Closing the Gap: Sequence Mining at Scale

KAUSTUBH BEEDKAR and KLAUS BERBERICH, Max Planck Institute for Informatics
RAINER GEMULLA, University of Mannheim
IRIS MILIARAKI, Yahoo! Labs

8

Frequent sequence mining is one of the fundamental building blocks in data mining. While the problem has been extensively studied, few of the available techniques are sufficiently scalable to handle datasets with billions of sequences; such large-scale datasets arise, for instance, in text mining and session analysis. In this article, we propose MG-FSM, a scalable algorithm for frequent sequence mining on MapReduce. MG-FSM can handle so-called "gap constraints", which can be used to limit the output to a controlled set of frequent sequences. Both positional and temporal gap constraints, as well as appropriate maximality and closedness constraints, are supported. At its heart, MG-FSM partitions the input database in a way that allows us to mine each partition independently using any existing frequent sequence mining algorithm. We introduce the notion of w-equivalency, which is a generalization of the notion of a "projected database" used by many frequent pattern mining algorithms. We also present a number of optimization techniques that minimize partition size, and therefore computational and communication costs, while still maintaining correctness. Our experimental study in the contexts of text mining and session analysis suggests that MG-FSM is significantly more efficient and scalable than alternative approaches.

Categories and Subject Descriptors: H.2.8 [**Database Management**]: Database Applications—*Data mining*

General Terms: Algorithms, Experimentation, Performance, Theory

Additional Key Words and Phrases: Data mining, frequent sequence mining, MapReduce

ACM Reference Format:
Kaustubh Beedkar, Klaus Berberich, Rainer Gemulla, and Iris Miliaraki. 2015. Closing the gap: Sequence mining at scale. ACM Trans. Datab. Syst. 40, 2, Article 8 (June 2015), 44 pages.
DOI: http://dx.doi.org/10.1145/2757217

1. INTRODUCTION

Frequent sequence mining (FSM) is a fundamental component in a number of important data mining tasks. In text mining, for example, frequent sequences can be used to construct statistical language models for machine translation [Lopez 2008], information retrieval [Zhai 2008], information extraction [Tandon et al. 2011], or spam detection [Kant et al. 2012]. Word associations have also been applied to relation extraction [Nakashole et al. 2011]. In Web-usage mining and session analysis [Srivastava et al. 2000], frequent sequences describe common behavior across users (e.g., frequent sequences of page visits). In these and similar applications, inputs to FSM can get very large and may involve billions of sequences. For example, Microsoft provides access to an n-gram collection based on hundreds of billions of web pages, and Google published a corpus of more than 1 billion n-grams. Similarly, in web companies with millions of

Authors' addresses: K. Beedkar (corresponding author), Data and Web Science Group, University of Mannheim; email: kbeedkar@uni-mannheim.de; K. Berberich, Max Planck Institute for Informatics; R. Gemulla, University of Mannheim; I. Miliaraki, Yahoo! Labs.

0362-5915/2015/06-ART8 $15.00
DOI: http://dx.doi.org/10.1145/2757217

users, the amount of usage data can be substantial. At such massive scales, distributed and scalable FSM algorithms are essential.

Given a collection of input sequences of items, the goal of FSM is to find all subsequences that "appear" in sufficiently many input sequences. In text mining, for example, each input sequence corresponds to a document (or a sentence) and each item to a word in the document. The definition of *appears* is application-dependent; for instance, the goal of *n-gram mining* is to find frequent consecutive word sequences of length n, whereas the goal of *word association mining* is to find combinations of words that frequently appear in close proximity (but not necessarily consecutively). As another example, in session analysis, input sequences correspond to user sessions and items to user actions (often with an additional timestamp). Depending on the application, we may be interested in sequences of either consecutive or nonconsecutive actions that are sufficiently close (e.g., few actions in between or temporally close). This requirement of closeness is addressed by *gap-constrained frequent sequence mining* [Srikant and Agrawal 1996], in which FSM is parameterized with a *maximum-gap parameter* γ. Informally, for a given input sequence, we consider only subsequences that can be generated without skipping more than γ consecutive items. We obtain n-gram mining for $\gamma = 0$, word association mining for (say) $\gamma = 5$, and unconstrained FSM [Zaki 2001b] for $\gamma = \infty$.

In this article, we propose a scalable distributed (i.e., shared-nothing) FSM algorithm called MG-FSM.[1,2] Although scalable algorithms exist for n-gram mining [Huston et al. 2011; Berberich and Bedathur 2013], MG-FSM is the first distributed algorithm that supports general gap constraints. MG-FSM targets MapReduce [Dean and Ghemawat 2004]—which constitutes a natural environment for text mining and analysis of user access logs—but is also amenable to other distributed data processing platforms. At a high level, MG-FSM carefully partitions and rewrites the set of input sequences in such a way that each partition can be mined independently and in parallel. Once partitions have been constructed, an arbitrary gap-constrained FSM algorithm can be used to mine each partition; no postprocessing of results across partitions is needed.

MG-FSM extends the notion of item-based partitioning, which underlies a number of frequent pattern mining algorithms, including FP-growth [Han et al. 2004] as well as the distributed algorithms by Buehrer et al. [2007] and Cong et al. [2005] for frequent itemset mining, to gap-constrained frequent sequence mining. In more detail, we first develop a basic partitioning scheme that ensures correctness but allows for flexible partition construction. This flexibility is captured in our notion of w-equivalency, which generalizes the concept of a "projected database" used by many FSM algorithms. We also propose a number of novel optimization techniques that aim to reduce computational and communication costs, including *minimization* (prunes entire sequences), *reduction* (shortens long sequences), *separation* (splits long sequences), *aggregation* (of repeated sequences), and *lightweight compression*.

We also discuss various extensions to MG-FSM for special application scenarios. First, we show how MG-FSM can efficiently handle datasets in which input sequences are very long. Second, we discuss methods to support temporal sequence mining, in which items are annotated with timestamps. This allows MG-FSM to handle temporal gaps (such as "at most one minute" for session analysis). Finally, we extend MG-FSM to support mining only maximal and closed sequences with small overhead. Such sequences concisely represent the set of all frequent sequences and can significantly reduce the number of frequent sequences being mined with minimal loss of information.

[1]A preliminary version of this article appeared in Miliaraki et al. [2013].
[2]The source code of MG-FSM is publicly available at MG-FSM [2014].

Our experiments, in which we mine databases with more than 1 billion sequences, suggest that MG-FSM is multiple orders of magnitude faster than baseline algorithms for general gap-constrained FSM and is competitive to state-of-the-art algorithms for n-gram mining.

The remainder of this article is organized as follows. In Section 2, we formally define the problem of gap-constrained frequent sequence mining and establish the notation used throughout this work. In Section 3, we outline the MG-FSM algorithm and introduce the notion of w-equivalency on which MG-FSM is based. In Section 4, we describe in more detail the partition construction step of MG-FSM. Section 5 discusses an extension of MG-FSM for mining maximal or closed sequences. Section 6 includes implementation details, as well as a discussion on how to handle long input sequences and temporal sequences. Section 7 describes the results of our experimental study. We discuss related work in Section 8 and conclude in Section 9.

2. PRELIMINARIES

2.1. Problem Statement

A *sequence database* $\mathscr{D} = \{ S_1, \ldots, S_{|\mathscr{D}|} \}$ is a multiset of input sequences.[3] A *sequence* is an ordered list of items from some dictionary $\Sigma = \{ w_1, \ldots, w_{|\Sigma|} \}$.[4] We write $S = s_1 s_2 \cdots s_{|S|}$ to denote a sequence of length $|S|$, where $s_i \in \Sigma$ for $1 \leq i \leq |S|$. Denote by Σ^+ the set of all nonempty sequences constructed with items from Σ. In what follows, we will often use symbol T to refer to an input sequence in the database and symbol S to refer to an arbitrary sequence.

We now formally define the problem of gap-constrained frequent sequence mining considered in this article. We make use of three main parameters which control the set of sequences being mined: the support threshold σ, the maximum-gap parameter γ, and the length threshold λ. In a nutshell, σ controls when a sequence is considered frequent, γ controls whether contiguous sequences ($\gamma = 0$) or sequences with gaps ($\gamma > 0$) are to be mined, and λ places a length restriction on the output sequences. The choice of values for these parameters is application dependent; we give some examples at the end of this section.

Denote by $\gamma \geq 0$ a *maximum-gap parameter*. We say that S is a *γ-subsequence* of T, denoted $S \subseteq_\gamma T$, when S is a subsequence of T and there is a gap of at most γ between consecutive items selected from T. Standard n-grams correspond to 0-subsequences. Formally, $S \subseteq_\gamma T$ if and only if there exist indexes $i_1 < \cdots < i_{|S|}$ such that: (i) $s_k = t_{i_k}$ for $1 \leq k \leq |S|$, and (ii) $i_{k+1} - i_k - 1 \leq \gamma$ for $1 \leq k < |n|$. For example, if $T = abcd$, $S_1 = acd$ and $S_2 = bc$, then $S_1 \subseteq_1 T$ (but $S_1 \not\subseteq_0 T$) and $S_2 \subseteq_0 T$.

The *γ-support* $\mathrm{Sup}_\gamma(S, \mathscr{D})$ of S in database $\mathscr{D}$ is given by the multiset

$$\mathrm{Sup}_\gamma(S, \mathscr{D}) = \{T \in \mathscr{D} : S \subseteq_\gamma T\}.$$

Denote by $f_\gamma(S, \mathscr{D}) = |\mathrm{Sup}_\gamma(S, \mathscr{D})|$ the *γ-frequency* of sequence S. Our measure of frequency corresponds to the notion of *document frequency* in text mining, that is, we count the number of input sequences (documents) in which S occurs (as opposed to the total number of occurrences of S). For support threshold $\sigma > 0$, we say that sequence S is (σ, γ)-frequent if $f_\gamma(S, \mathscr{D}) \geq \sigma$. Since many applications of FSM need to consider

[3]We indicate both sets and multisets using { }; the appropriate type is always clear from the context. The operators $\uplus$, $\overset{\cap}{}$, and $\setminus^+$ correspond to multiset union, multiset intersection, and multiset difference.

[4]A more general variant of this problem is often considered in the literature, in which sequences are formed of itemsets rather than of individual items. We focus on the important special case of individual items in this article (e.g., textual data, user sessions, event logs). Our methods can potentially be used to mine sequences of itemsets as well; see the discussion in Section 6.6.

only short frequent sequences, we introduce an additional *length parameter* λ; only sequences of length at most λ are output.

The gap-constrained frequent sequence mining problem considered in this article is as follows.

> Given a support threshold $\sigma \geq 1$, a maximum-gap parameter $\gamma \geq 0$, and a length threshold $\lambda \geq 2$, find the set $F_{\sigma,\gamma,\lambda}(\mathscr{D})$ of all (σ, γ)-frequent sequences in $\mathscr{D}$ of length at most λ. For each such sequence, also compute its frequency $f_\gamma(S, \mathscr{D})$.

For database $\mathscr{D} = \{abcaaabc, abcbbabc, abcccabc\}$, we obtain $F_{3,0,2}(\mathscr{D}) = \{a, b, c, ab, bc\}$, $F_{3,1,2}(\mathscr{D}) = \{a, b, c, ab, ac, bc\}$, and $F_{3,2,2}(\mathscr{D}) = \{a, b, c, ab, ac, bc, ca\}$.

The appropriate choice of parameter values depends on the application. The Google n-gram corpus [Brants and Franz 2006], for example, consists of sequences of at most 5 words that occur contiguously at least 40 times in a large document corpus; this setup corresponds to $\sigma = 40$, $\gamma = 0$, and $\lambda = 5$. A slight increase in γ lifts the contiguity requirement; this is useful for word association mining and allows to find frequent phrases that do not necessarily occur consecutively (e.g., "celebrate birthday" occurs nonconsecutively in "They celebrate his 23rd birthday." for $\gamma = 2$). In other applications, much larger values of γ may arise. For example, consider the problem of mining frequent sequences of user actions (e.g., buying a product, visiting a webpage, listening to a song) from log files; here successive actions should occur in close temporal proximity (say, within hours, days, or weeks). As we describe in Section 6.6, such temporal gap constraints can be rewritten into positional gap constraints as defined before (which are then often large).

2.2. MapReduce

MapReduce, developed by Dean and Ghemawat [2004] at Google, is a popular framework for distributed data processing on clusters of commodity hardware. It operates on key-value pairs and allows programmers to express their problem in terms of a *map* and a *reduce* function. Key-value pairs emitted by the map function are partitioned by key, sorted, and input into the reduce function. An additional *combine* function can be used to pre-aggregate the output of the map function and increase efficiency. The MapReduce runtime takes care of execution and transparently handles failures in the cluster. While originally proprietary, open-source implementations of MapReduce, most notably Apache Hadoop, are available and have gained widespread adoption.

2.3. Naïve Approach

A naïve approach to gap-constrained FSM in MapReduce modifies WordCount, which determines how often every word occurs in a document collection and is often used to explain how MapReduce works, as follows. In the map function, which is invoked on each input sequence, we emit all distinct γ-subsequences of length at most λ that occur in the input sequence. In the reduce function, we count how often every subsequence S has occurred, thus determining $f_\gamma(S, \mathscr{D})$, and emit it if frequent. The method is generally inefficient and not scalable since it creates and communicates large intermediate data: For example, if $|S| = n$ and $\lambda \geq n$, the naïve approach emits per input sequence $O(n^2)$ key-value pairs for $\gamma = 0$ and $O(2^n)$ key-value pairs for $\gamma \geq n$.

3. THE MG-FSM ALGORITHM

The idea behind item-based partitioning is to create one *partition* $\mathscr{P}_w$ for every σ-frequent item $w \in \Sigma$; we refer to w as the *pivot item* of partition $\mathscr{P}_w$. In the context of frequent itemset mining, item-based partitioning is exploited in the well-known FP-growth algorithm [Han et al. 2004] as well as the distributed frequent itemset

ALGORITHM 1: The MG-FSM algorithm

Require: Sequence database $\mathscr{D}$, σ, γ, λ, f-list $F_{\sigma,0,1}(\mathscr{D})$
1: MAP(T):
2: **for all** distinct $w \in T$ s.t. $w \in F_{\sigma,0,1}(\mathscr{D})$ **do**
3: Construct a sequence database $\mathscr{P}_w(T)$ that is (w, γ, λ)-equivalent to $\{T\}$
4: For each $S \in \mathscr{P}_w(T)$, output (w, S)
5: **end for**
6:
7: REDUCE(w, $\mathscr{P}_w$):
8: $F_{\sigma,\gamma,\lambda}(\mathscr{P}_w) \leftarrow \text{FSM}_{\sigma,\gamma,\lambda}(\mathscr{P}_w)$
9: **for all** $S \in F_{\sigma,\gamma,\lambda}(\mathscr{P}_w)$ **do**
10: **if** $p(S) = w$ and $S \neq w$ **then**
11: Output $(S,\ f_\gamma(S, \mathscr{P}_w))$
12: **end if**
13: **end for**

miners of Buehrer et al. [2007] and Li et al. [2008]. In our setting of frequent sequence mining, partition $\mathscr{P}_w$ is itself a sequence database (also called *projected database*) and captures relevant information about (some) frequent sequences containing pivot w. We first describe the MG-FSM algorithm in general. MG-FSM is based on the notion of w-equivalency which we introduce in Section 3.3. In particular, w-equivalency is a necessary and sufficient condition for the correctness of MG-FSM, which we establish in Section 3.4.

3.1. Algorithm Overview

MG-FSM is divided into a preprocessing phase, a partitioning phase, and a mining phase; all phases are fully parallelized. In the *preprocessing phase*, we gather basic statistics about the data. In the *partitioning phase*, we construct w-equivalent partitions for all frequent items w in Σ. Each of these partitions is mined independently and in parallel in the *mining phase* using an FSM algorithm of choice. The final output is obtained by filtering the output of the FSM algorithm locally at each partition. MG-FSM is given as Algorithm 1; the notation is described next and in Section 3.3.

Preprocessing. In the preprocessing phase, we compute the frequency of each item $w \in \Sigma$ and construct the set $F_{\sigma,0,1}(\mathscr{D})$ of frequent items, commonly called *f-list*. This can be done efficiently in a single MapReduce job (by running a version of WORDCOUNT that ignores repeated occurrences of items within an input sequence). At this point, we can already output the set of frequent sequences of length 1; we subsequently focus on sequences of length 2 and above.

In MG-FSM, we use the f-list to establish a total order $<$ on Σ: Set $w < w'$ if $f_0(w, \mathscr{D}) > f_0(w', \mathscr{D})$; ties are broken arbitrarily. Thus items are ordered by decreasing frequency. Write $S \leq w$ if $w' \leq w$ for all $w' \in S$ and denote by $\Sigma^+_{\leq w} = \{S \in \Sigma^+ : w \in S, S \leq w\}$ the set of all sequences that contain w but no items larger than w. Finally, denote by $p(S) = \min_{w \in S}(S \leq w)$ the *pivot item* of sequence S, that is, the largest item in S. Note that $p(S) = w \iff w \in S \wedge S \leq w \iff S \in \Sigma^+_{\leq w}$. For example, when $S = abc$, then $S \leq c$ and $p(S) = c$; here, as well as in all subsequent examples, we assume order $a < b < c < \cdots$.

Partitioning phase (map). The partitioning and mining phases of MG-FSM are performed in a single MapReduce job. We construct partitions $\mathscr{P}_w$ in the map phase: For each distinct item w in each input sequence $T \in \mathscr{D}$, we compute a small sequence database $\mathscr{P}_w(T)$ and output each of its sequences with reduce key w. We require $\mathscr{P}_w(T)$ to be "(w, γ, λ)-equivalent" to T; see Section 3.3. For now, assume that $\mathscr{P}_w(T) = \{T\}$; a key contribution of this article lies in the refined construction of $\mathscr{P}_w(T)$.

Mining phase (reduce). The input to the mining phase, which is run in the reduce step, is given by

$$\mathscr{P}_w = \biguplus_{T \in \mathscr{D}, w \in T} \mathscr{P}_w(T),$$

which is automatically constructed by the MapReduce framework. We run an arbitrary FSM algorithm with parameters σ, γ, and λ on $\mathscr{P}_w$ (denoted $\mathrm{FSM}_{\sigma,\gamma,\lambda}(\mathscr{P}_w)$ in Algorithm 1) to obtain the frequent sequences $F_{\sigma,\gamma,\lambda}(\mathscr{P}_w)$ as well as their frequencies. Since every frequent sequence may be generated at multiple partitions, we filter the output of the FSM algorithm in order to produce each frequent sequence exactly once. In particular, we output sequence S at partition $\mathscr{P}_{p(S)}$, that is, at the partition corresponding to its largest item. Observe that with our choice of $\mathscr{P}_w(T) = \{T\}$, $f_\gamma(S, \mathscr{P}_w) = f_\gamma(S, \mathscr{D})$ for all sequences S with $w \in S$, so that MG-FSM produces the correct result.

