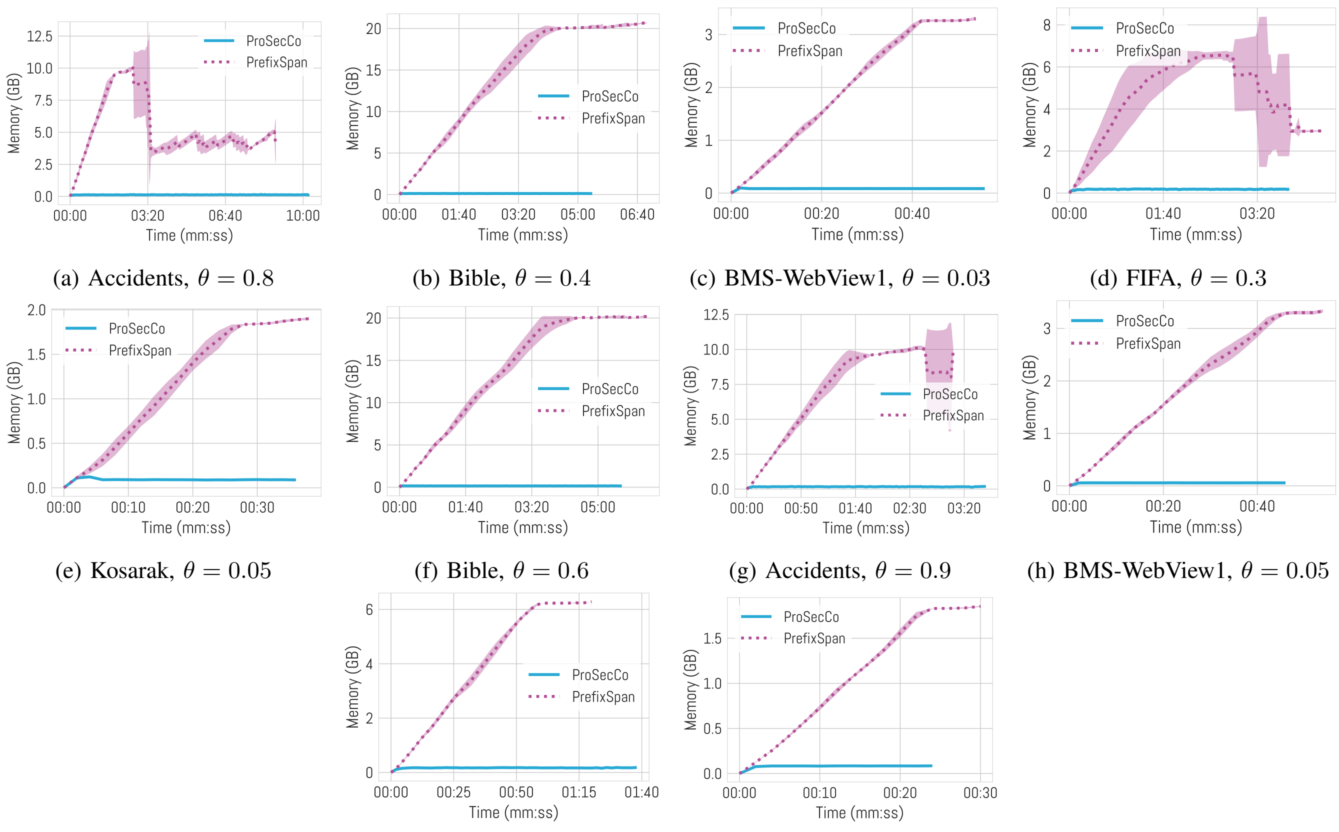


Fig. 6: Total runtime comparision for all experiments between PrefixSpan and ProSecCo including 95% confidence intervals. Numbers on top of bars represent ProSecCo's average runtime as a factor of PrefixSpan's average runtime.



(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) Bible, $\theta = 0.6$

(g) Accidents, $\theta = 0.9$

(h) BMS-WebView1, $\theta = 0.05$

(i) FIFA, $\theta = 0.4$

(j) Kosarak, $\theta = 0.15$

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