MG-FSM is reminiscent of the distributed frequent itemset mining algorithms of Buehrer et al. [2007] and Li et al. [2008]; the key difference lies in partition construction (line 3 of Algorithm 1), that is, in our notion of w-equivalency.

3.2. Example

Consider the database

$$\mathscr{D}_{\mathrm{ex}} = \{acb, dacbd, dacbddbca, bd, bcaddbd, addcd\} \tag{1}$$

and pivot c. Under our running assumption, we set $\mathscr{P}_c(T) = \{T\}$ if $c \in T$ and $\mathscr{P}_c(T) = \emptyset$ otherwise. We thus obtain

$$\mathscr{P}_c = \{acb, dacbd, dacbddbca, bcaddbd, addcd\}. \tag{2}$$

With such a partitioning, $\mathscr{P}_c$ is large so that there is substantial communication cost. Moreover, the FSM algorithm run on $\mathscr{P}_c$ in the mining phase produces a large number of sequences that do not pass the subsequent filter. For example, $F_{1,1,3}(\mathscr{P}_c)$ contains sequences da, dab, add, and so on, all of which are filtered out (and, in fact, also produced at partition $\mathscr{P}_d$). Such redundant computation is wasteful in terms of computational cost. In what follows, we introduce the concept of w-equivalency which will allow us to significantly reduce both communication and computational costs.

3.3. *w*-Equivalency

As mentioned earlier, w-equivalency is a necessary and sufficient condition for the correctness of MG-FSM; the flexibility implied by w-equivalency forms the basis of our partition construction algorithms of Section 4.

Say that a sequence S is a *pivot sequence* with respect to $w \in \Sigma$ if $p(S) = w$ and $2 \leq |S| \leq \lambda$. Denote by

$$G_{w,\gamma,\lambda}(T) = \left[F_{1,\gamma,\lambda}(\{T\}) \cap \Sigma^+_{\leq w}\right] \setminus \{w\}$$

the set of pivot sequences that occur in T, that is, are γ-subsequences of T. If $S \in G_{w,\gamma,\lambda}(T)$, we say that T (w, γ, λ)-generates (or simply w-generates) S. For example,

$$G_{c,1,2}(acbfdeacfc) = \{ac, cb, cc\}.$$

Recall that our choice of $<$ is based on the f-list, which ultimately aims to reduce variance in partition sizes: Partitions corresponding to highly frequent items are affected by many input sequences, but each input sequence generates few pivot sequences (e.g., a does not generate any pivot sequence in the previous example). In contrast, partitions corresponding to less frequent pivot items are affected by few input sequences, but each input sequence generates many pivot sequences (e.g., f).

We also extend the previous definition to sequence databases as follows:

$$G_{w,\gamma,\lambda}(\mathscr{D}) = \biguplus_{T \in \mathscr{D}} G_{w,\gamma,\lambda}(T). \tag{3}$$

Note that $G_{w,\gamma,\lambda}(T)$ is a set, whereas $G_{w,\gamma,\lambda}(\mathscr{D})$ is a multiset. This difference is a consequence of our use of document frequency, that is, we generate each subsequence at most once per input sequence but potentially many times per sequence database. We are now ready to define w-equivalency.

Definition 3.1. Two sequence databases $\mathscr{D}$ and $\mathscr{P}_w$ are (w, γ, λ)-*equivalent* (or simply w-*equivalent*) if and only if

$$G_{w,\gamma,\lambda}(\mathscr{D}) = G_{w,\gamma,\lambda}(\mathscr{P}_w).$$

Both databases thus generate the same (multiset of) pivot sequences.

Continuing the example of Section 3.2, observe that $\mathscr{P}_c$ as given in Eq. (2) is $(c, 1, 3)$-equivalent to $\mathscr{D}$ (see the discussion at the end of this section). However, so is partition

$$\mathscr{P}'_c = \{acb, acb, acb, bca, bca\}, \tag{4}$$

which is significantly smaller and contains many repeated sequences. In Section 4, we present a number of rewriting techniques that ultimately produce $\mathscr{P}'_c$ as given before.

The following lemma establishes that a w-equivalent database retains the frequency (with respect to γ) of pivot sequences; this property is exploited by our MG-FSM algorithm.

LEMMA 3.2. *If $\mathscr{D}$ and $\mathscr{P}_w$ are (w, γ, λ)-equivalent, then*

$$f_\gamma(S, \mathscr{P}_w) = f_\gamma(S, \mathscr{D})$$

for all $S \in \Sigma_w^+$ such that $2 \le |S| \le \lambda$.

PROOF. Denote by $f(T, \mathscr{A})$ the frequency of T in multiset $\mathscr{A}$. Note that f counts input sequences, whereas f_γ counts γ-subsequences. Pick an arbitrary pivot sequence S, that is, $p(S) = w$ and $2 \le |S| \le \lambda$. Since $\mathscr{P}_w$ is (w, γ, λ)-equivalent to $\mathscr{D}$, we have

$$\begin{aligned}
& G_{w,\gamma,\lambda}(\mathscr{P}_w) = G_{w,\gamma,\lambda}(\mathscr{D}) \\
\Longrightarrow\ & f(S, G_{w,\gamma,\lambda}(\mathscr{P}_w)) = f(S, G_{w,\gamma,\lambda}(\mathscr{D})) \\
\Longleftrightarrow\ & |\{\, T \in \mathscr{P}_w : S \in G_{w,\gamma,\lambda}(T) \,\}| \\
& \quad = |\{\, T \in \mathscr{D} : S \in G_{w,\gamma,\lambda}(T) \,\}| \\
\Longleftrightarrow\ & |\{\, T \in \mathscr{P}_w : S \subseteq_\gamma T \,\}| = |\{\, T \in \mathscr{D} : S \subseteq_\gamma T \,\}| \\
\Longleftrightarrow\ & f_\gamma(S, \mathscr{P}_w) = f_\gamma(S, \mathscr{D}),
\end{aligned}$$

where we applied the definition of $G_{w,\gamma,\lambda}(\mathscr{D})$ and the fact that $S \subseteq_\gamma T$ if and only if $S \in G_{w,\gamma,\lambda}(T)$ for our choice of S. □

Observe that the frequency $f_\gamma(S, \mathscr{P}_w)$ of any non-pivot sequence S does not affect w-equivalency and can thus be arbitrary and, in particular, larger than $f_\gamma(S, \mathscr{D})$. As we will see, this gives more flexibility for constructing partitions while still maintaining correctness. Also note that a partition can be w-equivalent to $\mathscr{D}$ for more than one item w. The perhaps simplest, nontrivial partitioning that is w-equivalent to $\mathscr{D}$ is given by $\mathscr{P}_w = \{\, T \in \mathscr{D} : w \in T \,\}$, which corresponds to the partitioning used in the beginning of this section. It is easy to see there is an infinite number of sequence databases $\mathscr{P}_w$ such that $\mathscr{P}_w$ is (w, γ, λ)-equivalent to $\mathscr{D}$. All these databases agree on the multiset $G_{w,\gamma,\lambda}(\mathscr{P}_w)$ of generated pivot sequences.

3.4. Correctness of MG-FSM

The following theorem establishes the correctness of MG-FSM.

THEOREM 3.3. *MG-FSM outputs each frequent sequence $S \in F_{\sigma,\gamma,\lambda}(\mathscr{D})$ exactly once and with frequency $f_\gamma(S, \mathscr{D})$. No other sequences are output.*

PROOF. We first show that, if MG-FSM outputs a sequence S, it does so exactly once. If $|S| = 1$, S is output in the preprocessing phase but not in the mining phase (due to line 7 of Algorithm 1). If $|S| > 1$, S is output at partition $\mathscr{P}_{p(S)}$ in the mining phase (passes line 7) but at no other partitions (does not pass line 7).

Now fix some $S \in F_{\sigma,\gamma,\lambda}(\mathscr{D})$. We show that MG-FSM outputs S with correct frequency. If $|S| = 1$, then S occurs in the f-list and is output with correct frequency in the preprocessing phase. Assume $|S| > 1$ and set $w = p(S)$. We claim that S is output with correct frequency at partition $\mathscr{P}_w$ during the reduce phase of Algorithm 1. First, observe that $\mathscr{P}_w$ is w-equivalent to $\mathscr{D}$ since

$$G_{w,\gamma,\lambda}(\mathscr{P}_w) = \biguplus_{T\in\mathscr{D}} G_{w,\gamma,\lambda}(\mathscr{P}_w(T)) = \biguplus_{T\in\mathscr{D}, w\in T} G_{w,\gamma,\lambda}(T)$$
$$= \biguplus_{T\in\mathscr{D}} G_{w,\gamma,\lambda}(T) = G_{w,\gamma,\lambda}(\mathscr{D}).$$

Here the first equality follows from Eq. (3) and line 4 of Algorithm 1, the second equality from the definition of w-equivalency and line 3, and the third equality from the fact that $G_{w,\gamma,\lambda}(T) = \emptyset$ if $w \notin T$. From Lemma 3.2, we immediately obtain $f_\gamma(S, \mathscr{P}_w) = f_\gamma(S, \mathscr{D})$. Since therefore $S \in F_{\sigma,\gamma,\lambda}(\mathscr{P}_w)$, S is found by the FSM algorithm run in line 8 of Algorithm 1 and the assertion follows.

Now fix some $S \notin F_{\sigma,\gamma,\lambda}(\mathscr{D})$. We show that MG-FSM does not output S. If $|S| = 1$, then $S = w$ and $f_\gamma(w) < \sigma$ so that S is neither output in the preprocessing phase (not σ-frequent) nor in the mining phase (filtered out in line 7). If $|S| > \lambda$, we also do not output S since it is too long to be produced by $\mathrm{FSM}_{\sigma,\gamma,\lambda}$ in line 8 of Algorithm 1. Finally, if $2 \le |S| \le \lambda$, then S could potentially be output at partition $\mathscr{P}_w$, where $w = p(S)$. However, by the preceding arguments, we have $f_\gamma(S, \mathscr{P}_w) = f_\gamma(S, \mathscr{D}) < \sigma$, so that $S \notin \mathrm{FSM}_{\sigma,\gamma,\lambda}(\mathscr{P}_w)$. □

4. PARTITION CONSTRUCTION

Recall that MG-FSM rewrites each input sequence T into a small sequence database $\mathscr{P}_w(T)$ for each $w \in T$. We have shown that MG-FSM produces correct results if $\mathscr{P}_w(T)$ is w-equivalent to T, that is, if $G_{w,\gamma,\lambda}(T) = G_{w,\gamma,\lambda}(\mathscr{P}_w(T))$. In this section, we propose rewriting methods that aim to minimize the overall size of $\mathscr{P}_w(T)$. In fact, the smaller the $\mathscr{P}_w(T)$, the less data needs to be communicated between the map and reduce phases of MG-FSM, and the less work need be performed by the FSM algorithm in the mining phase.

To see the need for rewriting, assume that we simply set $\mathscr{P}_w(T) = \{T\}$ as before. Such an approach is impractical for a number of reasons.

(1) Input sequence T is replicated to d partitions, where d corresponds to the number of distinct items in T; this is wasteful in terms of communication cost.
(2) Every frequent sequence $S \in F_{\sigma,\gamma,\lambda}(\mathscr{D})$ will be computed multiple times: If S contains d distinct items, it is first computed by the FSM algorithm at each of the d corresponding partitions but then output at partition $\mathscr{P}_{p(S)}$ only; this is wasteful in terms of computational cost. S may also be computed (but not output) at partitions corresponding to distinct items that do not occur in S.

(3) The choice of $\mathscr{P}_w(T) = \{ T \}$ leads to highly imbalanced partition sizes: Partitions corresponding to frequent items are large (since these items occur in many input sequences), whereas partitions corresponding to less frequent items will be smaller (since these items occur in less input sequences).

Our rewrites address each of these points. (1) To reduce communication cost, we do not replicate T completely to d partitions. Instead, we send only the "relevant part" of T to that subset of the d partitions in which T generates pivot sequences. (2) Our rewrites ensure that partition $\mathscr{P}_w$ does not contain any items larger than w and only those items less than w that contribute to pivot sequences. If S contains d distinct items, it is mined and output only at that partition corresponding to the least frequent item w of these items. As before, S may be mined (but not output) at partitions corresponding to items that do not occur in S, but now only at those partitions corresponding to items that are less frequent than w. This greatly reduces computational cost. (3) Finally, our rewrites ensure that partitions corresponding to frequent items will contain many short sequences with few distinct items, whereas those corresponding to infrequent items contain few long sequences with many distinct items. Although we do not make any formal claims, our experiments in Section 7.2 suggest that this approach drastically reduces variance in partition sizes.

To get some insight into potential rewrites, consider input sequence $T = cbdbc$. Each of the following sequence databases is $(c, 0, 2)$-equivalent to T: $\mathscr{P}_1 = \{cbdbc\}$, $\mathscr{P}_2 = \{cbbc\}$, $\mathscr{P}_3 = \{cbc\}$, and $\mathscr{P}_4 = \{cb, bc\}$. Note that, in $\mathscr{P}_4$, the frequencies of both b and c increased by one since they occur in two sequences; our notion of w-equivalency allows for such cases.[5] It is not obvious which of these databases is best overall. On the one hand, $\mathscr{P}_3$ appears preferable to $\mathscr{P}_1$ and $\mathscr{P}_2$ since it contains less items. On the other hand, the maximum sequence length $\mathscr{P}_4$ is smaller than the one of $\mathscr{P}_3$ (i.e., 2 versus 3).

In what follows, we propose a number of properties that are useful in partition construction: minimality, irreducibility, and inseparability. Since it is computationally expensive to satisfy all of these properties, we give efficient rewriting algorithms that satisfy weaker, practical versions.

4.1. Minimality

In this and subsequent sections, we assume that we are given an input sequence T and aim to produce a w-equivalent sequence database $\mathscr{P}_w(T)$. Unless otherwise stated, our running assumption is that $\mathscr{P}_w(T)$ is w-equivalent to T.

Minimality, as defined shortly, ensures that $\mathscr{P}_w(T)$ contains no *irrelevant sequences*, that is, sequences that do not generate a pivot sequence.

Definition 4.1 (*Minimality*). A sequence database $\mathscr{P}_w(T)$ is (w, γ, λ)-*minimal* if

$$G_{w,\gamma,\lambda}(S) \neq \emptyset \quad \text{for all } S \in \mathscr{P}_w(T).$$

Clearly, any sequence $S \in \mathscr{P}_w(T)$ for which $G_{w,\gamma,\lambda}(S) = \emptyset$ does not contribute to w-equivalency; we can thus safely prune such irrelevant sequences. Minimality also allows to prune entire input sequences, even if they contain the pivot. Consider, for example, input sequence $T = addcd$ with pivot c. For $\gamma = 1$ (and any $\lambda \geq 2$), T does not generate any pivot sequences so that we can set $\mathscr{P}_c(T) = \emptyset$. Note that, for this reason, the choice of $\mathscr{P}_w(T) = \{ T \}$ does not guarantee minimality. For any $\gamma > 1$, T does generate pivot sequence ac so that $\mathscr{P}_c(T)$ becomes nonempty.

[5]Since b and c are not pivot sequences, their frequencies do not affect w-equivalency: We have $G_{c,0,2}(\{T\}) = \{cb, bc\} = G_{c,0,2}(\mathscr{P}_4)$ and thus $\{T\}$ and $\mathscr{P}_4$ are w-equivalent. MG-FSM uses this property to split long sequences into shorter ones.

In general, we prune sequences that either do not contain the pivot or in which each occurrence of a pivot is surrounded by sufficiently many irrelevant items, that is, items larger than the pivot. In particular, $G_{w,\lambda,\gamma}(T) \neq \emptyset$ if and only if there is at least one occurrence of some item $w' \leq w$ within distance $\gamma + 1$ of an occurrence of pivot w. Length parameter λ does not influence minimality. If a sequence is irrelevant for some choice of γ, it is also irrelevant for all $\gamma' < \gamma$; the opposite does not hold in general. Thus minimality pruning is most effective when γ is small.

4.2. Irreducibility

Irreducibility is one of the main concepts employed by MG-FSM. We say that a sequence is irreducible if there is no shorter way to write it.

Definition 4.2 (*Irreducibility*). A sequence S is (w, γ, λ)-*irreducible* if there exists no sequence S' with length $|S'| < |S|$ such that

$$G_{w,\gamma,\lambda}(S) = G_{w,\gamma,\lambda}(S').$$

If such a sequence S' exists, we say that S *reduces* to S'. Moreover, we say $\mathscr{P}_w(T)$ is irreducible if all sequences $S \in \mathscr{P}_w(T)$ are irreducible. Consider, for example, the sequences $S = acdeb$ and $S' = acb$ and pivot c. Here S' is obtained from S by removing all *irrelevant* items, that is, all items larger than the pivot. Then S' is a $(c, 2, 2)$-reduction of S; it is not, however, a $(c, 1, 2)$-reduction of S. This is because $cb \in G_{c,1,2}(S')$ but $cb \notin G_{c,1,2}(S)$. Thus, perhaps contrary to expectation, we cannot simply remove all irrelevant items to obtain a reduced sequence: Whether an irrelevant item can be dropped depends on the particular choice of γ and λ. Note that the shortest $(c, 1, 2)$-reduction of S is given by ac.

We can reduce sequences in more sophisticated ways than by simply removing irrelevant items. For example, $S = cbac$ can be $(c, 0, 2)$-reduced to acb, but cannot be $(c, 0, 2)$-reduced to any sequence $S' \subset_\infty S$. Thus reduction can be non-monotonic, that is, may require reordering of items. As an additional example, consider the sequence $S = acadac$ which $(c, 0, 2)$-generates sequence ac twice. Since repeated generations do not affect w-equivalency, we can reduce S to aca. Both detection of non-monotonic reductions and (exhaustive) detection of repeatedly generated pivot sequences appear computationally challenging. Since we perform sequence reduction for every input sequence, we need it to be extremely efficient. We thus make use of a weaker form of irreducibility which does not consider such sophisticated rewrites.

To avoid confusion with repeated items, we explain our reduction techniques using indexes instead of items. Let $T = s_1 \cdots s_l$ and consider pivot w. We say that index i is w-*relevant* if s_i is w-relevant, that is, $s_i \leq w$; otherwise index i is w-irrelevant. In sequence $abddc$, for example, indexes 3 and 4 are c-irrelevant. Since irrelevant indexes do not contribute to a pivot sequence, it suffices to keep track of the fact that an index i is irrelevant, that is, we do not need to retain value s_i. We thus replace all items at irrelevant indexes by a special *blank symbol*, denoted "␣"; in what follows, we assume that $w < \text{␣}$ for all $w \in \Sigma$. Using blanks, sequence $abddc$ is written as $ab\text{␣}\text{␣}c$ (for pivot c). As discussed in Section 6, our use of a blank symbol is helpful in an implementation of MG-FSM since it enables effective compression of irrelevant items (e.g., $abddc$ can be written as $ab\text{␣}^2c$).

In what follows, we describe a number of reductions that reduce T by removing items or blanks (while still maintaining correctness). Our first reduction, termed *unreachability reduction*, removes unreachable items, that is, items that are "far away" from any pivot. Fix an input sequence $T = s_1 \cdots s_l$ and pick any index $1 \leq i \leq |T|$. To determine whether index i is unreachable, we consider the "distance" of i to its surrounding pivots. Suppose there is a pivot at an index $i' < i$, that is, $s_{i'} = w$, and denote by i_{prev}

the largest such index. Informally, the *left distance* $l_{w,\gamma,\lambda}(i \mid T)$ of index i is given by the smallest number of items that we need to "step onto" when moving from i_{prev} to i via: (1) relevant items and (2) by skipping at most γ items in each step. If no such path exists or if its length is larger than λ, we set $l_{w,\gamma,\lambda}(i \mid T) = \infty$. Similarly, we define the *right distance* $r_{w,\gamma,\lambda}(i \mid T)$ of index i as the distance to the closest pivot to the right of index i. A formal definition of left and right distances is given in Appendix A. For example, we obtain the following left and right distances for pivot c, $\gamma = 1$, and $\lambda = 4$.

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
s_i	***c***	***a***	␣	***b***	***a***	***b***	␣	***a***	␣	***c***	␣	␣	***a***	␣
$l_{c,1,4}$	1	2	(2)	3	4	4				1	(2)	(2)		
$r_{c,1,4}$	1			4	4	3	(3)	2	(2)	1				

Here blank entries indicate infinite distance and entries in parentheses indicate irrelevant items (which cannot be "stepped onto"). For example, the closest pivot to the left of index 5 occurs at index 1. We can reach index 5 from index 1 via the items at the four indexes $1 \cdot 2 \cdot 4 \cdot 5$ (but not via $1 \cdot 3 \cdot 5$ because index 3 is irrelevant); thus the left distance of index 5 is four.

Definition 4.3 (*Reachability*). Let $T = s_1 \cdots s_l$. Index i is (w, γ, λ)-*reachable* if

$$\min\left\{ l_{w,\gamma,\lambda}(i \mid T), r_{w,\gamma,\lambda}(i \mid T) \right\} \leq \lambda.$$

Continuing the prior example, indexes 1–12 are $(c, 1, 4)$-reachable, indexes 1–4 and 6–12 are $(c, 1, 3)$-reachable, and indexes 1–3 and 8–12 are $(c, 1, 2)$-reachable.

LEMMA 4.4 (UNREACHABILITY REDUCTION). *Let $T = s_1 \cdots s_l$ and denote by I the set of all (w, γ, λ)-unreachable indexes in T. Then*

$$G_{w,\gamma,\lambda}(T) = G_{w,\gamma,\lambda}(T_{-I}),$$

where T_{-I} is obtained by removing the items at indexes I from T.

A proof of this lemma is given in Appendix A. The lemma asserts that, in fact, we can simply remove all unreachable indexes at once. In our example, we obtain $ca␣a␣c␣␣$ for $\lambda = 2$ (removing indexes 4–7, 13, and 14) and $ca␣bb␣a␣c␣␣$ for $\lambda = 3$ (removing indexes 5, 13, and 14).

The computational cost of our unreachability reduction mainly depends on the time required to compute left and right distances. This can be done efficiently as follows: We compute the left distances in a forward scan of T, and right distances in a subsequent backward scan. During each scan, we keep track of the position i and distance d of the most recently processed relevant item; we also keep track of the index i' of the most recently processed relevant item at distance $d - 1$. This information is sufficient to compute the distance of the currently processed item. With this approach, computation of distances takes time linear in $|T|$.

Reconsider the sequence $T = ca␣a␣c␣␣$ from before. Clearly, the two blanks at the end do not carry useful information so that we would like to reduce T to $ca␣a␣c$. The following lemma asserts that we can do so, that is, we can drop prefixes and suffixes of irrelevant items.

LEMMA 4.5 (PREFIX/SUFFIX REDUCTION).

$$G_{w,\gamma,\lambda}(␣^{l_1} T ␣^{l_2}) = G_{w,\gamma,\lambda}(T).$$

See Appendix B for a proof of this lemma. Prefix/suffix reduction is particularly effective in conjunction with unreachability reduction. In fact, irrelevant items that are not part of a prefix or suffix in T can become so after removing unreachable items.

In our ongoing example, this is the case for the irrelevant items at indexes 11 and 12, which became part of the suffix only after the removal of indexes 13 and 14. In fact, if T contains exactly one pivot, and T' is obtained from T by an unreachability reduction followed by a prefix/suffix reduction, one can show that $|T'| \leq 2(\gamma + 1)\lambda$, a quantity that is independent of $|T|$.

The following lemma asserts that we can shrink long sequences of blanks.

LEMMA 4.6 (BLANK REDUCTION). *For $k > \gamma + 1$,*

$$G_{w,\gamma,\lambda}(T_1 \sqcup^k T_2) = G_{w,\gamma,\lambda}(T_1 \sqcup^{\gamma+1} T_2).$$

The proof is similar to that of Lemma 4.5 and omitted. Thus every sequence of $k > \gamma+1$ blanks can be replaced by exactly $\gamma+1$ blanks. Note that blank reduction can be effective only when T contains multiple occurrences of the pivot (since otherwise T does not contain more than γ blanks after our unreachability and prefix/suffix reductions).

The preceding reductions are not complete, that is, they do not necessarily produce irreducible sequences. For example, sequence $ca\sqcup a\sqcup c$ is $(c, 1, 2)$-reducible to $ca\sqcup ac$ (and, in fact, to aca); this reduction is not covered by our techniques. In our experiments, however, we found that the simple reduction techniques described earlier already lead to a significant reduction of partition sizes.

4.3. Aggregation

Reconsider the example database $\mathscr{D}_{\text{ex}}$ given in Eq. (1). If we apply all the reduction techniques given before, we obtain

$$\mathscr{P}_c'' = \{\, acb, acb, acb \sqcup\sqcup bca, bca \,\}$$

for pivot c, $\sigma = 1$, $\gamma = 1$, and $\lambda = 3$. Observe that sequence acb is repeated in $\mathscr{P}_c''$, even though $\mathscr{D}_{\text{ex}}$ does not contain any repeated sequences. To reduce communication cost, we can aggregate such repeated sequences and represent them using (sequence, frequency)-pairs. Thus we obtain

$$\mathscr{P}_c''' = \{(acb, 2), (acb \sqcup\sqcup bca, 1), (bca, 1)\}.$$

Compression of repeated sequences can be performed efficiently by exploiting the *combine* functionality of MapReduce. When the FSM algorithm run in the mining phase is able to exploit frequency information, computational cost and memory consumption may also reduce; see Section 6 for details.

4.4. Inseparability

Recall the set $\mathscr{P}_c''$ provided earlier. We show next that sequence $acb \sqcup\sqcup bca \in \mathscr{P}_c''$ can be "split" into two sequences acb and bca without sacrificing $(c, 1, 3)$-equivalency to $\mathscr{D}_{\text{ex}}$. Such sequence splitting, which we refer to as *separation*, is meant to increase the effectiveness of aggregation. In fact, if we perform the preceding split and aggregate, we obtain partition $\{(acb, 3), (bca, 2)\}$ which is compact and constitutes an aggregated version of the partition of Eq. (4) that we promised to obtain in Section 3.3.

Definition 4.7 (*Separability*). An input sequence T is *weakly* (w, γ, λ)*-separable* if there exist sequences T_1 and T_2 such that $G_{w,\gamma,\lambda}(T) = G_{w,\gamma,\lambda}(\{\, T_1, T_2 \,\})$, $G_{w,\gamma,\lambda}(T_1) \neq \emptyset$, and $G_{w,\gamma,\lambda}(T_2) \neq \emptyset$; otherwise it is weakly (w, γ, λ)*-inseparable*. T is strongly (w, γ, λ)-*separable* (or simply (w, γ, λ)-separable) if additionally $|T_1| + |T_2| \leq |T|$.

Note that separation is possible only because we allow for an increase of frequencies on nonpivot sequences in $\mathscr{P}_w(T)$. If a sequence is w-separable, we can safely write it in terms of multiple shorter sequences which we refer to as *splits*. As indicated previously,

both strong and weak separation improve the effectiveness of aggregation, and strong separation additionally reduces the overall partition size.

Revisiting $S = acb␣␣bca$, we observe that S is $(c, 1, 3)$-separable into splits $\{acb, bca\}$. In general, one can test for weak separability as follows: Construct the set $G_{w,\gamma,\lambda}(S)$ and create a graph (V, E), where $V = G_{w,\gamma,\lambda}(S)$ and edge $(S_1, S_2) \in E$ if there exists an item $w' \in \Sigma$ and a sequence S' of form ww' or $w'w$ such that $S' \subseteq_0 S_1$ and $S' \subseteq_0 S_2$. If the resulting graph is connected, S is not (even weakly) (w, γ, λ)-separable. Intuitively, this is because any input sequence that generates S_i, $i \in \{1, 2\}$, will also generate S'. Since S' is a pivot sequence, however, we must not generate it in more than one split, which implies that S_1 and S_2 must be generated by the same split. In our example, we have $G_{c,1,3}(S) = \{ac, acb, cb, bc, bca, ca\}$; the corresponding graph has two connected components so that S is $(c, 1, 3)$-separable. As a final remark, one can show that any sequence S can be separated into k splits, where k is the number of connected components in the graph corresponding to the pivot sequences generated by S.

As with reduction, there are quite sophisticated cases of separable sequences. As a pathological example, it is possible that an irreducible sequence is weakly separable only into irreducible sequences of larger length. Consider, for example, sequence $T = abc$, pivot c, $\gamma = 1$, and $\lambda = 3$. Here T is irreducible and can be separated into splits $\{ac, a␣bc\}$ in which each sequence is again irreducible. Separations such as this one appear counterproductive. Moreover, the weak separation detection method outlined before is too expensive in practice since it generates $G_{w,\gamma,\lambda}(S)$. In what follows, we present a simple separation technique, called *blank separation*, that is efficient and handles cases such as sequence $acb␣␣bca$ discussed earlier.

Assume S is of form $S_1␣^{k_1}S_2␣^{k_2}\cdots␣^{k_{n-1}}S_n$, where $k_i > \gamma$ for $1 \leq i < n$, that is, consists of subsequences separated by sufficiently many blanks. Our blank separation technique first breaks up S into the set $B(S) = \{S_1, \ldots, S_n\}$; irrelevant subsequences (i.e., subsequences that do not generate a pivot sequence) are not included in $B(S)$. For example, $B(acb␣␣bca) = \{acb, bca\}$ for $\gamma = 1$. Since the S_i are separated by at least $\gamma + 1$ gaps in S, every pivot sequence in S is generated by one or more of the S_i (i.e., it does not span multiple S_i). Thus $D(G_{w,\gamma,\lambda}(B(S))) = G_{w,\gamma,\lambda}(S)$, where $D(A)$ denotes the set of distinct sequences in multiset A. We now consider the S_i as candidate splits and proceed as previously: We first construct a graph $G = (V, E)$, where $V = D(B(S))$ and $(S_i, S_j) \in E$, $i \neq j$, if S_i and S_j generate a joint pivot sequence.[6] Denote by $B_1, \ldots, B_k$ the connected components of G. We output a single sequence S'_i for component B_i by *stitching* the subsequences in B_i with sufficiently many blanks as

$$S'_i = B_{i,1}␣^{\gamma+1}B_{i,2}␣^{\gamma+1}\cdots␣^{\gamma+1}B_{i,|B_i|},$$

where $B_{i,j}$ refers to the j-th sequence in B_i. In our ongoing example, we have $B_1 = \{acb\}$, $B_2 = \{bca\}$ and thus $S'_1 = acb$ and $S'_2 = bca$. As another example, consider the sequence $bcb␣bca␣ac␣bca␣c$ for $\gamma = 0$ and $\lambda = 3$. Then $B_1 = \{bcb, bca\}$ and $B_2 = \{ac\}$, and thus $S'_1 = \{bcb␣bca\}$ and $S'_2 = \{ac\}$. Thus blank separation is able to detect (to some extent) repeated subsequences as well as irrelevant subsequences.

In combination with our reduction techniques, blank separation is only effective if S contains more than one occurrence of a pivot. This is because our reduction techniques ensure that otherwise S does not contain more than γ consecutive blanks. On datasets in which items are rarely repeated, blank separation is therefore not effective. For example, we found that in our experiments on text data (where each sequence corresponded to a single sentence), the overall effect of blank separation was marginal.

[6]This can be tested efficiently as follows. We compute once for each split S_i the set L_i (R_i) of the $\gamma + 1$ items to the left (right) of the pivot in S_i (if there is more than one occurrence of a pivot, repeat for each occurrence and union). Two splits S_i and S_j generate a joint pivot sequence if and only if $L_i \cap L_j \neq \emptyset$ or $R_i \cap R_j \neq \emptyset$.

4.5. Summary

Algorithm 2 integrates all of our rewrites into a single algorithm; this approach is more efficient than performing the rewrites one after the other.

We start with a backward scan of the input sequence to obtain all right distances (lines 2–4); distance computation is performed incrementally as described in Section 4.2. We then perform a forward scan of T to compute the set $B(T)$ of candidate splits for blank separation (lines 6–37). During the scan, we maintain a current candidate split S which we incrementally grow as we see more items. In particular, for each encountered item, we first compute its left distance (line 11). If the item is not reachable, we ignore it (unreachability reduction; lines 13–15 and 22–24). Otherwise, the item is reachable. If it is also relevant, we append it to the candidate split (line 20). If it is irrelevant, we proceed as follows. If the item occurs at the beginning of the candidate split, we ignore it (prefix reduction; lines 22–24). Otherwise, we remember that we have seen an irrelevant item, that is, a blank (line 25). As soon as we see $\gamma + 1$ consecutive blanks (blank separation; lines 26–32), we store the current candidate split in $B(T)$ if it is relevant (thereby ensuring minimality; lines 27–29, also lines 35–37) and start a new candidate split (lines 30–31). Only if we see a relevant item after a series of $\leq\gamma$ blanks do we actually append blanks to candidate split S (suffix reduction; lines 16–19).

Once the candidate splits have been computed, we proceed as in Section 4.4 to compute $\mathscr{P}_w(T)$ by stitching together overlapping candidate splits (lines 40–45). Note that if blank reduction is not used, we simply stitch together all candidate splits in $B(T)$ to obtain a single rewritten output sequence.

5. MAXIMALITY AND CLOSEDNESS

Frequent sequence mining from large datasets can potentially generate a large number of sequences, especially when the support threshold is low and the length parameter large. A standard approach [Yan et al. 2003; Fournier-Viger et al. 2013] to reduce the number of mined sequences without losing information is to output only sequences that are maximal or closed. Such sequences compactly represent the set of all frequent sequences along with their exact frequency (closed sequences) or a lower bound thereof (maximal sequences). In this section, we show how MG-FSM can be adapted to mine maximal or closed sequences in a scalable fashion.

5.1. Definitions

The key motivation behind maximal and closed sequence mining is that knowing that a sequence S' is frequent also provides information about whether certain subsequences of S' are frequent. In more detail, set

$$\gamma^- = \begin{cases} 0 & \text{if } \gamma < \infty \\ \infty & \text{if } \gamma = \infty. \end{cases}$$

The following lemma describes the relationship between the frequency of a sequence S' and the frequency of (some of) its subsequences.

LEMMA 5.1 (SUPPORT MONOTONICITY). *Let S and S' be two sequences such that $S \subseteq_{\gamma^-} S'$. For all sequence databases $\mathscr{D}$, we have*

$$\mathrm{Sup}_\gamma(S, \mathscr{D}) \supseteq \mathrm{Sup}_\gamma(S', \mathscr{D}).$$

It directly follows that $f_\gamma(S, \mathscr{D}) \geq f_\gamma(S', \mathscr{D})$. Thus, if S' is frequent and S is a γ^--subsequence of S', then S must also be frequent. In particular, if $\gamma = \infty$, every subsequence of S' is frequent. If $\gamma < \infty$, the consecutive subsequences of S' are frequent. The lemma is proven in Appendix C.

ALGORITHM 2: Partition construction

Require: Input sequence $T = s_1 \dots s_k$, w, γ, λ, f-list $F_{\sigma,0,1}(\mathscr{D})$
Ensure: $\mathscr{P}_w(T)$

// *backward scan*
for $i \leftarrow k$ **downto** 1 **do**
  Compute right distance $r_{w,\gamma,\lambda}(i \mid T)$
end for

// *forward scan, compute candidate splits (blank separation)*
$B(T) \leftarrow \emptyset$ // *set of candidate splits*
$S \leftarrow \emptyset$ // *buffer for next candidate split*
blanks $\leftarrow 0$ // *number of consecutive blanks seen*
for $i \leftarrow 1$ **to** k **do**
  Compute left distance $l_{w,\gamma,\lambda}(i \mid T)$
  if $s_i \leq w$ **then** // *relevant item*
    if index i is not (w, γ, λ)-reachable **then** // *unreachability reduction*
      continue
    end if
    if *blanks* > 0 **then** // *suffix reduction*
      Append $\sqcup^{blanks}$ to S
      blanks $\leftarrow 0$
    end if
    Append s_i to S
  else // *irrelevant item*
    if $S = \emptyset$ **or** i is not (w, γ, λ)-reachable **then** // *prefix/unreachability reduction*
      continue
    end if
    blanks $\leftarrow$ *blanks* $+ 1$
    if *blanks* $\geq \gamma + 1$ **then** // *blank reduction, blank separation*
      if $|S| > 1$ **then** // *minimality test*
        $B(T) \leftarrow B(T) \biguplus \{ S \}$ // *candidate split found*
      end if
      $S \leftarrow \emptyset$ // *start new candidate split*
      blanks $\leftarrow 0$
    end if
  end if
end for
if $|S| > 1$ **then** // *minimality test*
  $B(T) \leftarrow B(T) \biguplus \{ S \}$ // *candidate split found*
end if

// *compute splits: test for separability and stitch candidate splits*
$\mathscr{P}_w(T) \leftarrow \emptyset$
Construct graph $G = (V, E)$ with $V = D(B(T))$ and $(S_i, S_j) \in E$ for $i \neq j$ iff S_i and S_j generate a joint pivot sequence
for each connected component $\{ S_1, \dots, S_l \}$ of G **do**
  $S' \leftarrow S_1 \sqcup^{\gamma+1} S_2 \sqcup^{\gamma+1} \dots \sqcup^{\gamma+1} S_l$
  $\mathscr{P}_w(T) \leftarrow \mathscr{P}_w(T) \cup \{ S' \}$
end for

return $\mathscr{P}_w(T)$

Note that we carefully distinguished the cases $\gamma = \infty$ and $\gamma < \infty$. The reason is that, perhaps contrary to intuition, not every subsequence of a frequent sequence is necessarily frequent as well. To see this, consider the database

$$\mathscr{D}_{\text{ex}} = \{\, abc, abec, abcd, abcfd \,\}, \tag{5}$$

and its set of $(2, 1, 4)$-frequent sequences:

$$F_{2,1,4}(\mathscr{D}_{\text{ex}}) = \{a(4), b(4), c(4), d(2), ab(4), ac(4), bc(4), \\ cd(2), abc(4), acd(2), bcd(2), abcd(2)\}.$$

Here we also provide frequencies (which are not formally part of $F_{2,1,4}(\mathscr{D}_{\text{ex}})$). Observe that sequence $S = abcd$ is frequent whereas some of its subsequences are not. In particular, sequence $bd \subseteq_1 S$ is not frequent, even though bd is a γ-subsequence (but not a γ^--subsequence) of S. As asserted by Lemma 5.1, all consecutive subsequences of S are indeed frequent (a, ab, abc, ...).

Consider a frequent sequence $S' \in F_{\sigma,\gamma,\lambda}(\mathscr{D})$ of length l. The prior lemma implies that, when $\gamma = \infty$, each of the $2^l - 1$ nonempty subsequences of S' are also frequent. Similarly, when $\gamma < \infty$, each of the $l(l+1)/2$ nonempty consecutive subsequences of S are frequent. The goal of mining maximal sequences is to avoid outputting these "redundant" sequences. In particular, a sequence is maximal if and only if it is not redundant.

Definition 5.2 (*Maximality*). A sequence S is $(\sigma, \gamma, \lambda)$-*maximal* if S is $(\sigma, \gamma, \lambda)$-frequent and there is no sequence $S' \supset_{\gamma^-} S$ which is also $(\sigma, \gamma, \lambda)$-frequent. The set of $(\sigma, \gamma, \lambda)$-maximal sequences is given by

$$F^{\max}_{\sigma,\gamma,\lambda}(\mathscr{D}) = \left\{\, S \in F_{\sigma,\gamma,\lambda}(\mathscr{D}) \mid \neg\exists S' \in F_{\sigma,\gamma,\lambda}(\mathscr{D}) : S \subset_{\gamma^-} S' \,\right\}.$$

For our running example, we obtain

$$F^{\max}_{2,\infty,4}(\mathscr{D}_{\text{ex}}) = \{abcd(2)\}, \\ F^{\max}_{2,1,4}(\mathscr{D}_{\text{ex}}) = \{acd(2), abcd(2)\}. \tag{6}$$

As alluded to previously, we can reconstruct the set of all frequent sequences from the set of maximal sequences:

$$F_{\sigma,\gamma,\lambda}(\mathscr{D}) = \left\{S \mid S \subseteq_{\gamma^-} S', S' \in F^{\max}_{\sigma,\gamma,\lambda}(\mathscr{D})\right\}.$$

This is true because: (1) every γ^--subsequence of a maximal sequence must be $(\sigma, \gamma, \lambda)$-frequent and (2) every $(\sigma, \gamma, \lambda)$-frequent sequence must be a γ^--subsequence of some maximal sequence. Here (1) holds by Lemma 5.1, and (2) holds by definition of $F^{\max}_{\sigma,\gamma,\lambda}(\mathscr{D})$.

A similar reasoning can be applied to closed sequences. Here we want to be able to reconstruct from the set of closed sequences the set of frequent sequences along with their frequencies. The following notion of closedness allows for such reconstruction.

Definition 5.3 (*Closedness*). A sequence S is $(\sigma, \gamma, \lambda)$-*closed* if S is $(\sigma, \gamma, \lambda)$-frequent and there is no $(\sigma, \gamma, \lambda)$-frequent sequence $S' \supset_{\gamma^-} S$ of the same frequency, that is, with $f_\gamma(S', \mathscr{D}) = f_\gamma(S, \mathscr{D})$. The set of $(\sigma, \gamma, \lambda)$-closed sequences is given by

$$F^{\text{closed}}_{\sigma,\gamma,\lambda}(\mathscr{D}) = \left\{\, S \in F_{\sigma,\gamma,\lambda}(\mathscr{D}) \mid \neg\exists S' \in F_{\sigma,\gamma,\lambda}(\mathscr{D}) : S \subset_{\gamma^-} S' \wedge f_\gamma(S, \mathscr{D}) = f_\gamma(S', \mathscr{D}) \,\right\}.$$

For our running example, we obtain

$$F^{\text{closed}}_{2,\infty,4}(\mathscr{D}_{\text{ex}}) = \{abcd(2), abc(4)\}, \\ F^{\text{closed}}_{2,1,4}(\mathscr{D}_{\text{ex}}) = \{ac(4), abc(4), acd(2), abcd(2)\}. \tag{7}$$

Ignoring frequencies, observe that $F^{\max}_{\sigma,\gamma,\lambda}(\mathscr{D}) \subseteq F^{\text{closed}}_{\sigma,\gamma,\lambda}(\mathscr{D}) \subseteq F_{\sigma,\gamma,\lambda}(\mathscr{D})$ so that reconstruction of frequent sequences is still possible. The frequency of a reconstructed sequence is the maximum of the frequencies of its closed consecutive supersequences. For example, $f_1(bc) = \max\{f_1(abc),\ f_1(abcd)\} = \max\{4, 2\} = 4$.

5.2. Mining Maximal Sequences

One way to adapt MG-FSM to mine maximal sequences is to first compute the set of all frequent sequences $F_{\sigma,\gamma,\lambda}(\mathscr{D})$ and subsequently filter out sequences that are not maximal. This approach is not efficient, however: First, we mine too many sequences from each partition (i.e., sequences that cannot possibly be maximal). Second, a naïve approach for the subsequent filtering step takes $O(|F_{\sigma,\gamma,\lambda}(\mathscr{D})|^2)$ time. In what follows, we propose a more suitable approach which integrates the maximality constraint directly into MG-FSM. We refer to this adaptation as MG-FSM$^+$.

Recall that MG-FSM creates one partition $\mathscr{P}_w$ for each item w. Let S be a pivot sequence for $\mathscr{P}_w$; that is, $p(S) = w$ and $2 \leq |S| \leq \lambda$. Then MG-FSM guarantees $f_\gamma(S, \mathscr{P}_w) = f_\gamma(S, \mathscr{D})$. Now suppose S is frequent so that MG-FSM outputs it when mining $\mathscr{P}_w$. Ideally, we would like MG-FSM$^+$ to output S if and only if it is also maximal. Unfortunately, we cannot test for maximality locally in each partition because the frequencies $f_\gamma(S', \mathscr{P}_w)$ of supersequences $S' \supset_{\gamma^-} S$ may not (and usually will not) coincide with the corpus frequency $f_\gamma(S', \mathscr{D})$ when $p(S') \neq w$. To see this, fix σ, γ, and λ and denote the output of MG-FSM at partition $\mathscr{P}_w$ by

$$F_w(\mathscr{P}_w) = \left\{ S \in F_{\sigma,\gamma,\lambda}(\mathscr{D}) \mid p(S) = w \wedge S \neq w \right\}.$$

For our example database $\mathscr{D}_{\text{ex}}$ of Eq. (5) and $\sigma = 2$, $\gamma = 1$ and $\lambda = 4$, we have

$$F_c(\mathscr{P}_c) = \left\{ ac(4), bc(4), abc(4) \right\}.$$

Now consider frequent sequence $S = abc \in F_c(\mathscr{P}_c)$. Sequence S is not maximal since sequence $S' = abcd \supset_0 S$ is frequent in $\mathscr{D}_{\text{ex}}$. However, $S' \notin F_c(\mathscr{P}_c)$ so that we cannot decide locally at $\mathscr{P}_c$ whether or not S is maximal.

A key ingredient to MG-FSM$^+$ is to test for *local maximality* and to output in each partition only those frequent sequences that are locally maximal. Our local maximality test exploits that, whenever a sequence is not locally maximal, then it is also not (globally) maximal; we thus do not incorrectly filter out maximal sequences. The set of locally maximal sequences at partition $\mathscr{P}_w$ is given by the following.

Definition 5.4 (*Local and Global Maximality*). A sequence S with $p(S) = w$ is *locally maximal* if $S \in F^{\max}_w(\mathscr{P}_w)$, where

$$F^{\max}_w(\mathscr{P}_w) = \left\{ S \in F_w(\mathscr{P}_w) \mid \neg\exists S' \in F_w(\mathscr{P}_w) : S \subset_{\gamma^-} S' \right\}.$$

Sequence S is *globally maximal* if $S \in F^{\max}_{\sigma,\gamma,\lambda}(\mathscr{D})$.

Thus a sequence S with pivot $w = p(S)$ is locally maximal if it is maximal with respect to the output $F_w(\mathscr{P}_w)$ at the partition $\mathscr{P}_w$ that mines S. Stated differently, a sequence S is locally maximal if and only if S is frequent, $|S| \geq 2$, and there is no frequent sequence $S' \supset_{\gamma^-} S$ with the same pivot item (i.e., $p(S') = p(S)$). For our running example, we obtain

$$F^{\max}_c(\mathscr{P}_c) = \{ac(4), abc(4)\}.$$

Observe that $bc \in F_c(\mathscr{P}_c)$ but, since $abc \in F_c(\mathscr{P}_c)$, $bc \notin F^{\max}_c(\mathscr{P}_c)$. Also observe that bc is indeed not maximal. Figure 1 shows the set of locally maximal sequences for each of the partitions obtained for our example database. The following lemma asserts that we can safely filter out sequences that are not locally maximal.

LEMMA 5.5. *Every globally maximal sequence S with $|S| \geq 2$ is also locally maximal.*

A proof of this lemma is given in Appendix D. Note that the opposite does not necessarily hold, that is, there can exist a sequence S that is locally but not globally maximal. This happens when: (1) there is no frequent sequence $S' \supset_{\gamma^-} S$ with $p(S') = w$ but (2) there is a frequent sequence $S'' \supset_{\gamma^-} S$ with $p(S'') > w$. Note that $p(S'') \geq w$ for all $S'' \supseteq S$. Here (1) implies that S is locally maximal and (2) implies that S is not globally maximal. We refer to such sequences as *spurious* sequences. In $F_c^{\max}(\mathscr{P}_c)$ shown before, all sequences are spurious (since both *acd* and *abcd* are frequent).

MG-FSM$^+$ (which is given as Algorithm 3) is divided into two steps, each corresponding to a MapReduce job.

(1) Mine and output the set $F_w^{\max}(\mathscr{P}_w)$ of locally maximal sequences for each partition $\mathscr{P}_w$; this step is similar to MG-FSM. A straightforward approach to obtain $F_w^{\max}(\mathscr{P}_w)$ for each partition $\mathscr{P}_w$ is to first compute $F_w(\mathscr{P}_w)$ and then test whether each so-obtained sequence is locally maximal. A more efficient alternative which we use in MG-FSM$^+$ is to directly mine locally maximal sequences instead. To do so, we can use any suitable maximal sequence miner, such as the algorithm of Fournier-Viger et al. [2013] or the method discussed in Section 6.4.[7]
(2) Determine the set of globally maximal sequences by identifying and eliminating all spurious sequences.

The algorithm is illustrated in Figure 1.

In the remainder of this section, we discuss an efficient technique for pruning spurious sequences. Let S^+ be a locally maximal sequence with $p(S^+) = w^+$; that is, $S^+ \in F_{w^+}^{\max}(\mathscr{P}_{w^+})$. Furthermore, let S be a γ^--subsequence of S^+ and set $w = p(S)$. The key idea of our approach is as follows: If S is locally maximal, then S^+ "proves" that S is spurious; we refer to such an S^+ as a *witness* for the spuriousness of S. In our running example, $S = ac$ is a spurious sequence at partition $\mathscr{P}_c$; sequence $S^+ = acd$ from partition $\mathscr{P}_d$ is its witness (see also Figure 1). Note that a spurious sequence can have more than one witness.

To ensure the efficiency of the pruning step, we need to ensure that we find a witness for each spurious sequence efficiently and in parallel. MG-FSM$^+$ uses the following observation to restrict search to the set of *primary witnesses*.

LEMMA 5.6. *Let S be a spurious sequence. Then there is a* primary-witness sequence *S^+ which satisfies:*

(1) $S \subset_{\gamma^-} S^+$, $p(S^+) > p(S)$, and S^+ is frequent,
(2) S^+ is locally maximal,
(3) there is no intermediate sequence S^* with $p(S^*) < p(S^+)$ and $S \subset_{\gamma^-} S^* \subset_{\gamma^-} S^+$.

Again, there can be more than one primary witness for S. The key property exploited by MG-FSM$^+$ is (3). We provide some intuition on the assertion of the lemma here, while a formal proof can be found in Appendix E. First, in our example of Figure 1, *abc* (from $\mathscr{P}_c$) is a primary witness for spurious sequence *ab* (from $\mathscr{P}_b$). In contrast, even though sequence *abcd* (from $\mathscr{P}_d$) is a witness for *ab*, it is not a primary witness since it violates (3) with intermediate sequence *abc*. In general, the lemma tells us that if S is spurious, then there is primary witness sequence S^+ which contains S as a γ^--subsequence arranged in a "certain way". To see how S^+ is arranged, let $w^+ = p(S^+)$.

[7]This approach is valid if we ensure that $p(T) \leq w$ for all $T \in \mathscr{P}_w$. Since during rewriting (MAP1, Section 4.2) we replace all irrelevant items (i.e., items $> w$) by blanks, this property holds.

ALGORITHM 3: The MG-FSM$^+$ algorithm

Require: Sequence database $\mathscr{D}$, σ, γ, λ, f-list $F_{\sigma,0,1}(\mathscr{D})$, type $\in$ {max, closed}
1: MAP1(T):
2: Same as MAP(T) in Algorithm 1; removal of irrelevant items required
3:
4: REDUCE1(w, $\mathscr{P}_w$):
5: $F^{\text{type}}_{\sigma,\gamma,\lambda}(\mathscr{P}_w) \leftarrow \text{FSM}^{\text{type}}_{\sigma,\gamma,\lambda}(\mathscr{P}_w)$
6: **for all** $S \in F^{\text{type}}_{\sigma,\gamma,\lambda}(\mathscr{P}_w)$ **do**
7: **if** $p(S) = w$ and $S \neq w$ **then**
8: Output $(S,\ f_\gamma(S, \mathscr{P}_w))$
9: **end if**
10: **end for**
11:
12: MAP2(S^+, $f_\gamma(S^+, \mathscr{D})$): // *where* $S^+ \in F_{\sigma,0,1}(\mathscr{D}) \cup \bigcup_w F^{type}_w(\mathscr{P}_w)$
13: $f^+ \leftarrow f_\gamma(S^+, \mathscr{D})$
14: $l^+ \leftarrow |S^+|$
15: **for all** $S \in W_\gamma(S^+)$ **do**
16: Output $(S, \langle l^+, f^+\rangle)$
17: **end for**
18:
19: REDUCE2(S, { $\langle l, f\rangle$ }):
20: **switch** (type)
21: **case** max**:**
22: $\langle l^*, f^*\rangle \leftarrow$ pair in { $\langle l, f\rangle$ } having maximum length l
23: **case** closed**:**
24: $\langle l^*, f^*\rangle \leftarrow$ pair in { $\langle l, f\rangle$ } having highest frequency f;
resolve ties by picking the pair with maximum length l
25: **end switch**
26: **if** $|S| = l^*$ **then**
27: Output (S, f^*)
28: **end if**

If $\gamma < \infty$, property (1) implies that S appears consecutively in S^+. The spuriousness of S along with property (3) imply that there is either pivot w^+ or the start/end of the sequence to the left and right of the occurrence of S. Continuing the previous example, ab occurs consecutively in its primary witness abc and is enclosed by the start of the sequence to the left and pivot c to the right. Similarly, if $\gamma = \infty$, the spuriousness of S along with property (3) imply that S is obtained from S^+ by dropping all pivots w^+ from S^+.

We are now ready to describe the second step of MG-FSM$^+$ (lines 12–28 of Algorithm 3) which removes spurious sequences. The step is divided into a partitioning phase which matches primary witnesses with their spurious sequences, and a filtering phase which produces the final output.

*Partitioning phase (*MAP2*).* We map over those locally maximal sequences obtained in the first step of MG-FSM$^+$ (sequences of length at least 2) as well as over the f-list (length 1). For each sequence S^+, we generate the set of sequences for which S^+ can potentially be a primary witness. By the arguments immediately following Lemma 5.6, there is only a small set of such sequences. In more detail, set $w^+ = p(S^+)$ and divide S^+ into nonempty *chunks* $S_1, \ldots, S_n$ by splitting at pivots, that is,

$$S^+ = (w^+)^* S_1 (w^+)^+ S_2 (w^+)^+ \ \ldots \ (w^+)^+ S_n (w^+)^*,$$

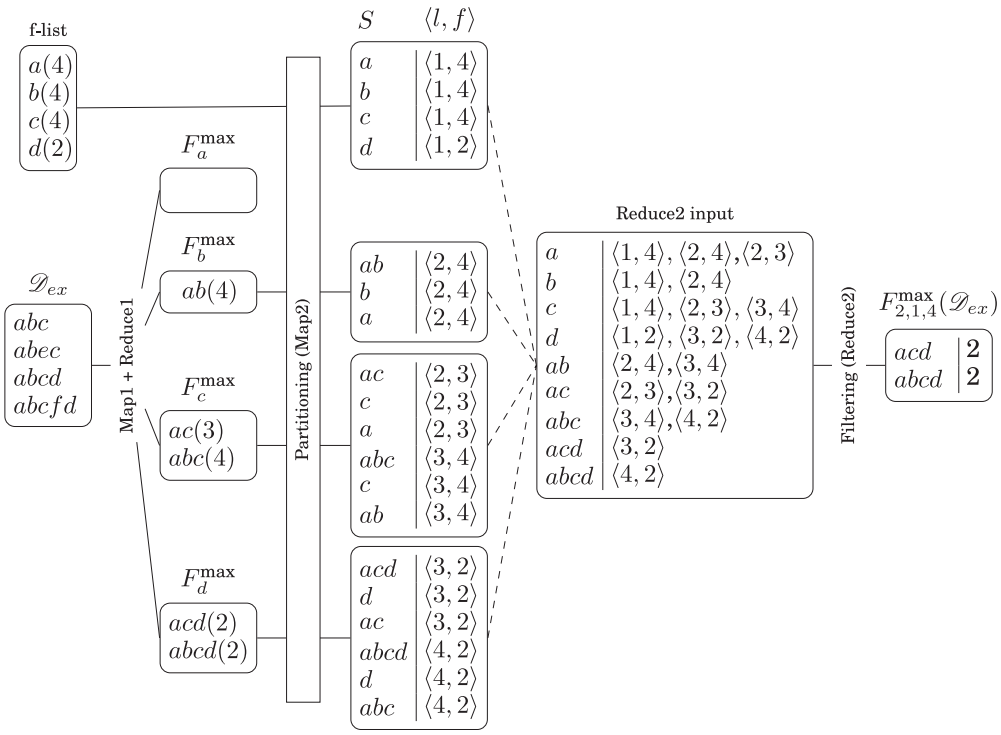


Fig. 1. Mining maximal sequences with MG-FSM$^+$ ($\sigma = 2$, $\gamma = 1$, $\lambda = 4$).

such that $p(S_i) < w^+$; here $(w^+)^*$ ($(w^+)^+$) denotes 0 or more (1 or more) occurrences of pivot w^+. Denote by

$$W_\gamma(S^+) = \{S^+\} \cup \{w^+\} \cup \begin{cases} \{\, S_1 S_2 S_3 \ldots S_n \,\} & \text{if } \gamma = \infty \\ \{\, S_1, S_2, S_3, \ldots, S_n \,\} & \text{if } \gamma < \infty, \end{cases}$$

the set of sequences for which S^+ can be a primary witness, as well as w^+ and S^+ itself. Note that we include pivot w^+ because, if $|S^+| \geq 2$, S^+ proves 1-sequence w^+ from the f-list to be spurious. For example, for sequence $S^+ = abcdbb$ with $p(S^+) = d$, we have $W_\infty(S^+) = \{\, abcdbb, d, abcbb \,\}$ and $W_1(S^+) = \{\, abcdbb, d, abc, bb \,\}$.

We emit a key-value pair for every sequence $S \in W_\gamma(S^+)$: the key is S, and the value is fixed to the pair of length and frequency of S^+, that is, we output $(S, \langle |S^+|, f_\gamma(S^+, \mathscr{D})\rangle)$. Figure 1 shows the output of Map2 for our example database (by partition and by key). Note that only key-value pair $(S^+, \langle |S^+|, f_\gamma(S^+, \mathscr{D})\rangle)$ has the length of the key equal to the length recorded in the value; for all other key-value pairs $(S, \langle l, f\rangle)$, we have $S \neq S^+$ and $l = |S^+| > |S|$. The total length of all emitted key-value pairs is linear in total length of the set of locally maximal sequences.

Filtering phase (Reduce2). The reduce phase processes independently each sequence output as a key in Map2. These sequences consist of all frequent 1-sequences (from the f-list), all locally maximal sequences, and some additional sequences contained in the set $W_\gamma(S^+)$ of some frequent sequence S^+. For each sequence S, we are given the corresponding *evidence set* $E(S) = \{\langle l, f\rangle\}$ of (length, frequency)-pairs as input. We first determine whether or not S is a frequent 1-sequence or a locally maximal sequence. In particular, if there is no pair $\langle l_S, f_S\rangle \in E(S)$ such that $l_S = |S|$, then S is a member of some set $W_\gamma(S^+)$ but is itself neither a frequent 1-sequence nor locally maximal. Thus

we do not output S. Otherwise, there is a pair $\langle l_S, f_S \rangle \in E(S)$ such that $l_S = |S|$; this pair was produced when processing $S^+ = S$ in MAP1 so that $f_s = f_\gamma(S^+, \mathscr{D}) = f_\gamma(S, \mathscr{D})$. We now need to determine whether S is globally maximal. If there is any additional pair $\langle l^+, f^+ \rangle$ in $E(S)$, then this pair must have been generated from a primary witness S^+ for S (with $S^+ \neq S$ and $S \in W_\gamma(S^+)$) and we have $l^+ > |S|$. We conclude that S is spurious. Finally, if there is no such pair, then S does not have a primary witness. Lemma 5.6 then implies that S is globally maximal so that we output (S, f_S). All of the preceding steps can be performed jointly as follows: Select any pair $\langle l^*, f^* \rangle$ of maximum length from $E(S)$ and output (S, f^*) if and only if $|S| = l^*$. Figure 1 shows the output of REDUCE2 for our example database. Sequences *acd* and *abcd* are the only globally maximal sequences in this example; these sequences are correctly identified by our approach.

In our implementation of the second step of MG-FSM$^+$, we further improve efficiency by making use of the combine functionality of MapReduce. In particular, our combine function mirrors REDUCE2; the key difference is that we only and always output the key-value pair $(S, \langle l^*, f^* \rangle)$. Combiners thus prune length-frequency pairs that are not needed in REDUCE2 so that correctness is maintained. Our use of combiners reduces the communication cost between the map and reduce phases as well as the computational cost in the reduce phase itself.

5.3. Mining Closed Sequences

MG-FSM$^+$ can also be used to mine closed sequences using a similar approach as described before; see Algorithm 3. We outline the key differences in this section.

In the first step, we mine and output the set $F_w^{closed}(\mathscr{P}_w)$ of *locally closed* sequences in each partition $\mathscr{P}_w$, where

$$F_w^{\text{closed}}(\mathscr{P}_w) = \left\{ S \in F_w(\mathscr{P}_w) \mid \neg\exists S' \in F_w(\mathscr{P}_w) : S \subset_{\gamma^-} S' \wedge f_\gamma(S, \mathscr{P}_w) = f_\gamma(S', \mathscr{P}_w) \right\}.$$

As before, we can use any closed sequence miner to obtain this set (line 5 of Algorithm 3). For our example database, we obtain

$$\begin{aligned} F_a^{\text{closed}}(\mathscr{P}_a) &= \emptyset, \\ F_b^{\text{closed}}(\mathscr{P}_b) &= \{ab(4)\}, \\ F_c^{\text{closed}}(\mathscr{P}_c) &= \{ac(3), abc(4)\} \text{ and} \\ F_d^{\text{closed}}(\mathscr{P}_d) &= \{acd(2), abcd(2)\}. \end{aligned}$$

These sets (coincidentally) agree with the corresponding sets of maximally closed sequences shown in Figure 1. (The set of globally closed sequences for our example database is given in Eq. (7).)

In the second step, we determine *globally closed* sequences by identifying and eliminating all spurious sequences, that is, sequences that are locally but not globally closed (*ab* is spurious in our example). We use a slightly different notion of witness: a locally closed sequence S^+ is a *potential witness* of the spuriousness of S if as before $S \subset_{\gamma^-} S^+$; it is a *witness* if additionally $f_\gamma(S, \mathscr{D}) = f_\gamma(S^+, \mathscr{D})$. As in the case of maximality, we can show there must exist a primary witness for each spurious sequence; the proof is similar to Lemma 5.6 and omitted here. For example, the sequence $ab \in F_b^{closed}(\mathscr{P}_b)$ is spurious; its witness (and primary witness) is sequence $abc \in F_c^{closed}(\mathscr{P}_c)$ with $f_1(ab, \mathscr{D}_{ex}) = f_1(abc, \mathscr{D}_{ex}) = 4$. As another example, the sequence $ac \in F_c^{closed}(\mathscr{P}_c)$ is not spurious; the only potential witness is $acd \in F_d^{closed}(\mathscr{P}_d)$, but *acd* has incorrect frequency ($f_1(ac, \mathscr{D}_{ex}) = 3 \neq 2 = f_1(acd, \mathscr{D}_{ex})$) so that it is not a witness.

To eliminate spurious sequences, we map over the locally closed sequences as well as over the f-list exactly as in the case of maximality (MAP1); that is, we output for each

sequence S^+ the set $W_\gamma(S^+)$ of sequences for which S^+ can be a primary witness. We then check in the filtering phase for each sequence S whether: (1) there is a potential witness (as before) that (2) agrees in frequency with S and thus is also a witness (new). Consider a sequence S and its corresponding evidence set $E(S) = \{ \langle l, f \rangle \}$. We select from $E(S)$ the pair $\langle l^*, f^* \rangle$ having highest frequency; we break ties by selecting the pair of maximum length. We then output (S, f^*) if and only if $|S| = l^*$. To see that this approach correctly determines globally closed sequences, assume that S is locally closed and spurious. Observe that each potential witness S^+ of S satisfies $f_\gamma(S^+, \mathscr{D}) \leq f_\gamma(S, \mathscr{D})$ by Lemma 5.1; equality holds if and only if S^+ is also a witness. Thus, if there is no witness for S, then S is the unique sequence of highest frequency in $E(S)$; we have $\langle l^*, f^* \rangle = \langle |S|, f_\gamma(S, \mathscr{D}) \rangle$ and thus output $(S, f_\gamma(S, \mathscr{D}))$. Otherwise, if there is a witness S^+, then $f_\gamma(S^+, \mathscr{D}) = f_\gamma(S, \mathscr{D})$ and our tie-breaking strategy applies. Since $|S^+| > |S|$, we select $\langle l^*, f^* \rangle = \langle |S^+|, f_\gamma(S^+, \mathscr{D}) \rangle$ and thus do not output S.

As mentioned previously, the input to the filtering step in our running example is coincidentally the same as in the case of maximal sequence mining; see Figure 1 under "Reduce2 input". First consider spurious sequence $S = ab$. We have $E(S) = \{ \langle 2, 4 \rangle, \langle 3, 4 \rangle \}$ and thus select $\langle l^*, f^* \rangle = \langle 3, 4 \rangle$ (produced from $W_1(abc)$). Since $l^* = 3 \neq 2 = |S|$, we conclude S is spurious. As another example, consider the globally closed sequence $S = ac$ with $E(S) = \{ \langle 2, 3 \rangle, \langle 3, 2 \rangle \}$. Here we select pair $\langle l^*, f^* \rangle = \langle 2, 3 \rangle$ (from $W_1(S)$), which is the unique pair of highest frequency. Since $l^+ = 2 = |S|$, we conclude S is not spurious and output S. In fact, pair $\langle 3, 2 \rangle \in E(S)$ has been generated from $W_1(acd)$; we correctly identify that acd is not a witness (even though it is a potential witness). The set of globally closed sequences obtained by our approach is given in Eq. (7).

6. MG-FSM IN PRACTICE

In this section, we give some guidance on how MG-FSM can be implemented efficiently in practice. The source code of our implementation is publicly available at MG-FSM [2014].

6.1. Compression

Careful compression of all intermediate sequences significantly boosted performance in our experiments. In particular, we assign an integer item identifier to each item and represent sequences compactly as arrays of item identifiers. We compress each such array using variable-byte encoding [Witten et al. 1999]. To make most effective use of this technique, we assign item identifiers in descending order of item frequency (as obtained from the f-list). More frequent items thus receive smaller item identifiers that can be represented using fewer bytes. Moreover, we replace irrelevant items by blanks (identifier -1) and use a form of run-length encoding [Witten et al. 1999] to represent consecutive sequences of blanks (e.g., ␣␣ by -2).

6.2. Grouping Partitions

Instead of creating a single partition for each distinct pivot, we greedily combine multiple pivots into a single partition such that each partition contains at least m sequences (e.g., $m = 10{,}000$). The goal of grouping is to increase efficiency by combining multiple small partitions into a larger one. Grouping is performed by scanning the f-list in descending order of frequency and combining items until their aggregate frequency exceeds m.

6.3. Aggregation

To implement aggregation (Section 4), we make use of both the combine function and the secondary sort facility of Hadoop. Recall Algorithm 1, in which we output pairs of

form (w, S), where w is a pivot and S a sequence. In our actual implementation, we output (pivot/sequence, frequency)-pairs, that is, pairs of the form $([w, S],\ 1)$ instead. We customize the partitioning function of MapReduce to ensure that w is used as partitioning key as before. This representation allows to use the combine function to aggregate multiple occurrences of the same sequence locally. We perform the final aggregation in the reduce function before invoking our FSM method (exploiting secondary sort to avoid buffering).

6.4. Frequent Sequence Mining

MG-FSM can use any existing FSM method to mine one of its partitions. In our implementation, we use a method inspired by GSP [Srikant and Agrawal 1996] and SPADE [Zaki 2001b]. Like the former, our method is based on a level-wise approach and generates frequent $(k+1)$-sequences from the frequent k-sequences; like the latter, our method operates on what is essentially an inverted index for sequences. Initially, while reading the sequences of a partition to be processed, our method builds an inverted index that maps 2-sequences to their offsets in the input sequences. We can then emit all frequent 2-sequences and remove infrequent ones from the index. Afterwards, our method iteratively combines frequent k-sequences by intersecting the corresponding inverted index lists to construct an inverted index for $(k+1)$-sequences. Frequent $(k+1)$-sequences can be emitted immediately once they have been determined. The inverted index for k-sequences can be deleted at the end of each iteration. Further efficiency can be gained by replacing the general-purpose sequence miner described earlier by a local sequence miner that targets MG-FSM's partitioning and mines pivot sequences (and only pivot sequences) directly; see the pivot sequence miner of Beedkar and Gemulla [2015] for an example of such a method. We do not consider such specialized methods in this article.

Our implementation is aware of the aggregation optimization described in Section 4, that is, it operates directly on input sequences along with their aggregate frequency. Although Java based, our implementation avoids object creation to the extent possible. Inverted index lists, for instance, are encoded compactly as byte arrays using the compression techniques described before.

To mine maximal sequences in each partition, we proceed as previously but do not immediately output frequent k-sequences when we find them. Instead, we delay output until after we have computed all frequent $(k+1)$-sequences; a frequent k-sequence S is output if there is no frequent $(k+1)$-sequence built from S (or if $k = \lambda$). In more detail, we mark, during the construction of the index of $(k+1)$-sequences, each k-sequence that contributed to a frequent $(k+1)$-sequence. Once the new index is constructed, we output all unmarked frequent k-sequences before deleting the k-sequence index. This procedure can also be adapted to mine all closed sequences. The key difference is that we mark a frequent k-sequence S as nonclosed if it contributes to a frequent $(k+1)$-sequence with the same support as S.

The simple implementation of maximal and closed sequence mining described earlier worked reasonably well in our experiments. Since, as before, we can use any suitable algorithm as a local sequence miner, further efficiency can be gained by using a more efficient maximal or closed sequence miner; see Pei et al. [2004], Fournier-Viger et al. [2013], Wang and Han [2004], and Li and Wang [2008] for examples.

6.5. Long Sequences

We have assumed so far that the sequences in the sequence database are relatively short (e.g., sequences that correspond to sentences in text mining). This allows to scan with low cost the entire input sequence repeatedly during partition construction. The assumption of short sequences does not generally hold, that is, in some applications

sequences can be very long (e.g., sequences that correspond to entire documents in text mining). To handle long sequences, we need to ensure that the rewriting techniques of MG-FSM remain efficient as the length of the input sequences increases.

Recall that, for an input sequence T, our rewriting methods perform a backward and a forward scan for each pivot item $w \in T$ (see Section 4.5). The total number of backward-forward scans depends on the number of distinct (frequent) items in T, but generally can be as high as $|T|$. Since the scans take linear time, the computational cost is $O(|T|^2)$. This quadratic overall cost is acceptable when T is short ($|T|$ small), but imposes severe overheads when T is long ($|T|$ large). To avoid this performance bottleneck, we propose to build for each input sequence T an *inverted index* structure that maps pivot items to their respective positions in T. By utilizing such an index, we avoid performing a full backward-forward scan and instead perform a focused scan "around" the occurrences of the pivot under consideration.

In more detail, we perform an initial pass over input sequence T to build an inverted index. The index stores, for each distinct frequent item $w \in T$, the positions of w's occurrences in T.[8] Given such an index, we construct $\mathscr{P}_w(T)$ for each indexed pivot w as follows. Instead of scanning T, we only consider those parts of T that are sufficiently close to w; we ensure that all omitted parts cannot contribute to a pivot sequence. In more detail, denote by $I = (i_1, \ldots, i_r)$ the positions at which w occurs in T. We then restrict the forward-backward scan to the union of the ranges $[i_j - (\gamma + 1)(\lambda - 1), i_j + (\gamma + 1)(\lambda - 1)]$ for $1 \leq j \leq r$. All items outside of these ranges are unreachable and therefore do not need to be considered (cf. Lemma 4.4).

There is a trade-off between the construction cost and the benefit of the inverted index. We have described previously an inverted index that maintains all positions of each pivot item; we subsequently refer to this index as *full* index. A simple alternative is to use a *min-max* index, which is more efficient to construct. In particular, the min-max index maintains only the positions of the first and the last occurrence of each pivot. Given such an index, we perform for pivot w a forward-backward scan of range $[l_w - (\gamma + 1)(\lambda - 1), r_w + (\gamma + 1)(\lambda - 1)]$, where l_w and r_w are the positions of the first and last occurrences of w. Observe that, if there is only one occurrence of w, we scan identical ranges when using either the min-max index or the full index. If there are multiple occurrences of a pivot, the full index can be more effective than the min-max index (especially when the left- and rightmost occurrences are far apart).

We now illustrate the effect of the inverted index on the processing cost of a backward-forward scan using an example. Consider sequence $T = cadbaefebdaecdgadfae$, pivot c, $\gamma = 1$, and $\lambda = 3$. To construct partition $\mathscr{P}_c(T)$ without an index, we need to scan T twice and thus process $2|T| = 40$ items. If we use the min-max index, we restrict our scans to the neighborhood of c's first occurrence ($l_c = 1$) and c's last occurrence ($r_c = 13$). We scan twice the range $[1 - 2 \cdot 2, 13 + 2 \cdot 2]$ (i.e., range $[1, 17]$) for a total of 34 processed items. Finally, if we use a full index, we scan twice the ranges $[1, 5]$ (for the occurrence of c at position 1) and $[9, 17]$ (position 13). The total number of processed items is 28.

In our experiments (Section 7), we study how the use of inverted indexes affects the performance of our rewriting methods (i.e., cost of the map phase). Our results indicate large performance gains when inverted indexes are used for long sequences.

6.6. Temporal Sequences

We have restricted attention so far to frequent sequence mining with a positional gap constraint. In applications such as session analysis, however, input sequences are often built from time-annotated events instead of items; in such applications, temporal gap

[8]We use the term "position" instead of "index" to avoid confusion with the index structure.

constraints are more suitable [Srikant and Agrawal 1996]. This means that we want to treat a pair of events as sufficiently close if the in-between time span is small (e.g., events that occur within 1 hour), that is, independently of the number of events that occur in between. In this section, we describe how MG-FSM can be adapted to support such temporal gap constraints.

Definition 6.1. A *temporal sequence* is an ordered list $T = s_1(t_1)\,s_2(t_2)\cdots s_l(t_l)$ of *events*, that is, item-timestamp pairs. For $1 \leq i \leq l$, event $s_i(t_i)$ consists of item s_i taken from dictionary Σ and timestamp t_i taken from a discrete set of timestamps $\mathcal{T} \subseteq \mathbb{Z}$. The timestamps are distinct and ordered, that is, $t_i < t_j$ for $1 \leq i < j \leq l$.

Note that the timestamps can be of any desired granularity (e.g., seconds, minutes, hours, or days).

Denote by $\Delta_{ij} = t_j - t_i - 1$ the *temporal gap* between events $s_i(t_i)$ and $s_j(t_j)$, $i < j$. Observe that $\Delta_{ij} \geq 0$. To handle temporal gap constraints, we "convert" the temporal to a positional gap constraint by mapping the event sequence into a sequence of items and gaps. In particular, we convert temporal sequence $T = s_1(t_1)\,s_2(t_2)\cdots s_l(t_l)$ to regular sequence

$$T' = s_1 \textvisiblespace^{\Delta_{12}} s_2 \textvisiblespace^{\Delta_{23}} s_3 \cdots \textvisiblespace^{\Delta_{(l-1)l}} s_l.$$

Here, $\textvisiblespace$ denotes the blank symbol as before; $\textvisiblespace^{\Delta_{i(i+1)}}$ represents as many gaps as time units (without an event occurring) passed between events $s_i(t_i)$ and $s_{i+1}(t_{i+1})$. If two events occur at adjacent timestamps (i.e., their temporal gap is 0), then no blanks appear between the corresponding items. As described in Section 6.1, we reduce the overhead of adding gaps to the input sequences by using a compression technique that encodes sequences of consecutive blanks with run-length encoding. With such compression, the conversion takes linear time and space.

To mine frequent sequences with a temporal gap constraint, we simply run MG-FSM with a positional gap constraint on the converted sequences. In particular, denote by τ a maximum temporal-gap parameter; at most τ time units are thus allowed to pass between two events to be considered close. We then set maximum-gap parameter $\gamma = \tau - 1$ when running MG-FSM. To see why this produces the desired result, observe that our rewrite ensures that two events $s_i(t_i)$ and $s_j(t_j)$, $i < j$ with temporal gap Δ_{ij} have exactly Δ_{ij} items or blanks in-between them. The time passed between the occurrence of $s_i(t_i)$ and $s_j(t_j)$ is $\Delta_{ij} + 1$; for this reason, we set $\gamma = \tau - 1$ (instead of $\gamma = \tau$).

Consider, for example, the temporal sequence database $\mathscr{D} = \{a(2)\,b(3)\,c(6)\,a(8)\}$, which consists of a single temporal sequence with four events. After conversion, we obtain sequence database $\mathscr{D}' = \{ab\textvisiblespace\textvisiblespace c\textvisiblespace a\}$ by dropping timestamps and adding the respective number of blanks. Set $\sigma = 1$ and $\lambda = 3$. For a temporal gap constraint of one time unit ($\tau = 1$, and thus $\gamma = 0$), we obtain frequent sequences $F_{1,0,3}(\mathscr{D}') = \{a, b, c, ab\}$. For $\tau = 2$ ($\gamma = 1$), we obtain $F_{1,1,3}(\mathscr{D}') = \{a, b, c, ab, ca\}$. Finally, for $\tau = 3$ ($\gamma = 2$), we obtain $F_{1,2,3}(\mathscr{D}) = \{a, b, c, ab, bc, ca, abc, bca\}$.

As a final note, we remark that MG-FSM can also handle temporal sequences with repeated timestamps, provided that items with equal timestamps have meaningful order (but the granularity of the timestamp is too coarse-grained to capture this order). To do so, we modify both timestamps and the maximum-gap parameter in a way that makes timestamps unique, retains the original order of the events, and ensures correct results. In more detail, we conceptually multiply each timestamp by $2r - 1$ before conversion, where $r > 1$ is an upper bound on the number of events that can occur simultaneously. We then replace repeated timestamps by consecutive sequences of timestamps. For example, the sequence of events $a(1)\,b(1)\,c(2)\,d(2)\,e(2)$ is modified to $a(5)\,b(6)\,c(10)\,d(11)\,e(12)$ for $r = 3$. Now all timestamps are distinct; we convert the

database as described previously, obtaining $ab\_\_\_cde$ for our example. We run MG-FSM with maximum-gap parameter

$$\gamma = (\tau - 1)(2r - 1) + (3r - 3).$$

Here $3r - 3$ denotes the maximum positional gap between two events that originally occurred at consecutive timestamps. In our example, where $r = 3$, we set $\gamma = 1$ for $\tau = 0$ and $\gamma = 6$ for $\tau = 1$.

When $\gamma = 0$, a temporal rewrite can also be used to mine sequences of itemsets (as opposed to sequences of items, which is the focus of this article) with MG-FSM. To see this, consider the sequence of itemsets $T = \langle ac \rangle \rightarrow \langle b \rangle$, where we enclose itemsets by $\langle$ and $\rangle$ and separate itemsets by $\rightarrow$. Then T has the following nonempty subsequences of itemsets

$$\langle a \rangle, \langle b \rangle, \langle c \rangle, \langle ac \rangle, \langle a \rangle \rightarrow \langle b \rangle, \langle c \rangle \rightarrow \langle b \rangle, \langle ac \rangle \rightarrow \langle b \rangle.$$

We can rewrite T to a temporal sequence T' such that every subsequence of itemsets of T corresponds to a subsequence of items of T' and vice versa. To do so, we first flatten the itemsets to an item sequence and put a special *itemset marker item* $\rightarrow$ between itemsets. We obtain the sequence of items $T_1 = ac{\rightarrow}b$, which consists of four items. We now associate a timestamp with each item, starting from 1 and incrementing by 1 at each occurrence of the itemset marker item as well as at its consecutive item. We then reorder same-timestamp events lexicographically. This gives us the event sequence $T' = a(1)\,c(1){\rightarrow}(2)\,b(3)$. With a choice of $\tau = 1$, T' generates the sequences of items

$$a, b, c, ac, a{\rightarrow}b, c{\rightarrow}b, ac{\rightarrow}b,$$

as well as some additional sequences that start or end with $\rightarrow$; we ignore these additional sequences. Then T and T' have equivalent subsequences. Given a database of sequences of itemsets, we rewrite every sequence as just described, apply MG-FSM, and filter out the additional sequences. Although this technique correctly mines sequences of itemsets, it is limited to the case $\gamma = 0$ and care must be taken to support length constraints correctly. We thus do not explore this direction further.

7. EXPERIMENTAL EVALUATION

We conducted an extensive experimental study in the contexts of text mining and session analysis on large real-world datasets. In particular, we investigated the effectiveness of the various partition construction techniques of Section 4, studied the impact of parameters σ, γ, and λ, compared MG-FSM to the sequential FSM algorithms, the naïve algorithm, and a state-of-the-art n-gram miner, and evaluated weak and strong scalability of MG-FSM. We also studied the performance of MG-FSM$^+$ for mining maximal or closed sequences, the effectiveness of our indexing techniques for handling long input sequences, and MG-FSM's performance for mining temporal sequences.

We found that most of our optimizations for partition construction were effective; a notable exception was blank separation, which did not provide substantial efficiency gains on our text datasets (see the discussion at the end of Section 4.4). MG-FSM outperformed the naïve and sequential FSM algorithms by multiple orders of magnitude and was up to 2x faster than the state-of-the-art n-gram miner (which does not support gap constraints). Our scalability experiments suggest that MG-FSM scales well as we add more worker nodes and/or increase the dataset size. MG-FSM$^+$ mined maximal and closed sequences with only a small additional cost compared to mining all frequent sequences. Our use of inverted indexes for mining long input sequences was effective and significantly decreased rewriting costs. Finally, we observed that MG-FSM successfully mined temporal sequences on the Netflix dataset [Bennett and Lanning

Table I. Dataset Characteristics

	ClueWeb	NYT-sen	NYT-doc	Netflix
Average length	19	19	603	266
Maximum length	20,993	21,174	38,917	7,966
Total sequences	1,135,036,279	53,137,507	1,830,592	398,820
Total items	21,565,723,440	1,051,435,745	1,051,435,745	106,145,170
Distinct items	7,361,754	1,577,233	1,577,233	17,769
Total bytes	66,181,963,922	3,087,605,146	3,087,605,146	608,347,782

2007]; temporal sequence mining can be expensive, however, when the data contains large bursts of events in small timespans.

7.1. Experimental Setup

We implemented MG-FSM [MG-FSM 2014] as well as the naïve method of Section 2.3 in Java. For our sequential experiments, we used an implementation of PrefixSpan tuned for our setting.[9] We also obtained a Java implementation of Suffix-σ [Berberich and Bedathur 2013], a state-of-the-art n-gram miner, from its authors. Unless stated otherwise, we performed all of our rewriting steps for MG-FSM.

Sequential setup. We ran sequential FSM algorithms on a machine equipped with two Intel Xeon E5530 quad-core processors and 48GB of RAM running Debian Linux 7.5.

Hadoop cluster. We ran our experiments on a local cluster consisting of 11 Dell PowerEdge R720 computers connected using a 10GBit Ethernet connection. Each machine has 64GB of main memory, eight 2TB SAS 7200 RPM hard disks, and two Intel Xeon E5-2640 6-core CPUs. All machines ran Debian Linux (kernel version 3.2.48.1.amd64-smp), Oracle Java 1.7.0_25, and used the Cloudera cdh3u6 distribution of Hadoop 0.20.2. One machine acted as the Hadoop master node, the other 10 machines acted as worker nodes. The maximum number of concurrent map or reduce tasks was set to 8 per worker node. All tasks launched with 4GB heap space.

Datasets. We used three real-world datasets for our experiments; see Table I. The first dataset is a subset of ClueWeb [2009] (CW), which consists of 50 million English pages. This well-defined subset is commonly referred to as ClueWeb09-T09B and is also used as a TREC Web track dataset. We performed sentence detection using Apache OpenNLP and applied boilerplate detection and removal as described in Kohlschütter et al. [2010]. Each resulting sentence was then treated as an input sequence. Our second dataset is the New York Times corpus (NYT) [The New York Times 2008] which consists of over 1.8 million newspaper articles published between 1987 and 2007. We created two sequence databases from this corpus, denoted NYT-sen and NYT-doc, in which we respectively treat each sentence or each document as an input sequence. The sequences in NYT-doc are substantially longer than the ones in NYT-sen; we thus use NYT-doc to evaluate the effectiveness of our indexing techniques for long sequences (see Section 6.5). Our final dataset is the Netflix dataset [Bennett and Lanning 2007] which we use to evaluate our approach for mining temporal sequences. The Netflix data contains more than 100M ratings from 480k users for around 18k movies; each rating is annotated with a timestamp. We constructed a temporal database from this data by creating a temporal sequence for each user; this sequence consists of (timestamp, movie)-pairs ordered by timestamp. Since the Netflix dataset contains a few heavy-raters, with up to 5,500 ratings on a single day, we exclude these users from our dataset to ensure a meaningful output and keep runtimes manageable. All datasets were represented compactly as sequences of item identifiers as described in Section 6.

[9]We tried using the same SPADE-based FSM algorithm and implementation as in the local miner of MG-FSM, but it consistently ran out of memory on large data.

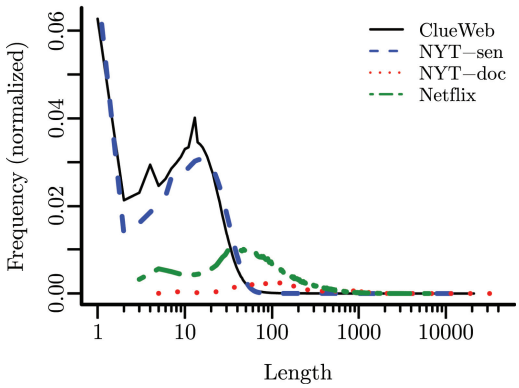


Fig. 2. Distribution of lengths of input sequences.

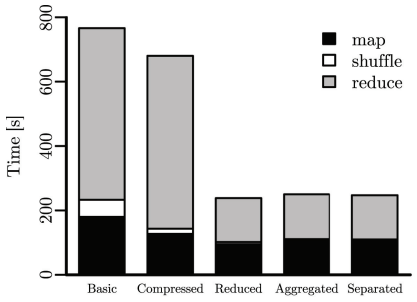
Figure 2 shows the length distribution of sequences in each of these datasets; note that the x-axis has a logarithmic scale. The length distribution is right-skewed in all datasets. The CW and the NYT-sen datasets contain shorter sequences: the mean length is 19 and more than 99% of the sequences have length less than 100. The NYT-doc and the Netflix datasets contain longer sequences: the mean lengths are 603 and 266, respectively, and the percentages of sequences with length less than 100 are 14% and 38%, respectively.

Methodology. We used the following measures in our evaluation. First, we measured the total time elapsed between launching a method and obtaining the result. We assume throughout that the f-list is made available for each method. For our methods for mining maximal and closed sequences, which make use of two MapReduce jobs, we additionally specify the overall elapsed time per MapReduce job. To provide more insight into potential bottlenecks, we also broke down total time into time spent for the map phase, shuffle phase, and reduce phase. Since these phases are overlapping in MapReduce, we report the elapsed time until finishing each phase, for example, the time until the last map task finishes. Second, we measured the total number of bytes received by reducers. Note that when a combiner is used, this number is usually smaller than the total number of bytes emitted by the mappers. All measurements were averaged over three independent runs.

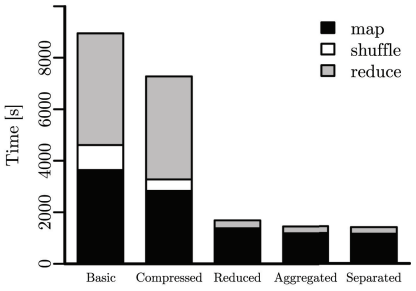
7.2. Partition Construction

We first evaluated the effectiveness of the compression and rewriting techniques of MG-FSM for partition construction. We used the following settings: *basic* (irrelevant items were replaced by blanks), *compressed* (consecutive blanks were compressed and leading/trailing blanks were removed), *reduced* (unreachable items were removed), *aggregated* (identical sequences were aggregated), and *separated* (blank separation was performed). We applied these optimizations in a stacked manner; for instance, when using the aggregation optimization, we also removed unreachable items and compressed consecutive blanks.

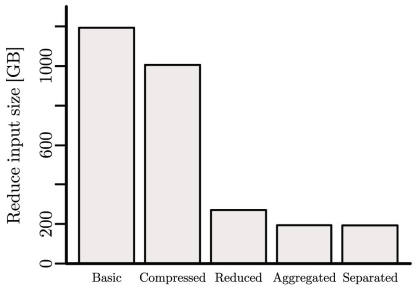
We used both NYT-sen and CW to explore the different rewriting techniques. The results are shown in Figures 3(a), 3(b), and 3(c), which also give the parameter settings. As can be seen, the removal of unreachable items resulted in a significant runtime improvement on both datasets, reducing the total time by a factor of up to 6 (for CW). For the smaller NYT-sen dataset, map tasks are relatively inexpensive throughout and our techniques mainly reduce the runtime of the much more costly reduce operations.



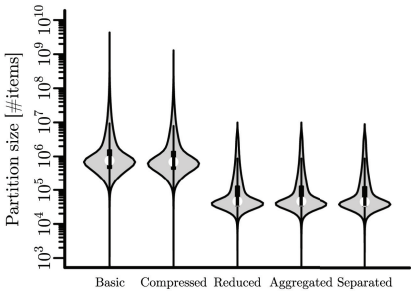
(a) NYT-sen ($\sigma = 100, \gamma = 1, \lambda = 5$)



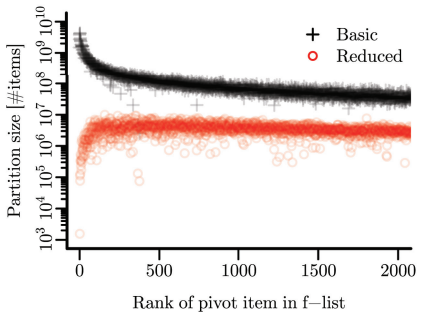
(b) CW ($\sigma = 1{,}000, \gamma = 0, \lambda = 5$)



(c) CW ($\sigma = 1{,}000, \gamma = 0, \lambda = 5$)



(d) CW ($\sigma = 1{,}000, \gamma = 0, \lambda = 5$)



(e) CW ($\sigma = 1{,}000, \gamma = 0, \lambda = 5$)

Fig. 3. Comparison of partition construction techniques.

For CW, both map and reduce runtimes are significantly reduced, the former mainly due to the large reduction in transferred bytes (Figure 3(c)). Aggregation is effective for CW, reducing the total number of bytes received by the reducers by more than 70GB (28%).

To gain further insight into the different alternatives for partition construction, we studied the distribution of partition sizes for the larger CW dataset. We measure the partition size by the sum of the lengths of the distinct sequences in the partition.[10] When the compressed representation is used, we count sequences of consecutive

[10]We consider distinct sequences because our implementation always aggregates repeated sequences before running the local miner, that is, whether or not aggregation with a combiner is used.

blanks as one item (recall that ␣␣␣ is then represented as -3). The results are shown in Figure 3(d) in terms of a violin plot. For each "violin", the white dot corresponds to the median partition size and the (small) black box to the range spanned by the 25%–75% quantiles. The gray area is a kernel density plot shown sideways; the wider it is, the more partitions fall into the corresponding range of partition sizes. The plot shows that partition sizes decrease by more than an order of magnitude when our reduction techniques are applied. There also a reduction in the variance of partition sizes: the coefficient of variation (standard deviation divided by mean) reduces from 9.6 for Basic to 2.6 for Reduced. We also analyzed the sizes of the partitions corresponding to the 2,000 most frequent items more closely; see Figure 3(e). As can be seen, the size of the respective largest partition is reduced by roughly three orders of magnitude when using Reduced instead of Basic. Also, the size of partitions is highly skewed for Basic (note the logarithmic scale): the largest partition is 117× larger than the 2,000-th largest partition. For Reduced, the partition sizes are significantly more stable: the largest partition is 4× larger than 2,000-th largest, which facilitates load balancing.

We also ran Naïve for the NYT-sen dataset (not shown); the algorithm finished after 225 minutes. In contrast, MG-FSM completes after 4 minutes and is thus more than 50 times faster.

7.3. Mining n-Grams

In our next experiments, we investigated the performance of MG-FSM for n-gram mining (i.e., $\gamma = 0$) and compared it against sequential FSM, the naïve method from Section 2, and a state-of-the-art approach called Suffix-σ [Berberich and Bedathur 2013]. We used both the NYT-sen and CW datasets and ran sequence mining in three different configurations of increasing output size; the results are shown in Figure 4.

We first evaluated all the methods on the (smaller) NYT-sen dataset (see Figure 4(a)). For the "easier" settings ($\sigma = 100, \lambda = 5$ and $\sigma = 10, \lambda = 5$), MG-FSM achieved an order of magnitude faster performance than Naïve. For the "harder" setting ($\sigma = 10$, $\lambda = 50$), MG-FSM was two orders of magnitudes faster than Naïve. MG-FSM also outperformed Suffix-σ in all settings (up to a factor of 1.6× faster). The total number of bytes transferred between the map and reduce phases is depicted in Figure 4(c); it is smallest for MG-FSM. We also observed that the sequential approach was faster than Naïve. Both MG-FSM and Suffix-σ were roughly an order of magnitude faster.

We also studied n-gram mining on the CW dataset (20× larger than NYT-sen). The sequential method ran out of memory for this dataset, and the naive method was too slow to produce acceptable runtimes. We thus focus on Suffix-σ and MG-FSM; the performance of these algorithms is shown in Figure 4(b) and Figure 4(d). Here MG-FSM was up to 2× faster and transferred up to 2× less data between map and reduce phases than Suffix-σ.

7.4. Impact of Parameter Settings

We studied the performance of MG-FSM as we varied the minimum support σ, the maximum gap γ, and the maximum length λ. We used the NYT-sen dataset throughout and set the default values to $\sigma = 100$, $\gamma = 1$, and $\lambda = 5$.

We first studied how the minimum support σ affects performance by increasing its value from 10 to 10,000. The results are shown in Figure 5(a). The map phase, which performs the rewriting of the input sequences, took roughly the same time for all different values of σ. This time mainly consists of the cost of scanning the input sequences, which is independent of σ. The reduce time, however, dropped significantly

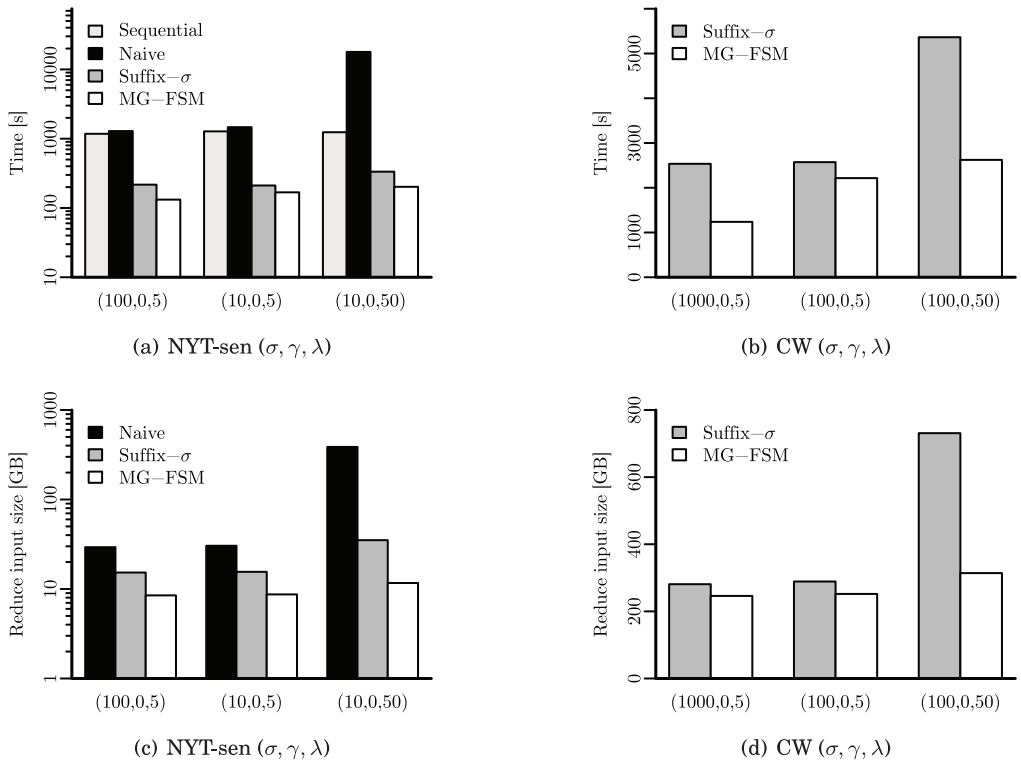


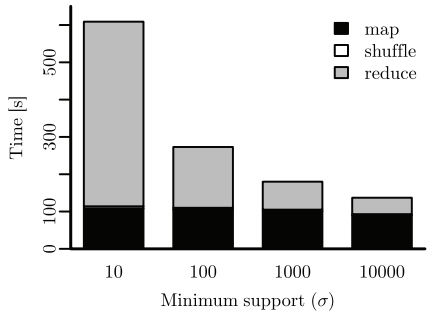
Fig. 4. Performance for n-gram mining.

as we increased the minimum support, mainly because the mining cost and output size reduced (the slowest reduce task took 44s for $\sigma = 10{,}000$). Due to the relatively large fraction of time spent in the map phase for large values of σ, we do not expect any significant runtime improvements if we increased σ further.

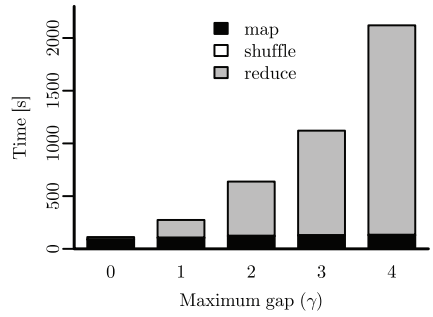
Second, we increased the maximum gap γ from 0 to 4. As we can see in Figure 5(b), γ strongly affected reduce time, while the impact on map time was again not significant. This was partly due to the larger number of bytes received by the reducers (see Figure 5(c)) and also because mining becomes harder when the output size increases: The total number of frequent sequences increased from 1,985,702 for $\gamma = 0$ to 51,166,966 for $\gamma = 4$.

Finally, we studied how the maximum length λ affects MG-FSM, varying its value from 5 to 20. As Figure 5(d) shows, λ had little effect on the map operations. Reduce time increased with increasing values of λ. This effect was less pronounced for larger values of λ (say, $\lambda \geq 10$). This is because we iteratively compute l-sequences by intersecting posting lists of corresponding $(l-1)$-sequences (see Section 6.4); as l increases, the posting lists become shorter and the cost of intersections reduces.

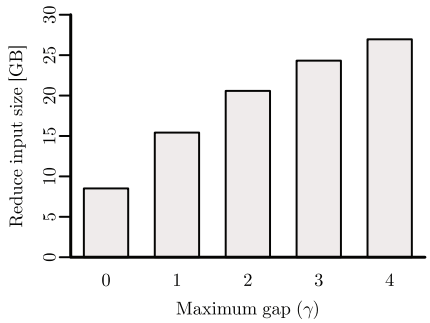
Table II lists some examples of frequent sequences that we mined from the NYT-sen dataset. The sequences include prominent named entities (sequence 5), frequent verbal phrases (sequences 1 and 2), popular quotes (sequence 6), and nouns and their common prepositions (sequences 3 and 4). The latter two sequences illustrate the effect of allowing positional gaps: the phrase "a flight" is frequently followed by the prepositions "from" and "to", but there is no particular flight from some place to some other place that is frequent (e.g., "flight from New York to Tokyo").



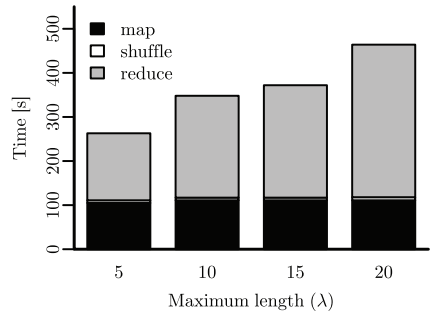
(a) NYT-sen ($\gamma = 1, \lambda = 5$)



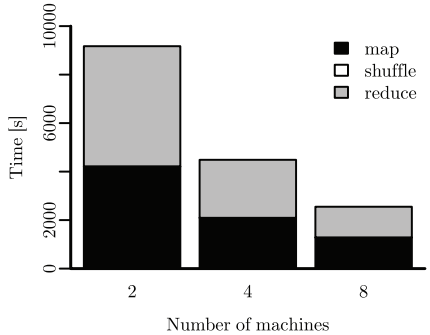
(b) NYT-sen ($\sigma = 100, \lambda = 5$)



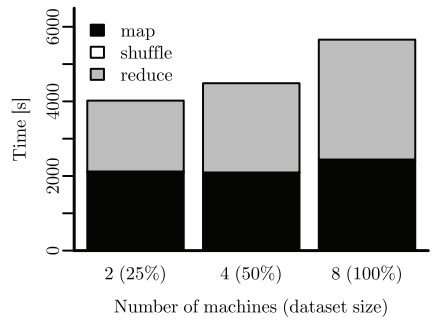
(c) NYT-sen ($\sigma = 100, \lambda = 5$)



(d) NYT-sen ($\sigma = 100, \gamma = 1$)



(e) 50% of CW ($\sigma = 1{,}000, \gamma = 1, \lambda = 5$)



(f) CW ($\sigma = 1{,}000, \gamma = 1, \lambda = 5$)

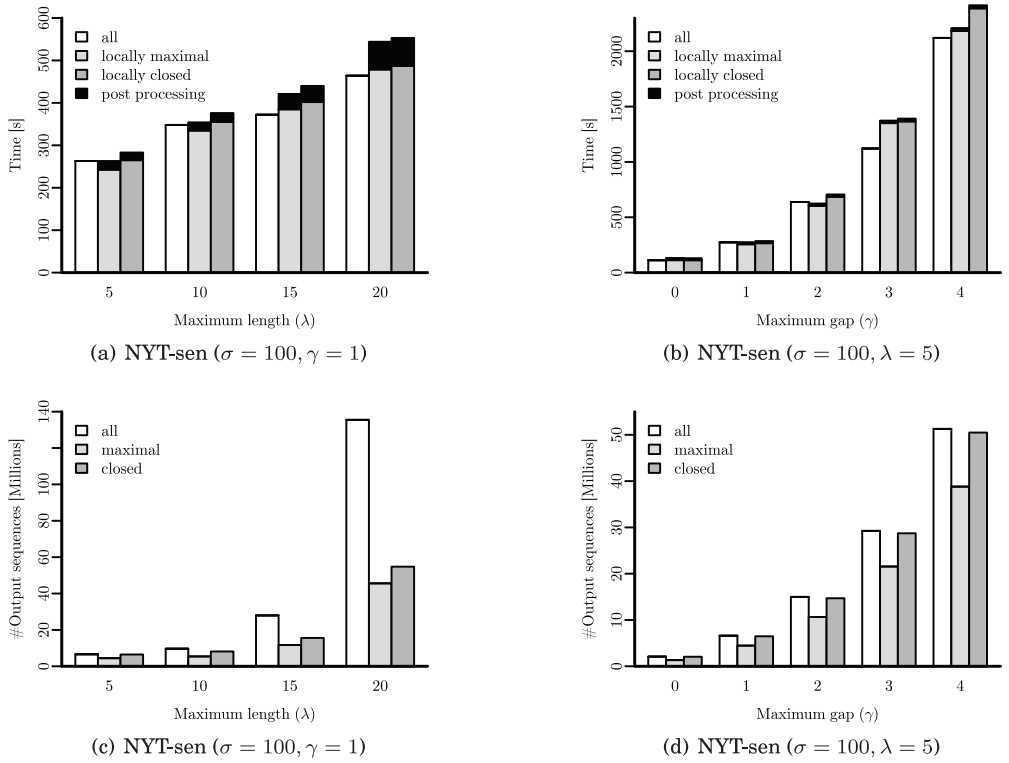
Fig. 5. Impact of parameter settings (a)–(d) and scalability results (e) and (f).

7.5. Scalability

In our next set of experiments, we explored the scalability of MG-FSM. To evaluate strong scalability, we ran MG-FSM on a fixed dataset using 2, 4, and 8 worker nodes ($\sigma = 1{,}000, \gamma = 1, \lambda = 5$). In order to finish the experiment in reasonable time on 2 nodes, we used a 50% sample of CW (consisting of more than half a billion input sequences). Our results are shown in Figure 5(e). MG-FSM exhibited linear scalability as we increased the number of available machines, managing to proportionally decrease the times for the map and reduce tasks. The ability of MG-FSM to scale up can be credited to the large number of partitions that can be processed and mined independently.

Table II. Example Frequent Sequences from the NYT-sen Dataset

Sequences (frequency)
(1) *studied at the university of* (914)
(2) *celebrate the anniversary of* (1021)
(3) *daughter of and of* (33094)
(4) *a flight from to* (1313)
(5) *senator barack obama of illinois* (185)
(6) *a thousand points of light* (117)



(a) NYT-sen ($\sigma = 100, \gamma = 1$)
(b) NYT-sen ($\sigma = 100, \lambda = 5$)
(c) NYT-sen ($\sigma = 100, \gamma = 1$)
(d) NYT-sen ($\sigma = 100, \lambda = 5$)

Fig. 6. Performance of mining maximal and closed sequences.

We also performed a weak scalability experiment for MG-FSM in which we simultaneously increased the number of available machines (2, 4, 8) and the size of the sequence database (25%, 50%, 100% of CW). In the ideal case, the total time remains constant as we scale out. As Figure 5(f) shows, this is almost true, but we observe a small increase in runtime on 8 worker nodes (around 20%). This is because doubling the size of the input sequence database can increase the number of output sequences by a factor larger than 2. In this specific case, 50% of ClueWeb generated 6M frequent sequences, whereas the full corpus generated 13.5M frequent sequences (a 2.25× increase).

7.6. Maximal and Closed Sequences

We evaluated the performance of MG-FSM$^+$ for mining maximal and closed sequences. Recall that MG-FSM$^+$, in contrast to MG-FSM, makes use of a postprocessing step to filter out spurious sequences. We report separately the time required to mine locally

maximal or closed sequences (first MapReduce job) and the time required for postprocessing (second job). In all experiments, we used the NYT-sen dataset and set the default values to $\sigma = 100$, $\gamma = 1$, and $\lambda = 5$.

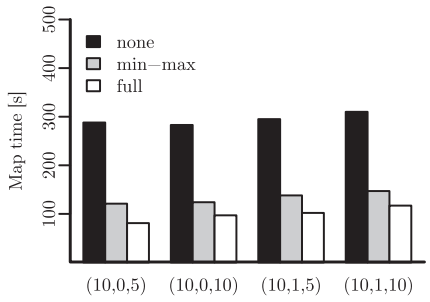
We first studied the performance of MG-FSM$^+$ for various choices of the maximum-length parameter λ, which we vary from 5 to 20. We set $\sigma = 100$ and $\gamma = 1$. Figure 6(a) shows the total runtime for mining all sequences, maximal sequences, as well as closed sequences. Figure 6(c) shows the corresponding number of output sequences. First, observe that, for large values of λ, the decrease in output size is significant (up to $3\times$ for maximality and $2.5\times$ for closedness); this shows that mining only maximal or closed sequences can be beneficial. Second, observe that the time required to mine maximal or closed sequences is close to that required to mine all frequent sequences, that is, the overhead of mining locally maximal or closed sequences as well as filtering spurious sequences in the postprocessing step is low. Finally, observe that the time required for postprocessing increases as we increase λ. As can be seen in Figure 6(c), large values of λ lead to a larger output sizes in all cases. This increase in output size translates to more work in the postprocessing step, which thus takes more time.

We also studied the impact of the maximum-gap parameter γ by varying its value from 0 to 4. We set $\sigma = 100$ and $\lambda = 5$. The results are shown in Figures 6(b) and 6(d). As before, the overhead of maximal or closed sequence mining (second job) was small. For large values of γ, we observed that the time required to mine locally maximal or closed sequences (first job) was slightly larger than that required to mine all sequences. This increase in runtime stems from our additional test for local maximality or closedness; we mine all sequences but only output the maximal and closed ones. This test took more time (up to 200s) as γ, and thus the number of sequences being processed and tested, increased. This problem is not inherent to MG-FSM$^+$: using a state-of-the-art maximal or closed sequence miner in the local mining step may reduce running time (see Section 6.4). Note that maximal and closed sequence mining was not particularly effective in reducing the output size for our choice of $\lambda = 5$. This happens because λ was comparably small and all sequences of length λ are maximal and closed.

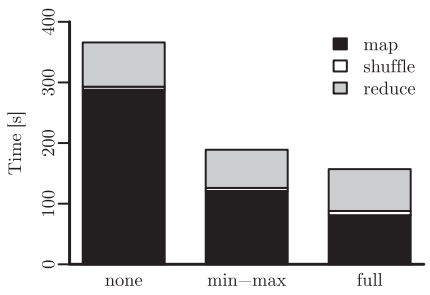
7.7. Long Sequences

In this group of experiments, we studied the performance of the indexing techniques of Section 6.5 for long sequences. We used the NYT-doc dataset in which each input sequence corresponds to an entire document. The average sequence length was 603 items; see Table I. We evaluated MG-FSM without indexing (termed *none*), with an index of the first and last position of each distinct item (*min-max*), as well as a full index of all positions (*full*). Recall that the goal of using indexes is to reduce the cost of rewriting (map phase of MG-FSM).

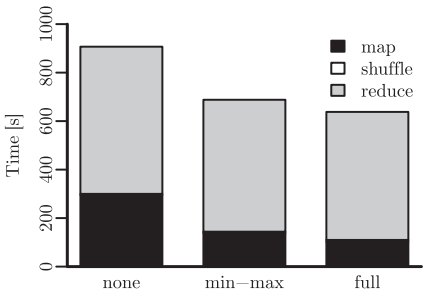
We first compared how the map time (i.e., the time until the last map tasks finished) was affected when the different kinds of indexes are used. We considered four configurations, and the results are shown in Figure 7(a). The benefit of the different indexes across different setups is similar: map time is mainly affected by the input, which remains the same, and is less sensitive to the parameters γ and λ. In all cases, the use of the min-max index reduced the total map time by more than half. When the full index was used, runtime was improved even more to 93s for setting ($\sigma = 10, \gamma = 0, \lambda = 10$) compared to 124s when the min-max index is used and 283s when no index is used. We also show the total runtimes, including reduce time, for two different settings of ($\sigma = 10, \gamma = 0, \lambda = 5$) and ($\sigma = 10, \gamma = 1, \lambda = 5$) in Figures 7(b) and 7(c). In the easier setting where $\gamma = 0$, the effect of using an index is large since map time corresponds to a large portion of the total time, which is reduced from 366s to 157s when the full index is used. Setting γ to 1 increases the reduce time, that is, mining takes longer. The total runtime is reduced from 907s (with no index) to 638s (with full index).



(a) NYT-doc (map time only)



(b) NYT-doc ($\sigma = 10, \gamma = 0, \lambda = 5$)



(c) NYT-doc ($\sigma = 10, \gamma = 1, \lambda = 5$)

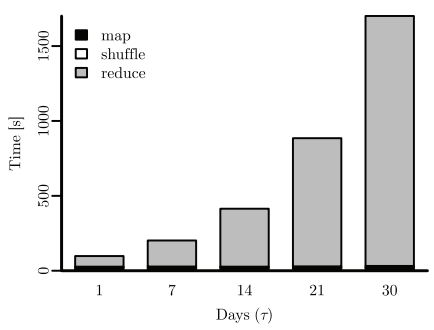
Fig. 7. Effectiveness of indexing for long sequences.

Our experiments also show that different length distributions of input sequences affect both map and reduce times. Recall that NYT-sen and NYT-doc contain the same data, but differ in sequence length (and number of sequences). The runtimes for $\sigma = 10$, $\gamma = 1$, and $\lambda = 5$ for NYT-set and NYT-doc are shown in Figures 5(a) and 7(c), respectively. When no indexing was used, the overall runtime of the map phase was more than $2\times$ larger on NYT-doc than on NYT-sen; that is, longer sequences translate to larger map times. With the full index, however, the map times of NYT-sen and NYT-doc were almost identical (108s versus 102s, respectively). Note that the total mining time (i.e., including the reduce phase) is not comparable because outputs are different.

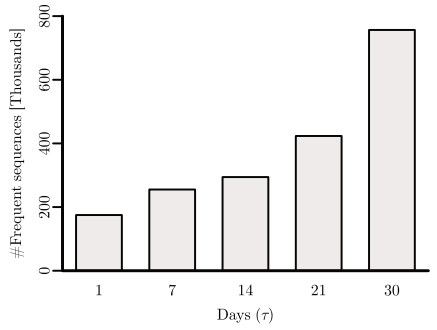
We also studied whether the use of indexing improves performance when the input sequences are short. Using the NYT-sen dataset with short sequences (where each sentence corresponds to a different input sequence and the average length is 19), we observed that the construction and maintenance of the index were slower than just scanning repeatedly the input sequences. When sequences are short, we need only perform a few, cheap scans so that index construction is not beneficial.

7.8. Temporal sequences

We now evaluate our approach for mining temporal sequences. Using the Netflix dataset [Bennett and Lanning 2007], we extracted temporal sequences of movies capturing the order in which these movies were rated by users. Mining frequent sequences in this context yields sequences of movies reflecting the chronological order in which a user viewed or rated them.



(a) Netflix (runtimes)



(b) Netflix (sequences)

Fig. 8. Temporal sequences.

Table III. Example Frequent Sequences from Netflix ($\sigma = 1000, \lambda = 5, \tau = 1\ day$)

Sequence of movie titles (frequency)
(1) *"Men in Black II", "Independence Day", "I, Robot"* (2,268)
(2) *"Pulp Fiction","Fight Club"* (7,406)
(3) *"Lord of the Rings: The Fellowship of the Ring", "Lord of the Rings: The Two Towers"* (19,303)
(4) *"The Patriot", "Men of Honor"* (28,710)
(5) *"Con Air", "The Rock"* (29,749)
(6) *"Pretty Woman", "Miss Congeniality"* (30,036)

We mined frequent sequences from this dataset for $\sigma = 1000, \lambda = 5$, and temporal gaps of 1, 7, 14, 21, and 30 days. The results are shown in Figures 8(a) and 8(b). Figure 8(a) depicts the runtimes as we increase the temporal gap (τ) from 1 day (which corresponds to $\gamma = 297$) to 30 days (corresponding to $\gamma = 6068$). Figure 8(b) shows the total size of the result, that is, how many frequent sequences were mined. Frequent sequences of user rating events within a 1-day timespan were mined in 98s and were 175,003 in total. When the temporal gap was increased to 30 days (1-month timespan), we mined 756,528 frequent sequences (a 4.32× increase) while total runtime had a significant 17× increase, again due to the large number of candidate 2-sequences constructed by the mining algorithm.

Table III includes some example sequences of movies mined from the Netflix dataset. We can see that this includes movies from a trilogy in chronological order (see sequence (3), which consists of movies from the *Lord of the Rings* trilogy) and movies with the same actor (see sequence (1), which consists of movies starring actor Will Smith).

8. RELATED WORK

We now relate the ideas put forward in this article to existing prior work. Prior approaches can be coarsely categorized with respect to the type of pattern being mined (frequent itemsets, frequent sequences, or application-specific special cases such as n-grams) and according to their parallelization (sequential, shared-memory parallel, shared-nothing parallel, or MapReduce).

Sequential approaches to frequent itemset mining fall into two families. Candidate generation and pruning methods such as Apriori [Agrawal et al. 1993] repeatedly scan the input data to count and prune candidate itemsets of increasing cardinality. Pattern growth approaches, in contrast, scan the input data only once and construct a compact representation of it. FP-growth [Han et al. 2004], as one such method, canonicalizes transactions by ordering items therein according to their support and represents the

input data as a compact augmented prefix tree. Frequent itemsets can then be determined efficiently by traversing this so-called FP-tree. Our approach uses a similar item-based partitioning of the output space as FP-growth. Moreover, to cope with input data that exceeds the available main memory, FP-growth works with small projected databases which contain all items required for one of the output partitions. Adaptations of these ideas have also been used for frequent sequence mining. GSP [Srikant and Agrawal 1996] generates candidate k-sequences by joining frequent $(k-1)$-sequences and prunes them by means of scanning the input data. FreeSpan [Han et al. 2000] can be seen as an early adaptation of FP-growth to mine frequent sequences. PrefixSpan [Pei et al. 2004], its successor, uses a pattern growth approach based on database projections, but employs a suffix-based partitioning of the output space. SPADE [Zaki 2001b] assumes an alternative vertical representation of the input data which can be understood as an inverted index that maintains, for each item, the list of transactions containing the item, and traverses the (conceptual) lattice of all sequences in breadth-first or depth-first order. Frequent episode mining [Mannila et al. 1997], a related but different problem, determines sequences that occur frequently within a single transaction. Zaki [2000] and Giannotti et al. [2006] considered variations of the temporal sequence mining problem discussed in this article.

To reduce the size of the frequent sequences, many studies have focused on mining closed sequences since they concisely represent the set of all frequent sequences. Adapting pattern growth approaches like PrefixSpan, Yan et al. [2003] describe the CloSpan algorithm to mine closed sequences. It maintains the set of already mined closed sequence candidates which are used to prune the search space and checks whether a newly found sequence is a candidate closed sequence. This method requires postprocessing to prune non-closed sequences. A potential limitation of CloSpan is that it requires to maintain the set of all closed sequence candidates in memory. To this end, Wang and Han [2004] proposed the BIDE algorithm which does not require to keep a candidate set of closed sequences. Instead, it uses a bi-directional closure checking scheme to determine whether a newly generated candidate sequence is closed. Li and Wang [2008] extend the framework of BIDE to mine closed sequences with gap constraints. Differing from pattern growth approaches, the ClaSP algorithm [Gomariz et al. 2013] exploits the vertical database format of SPADE to discover closed sequences. It traverses the lattice of all sequences in a depth-first order and uses ideas from CloSpan to generate candidate closed sequences and to prune non-closed sequences. Ideas for mining closed sequences can well be carried over to mine maximal sequences. In this direction, Fournier-Viger et al. [2013] describe the MaxSP algorithm which is based on BIDE to mine maximal sequences. A later algorithm called VMSP [Fournier-Viger et al. 2014], which is along the lines of ClaSP, mines maximal sequential patterns using the vertical database format. Luo and Chung [2005] describe the MSPX method which uses database samples for mining maximal sequential patterns. However, MSPX is an approximate algorithm and thus may not mine the complete set of maximal patterns. We refer to Han et al. [2007] for a more detailed discussion of sequential approaches to frequent pattern mining.

Parallel approaches to frequent itemset and sequence mining have been proposed for different machine models. For frequent itemset mining in parallel shared-memory architectures, Parthasarathy et al. [2001] describe an approach that aims for access locality when generating candidates in parallel. Zaki [2001a] investigates how SPADE can be parallelized by distributing data and/or work among machines. Buehrer et al. [2007] target parallel distributed-memory architectures and exploit the item-based partitioning of FP-growth to have different machines operate on partial aggressively pruned copies of the global FP-tree. Guralnik and Karypis [2004] examine how the projection-based pattern growth approach from Agarwal et al. [2001], which is

similar to PrefixSpan, can be parallelized by distributing data and/or work among machines. For the special case of closed sequences, Cong et al. [2005] describe a parallel distributed-memory variant of BIDE. They partition the sequence database based on frequent 1-sequences; a partition contains all suffix projections of sequences that contain the corresponding frequent 1-sequence. Only little work has targeted MapReduce as a model of computation. Li et al. [2008] describe PFP, another adaptation of FP-growth using item-based partitioning that is focused on finding the k most frequent itemsets. Recently, Lin et al. [2014] also exploited MapReduce for frequent subgraph mining using a partitioning approach.

Given the important role of n-grams in natural language processing and information retrieval, it is not surprising that several solutions exist for this specific special case of frequent sequence mining. SRILM [Stolcke 2002] is one of the best-known toolkits to compute and work with n-gram statistics for document collections of modest size. Brants et al. [2007] describe how large-scale statistical language models are trained at Google. To compute counts of n-grams having length five or less, they use a simple extension of WordCount in MapReduce (along the lines of the naïve approach of Section 2.3). Huston et al. [2011] develop distributed methods to build an inverted index for n-grams that occur more than once in the document collection. Most recently, Berberich and Bedathur [2013] described Suffix-σ, to which we compared in our experiments. The algorithm operates on suffixes, akin to Cong et al. [2005], and runs in a single MapReduce job.

To the best of our knowledge, none of the existing work provides a satisfactory solution to general frequent sequence mining in MapReduce. Perhaps closest to our work is that on parallel itemset mining by Buehrer et al. [2007] and Guralnik and Karypis [2004], which also makes use of item-based partitioning of the output space. In contrast to MG-FSM, these methods use database projections tailored to itemset mining; these projections cannot be used for frequent sequence mining and are generally less flexible than our partition construction techniques. Moreover, previous approaches have been evaluated on small-scale and/or synthetic datasets, whereas we use databases with more than 1B real-world input sequences in our experimental study.

Finally, this article is an extended version by Miliaraki et al. [2013], which proposes the basic MG-FSM algorithm. We extend this basic algorithm in multiple ways: (1) We propose novel methods for mining maximal and closed sequences, (2) we propose indexing techniques to efficiently mine sequence databases with long input sequences, and (3) we show how MG-FSM can be used to mine temporal sequences. All of our extensions have been studied in our experimental evaluation.

9. CONCLUSIONS

We proposed MG-FSM, a scalable algorithm for gap-constrained frequent sequence mining in MapReduce. MG-FSM partitions the input database into a set of partitions that can be mined efficiently, independently, and in parallel. Our partitioning is based on a novel notion of w-equivalency, which generalizes the concept of a "projected database" used in many frequent pattern mining algorithms. Scalability is obtained due to a number of novel optimization techniques, including unreachability reduction, prefix/suffix reduction, blank reduction, blank separation, aggregation, and lightweight compression. We also discussed how MG-FSM can handle long sequences, temporal gap constraints, and mining of only maximal or closed sequences. Our experiments suggest that MG-FSM is orders of magnitude more efficient and scalable than baseline algorithms for gap-constrained frequent sequence mining, and competitive to state-of-the-art algorithms for distributed n-gram mining (an important instance of gap-constrained FSM). For example, in our experiments, MG-FSM mined more than 1 billion input sequences for n-grams in less than half an hour on 10 worker machines.

There are a number of directions for extending the MG-FSM framework described in this article.[11] MG-FSM is a generic algorithm and uses a general-purpose sequence miner in the local mining phase. Using a customized miner which is aware of the MG-FSM framework may lead to higher overall efficiency. Perhaps more importantly, MG-FSM focuses on sequences of items, but some applications involve sequences of itemsets. It is not clear how these can be mined in a scalable fashion. Similarly, MG-FSM does not support mining of hierarchical patterns, sequences with attributes, or other pattern types such as frequent episodes or frequent partial orders, all of which are of interest to applications. Finally, integrating different notions of frequency (e.g., collection frequency), supporting further constraints (e.g., generalize at most 3 items), scoring the produced patterns by interestingness, and computing summary statistics about occurrences of frequent sequences and their gaps are promising directions for extending MG-FSM.

APPENDIXES

A. PROOF OF LEMMA 4.4 (UNREACHABILITY REDUCTION)

LEMMA 4.4 (UNREACHABILITY REDUCTION). *Let $T = s_1 \cdots s_l$ and denote by I the set of all (w, γ, λ)-unreachable indexes in T. Then*

$$G_{w,\gamma,\lambda}(T) = G_{w,\gamma,\lambda}(T_{-I}),$$

where T_{-I} is obtained by removing the items at indexes I from T.

We start by relating sequences of indexes of an input sequence T to the pivot sequences generated by T.

Definition A.1 (*Reachable Index Sequence*). Let $T = s_1 \cdots s_l$. We say that a sequence $I = i_1 i_2 \cdots i_n$ of increasing indexes is a (w, γ, λ)-*reachable index sequence* for T if $i_{k+1} - i_k \leq \gamma + 1$ for $1 \leq k < n$, $s_{i_k} \leq w$ for $1 < k < n$, $s_{i_k} = w$ for at least one $1 \leq k \leq n$, and $2 \leq n \leq \lambda$. Denote by $T(I) = s_{i_1} \cdots s_{i_n}$ the sequence generated by I, by $I_{w,\gamma,\lambda}(T)$ the set of all (w, γ, λ)-reachable index sequences for T, and by $I_{w,\gamma,\lambda}(S \mid T) \subseteq I_{w,\gamma,\lambda}(T)$ the set of (w, γ, λ)-reachable index sequences for T that generate S.

This definition matches the definition of γ-subsequences of length at most λ but allows the first and last item to be irrelevant. In fact, $S \in G_{w,\gamma,\lambda}(T)$ if and only if both $S \leq w$ and $I_{w,\gamma,\lambda}(S \mid T)$ is nonempty. Consider, for example, the input sequence $T = aadc$. We have $G_{c,2,3}(T) = \{ac, aac\}$, $I_{c,2,3}(T) = \{1 \cdot 4,\ 2 \cdot 4,\ 3 \cdot 4,\ 1 \cdot 2 \cdot 4\}$, $I_{c,2,3}(ac \mid T) = \{1 \cdot 4, 2 \cdot 4\}$ and $I_{c,2,3}(aac \mid T) = \{1 \cdot 2 \cdot 4\}$.

We make use of a generalized distance definition in our proof of the correctness of the unreachability reduction.

Definition A.2 (*Distance*). Let $T = s_1 \cdots s_{|S|}$. Let $1 \leq i, j \leq |S|$ and set $l = \min\{i, j\}$ and $r = \max\{i, j\}$. The (w, γ, λ)-*distance* $d_{w,\gamma,\lambda}(i, j \mid T)$ between i and j is given by

$$\min\left\{ |I| : I = i_1 \cdots i_n \in I_{w,\gamma,\lambda}(T), i_1 = l, i_n = r \right\} \cup \{\infty\}.$$

Note that $d_{w,\gamma,\lambda}(i, j \mid T) \in \{1, 2, \ldots, \lambda, \infty\}$. Intuitively, the distance is equivalent to the smallest number of items that we need to "step onto" when moving from i to j via relevant items, by skipping at most γ items in each step, and by stepping onto at least one pivot item; it is infinite if there is no such path of length at most λ. We can now define left and right distances formally.

[11] In recent work, Beedkar and Gemulla [2015] address some of the extensions mentioned here. In particular, their LASH system extends MG-FSM with support for hierarchical patterns and makes use of an efficient customized local sequence miner.

Definition A.3 (*Left Distance*). Let $T = s_1 \cdots s_l$. Fix some $1 \leq i \leq l$ and denote by $i_{prev} < i$ the largest index such that $s_{i_{prev}} = w$, if such an index exists. The (w, γ, λ)-*left distance* of index i is defined as $l_{w,\gamma,\lambda}(i \mid T) = d_{w,\gamma,\lambda}(i_{\text{prev}}, i \mid T)$ if i_{prev} exists; otherwise $l_{w,\gamma,\lambda}(i \mid T) = \infty$.

We define the (w, γ, λ)-*right distance* $r_{w,\gamma,\lambda}(i \mid T)$ similarly with respect to the closest pivot to the right of index i. The following lemma captures most of the proof of the correctness of unreachability reduction.

LEMMA A.4. *Let $T = s_1 \cdots s_l$ and let $1 \leq k \leq l$ be a (w, γ, λ)-unreachable index. Then*

$$G_{w,\gamma,\lambda}(T) = G_{w,\gamma,\lambda}(T'),$$

where T' is obtained by removing index k from T. Moreover, for $1 \leq i \leq j \leq l$, $i \neq k$, $j \neq k$, we have

$$d_{w,\gamma,\lambda}(i, j \mid T) = d_{w,\gamma,\lambda}(i', j' \mid T'), \tag{8}$$

where $i' = i$ if $i < k$ and $i' = i - 1$ if $i > k$ (similarly j').

The first part of the lemma states that we can safely remove a single unreachable item from T. The second part states that all distances between remaining indexes are unaffected. Thus, if an index i in T is unreachable in T, the corresponding index i' in T' will also be unreachable. We can thus remove all unreachable items in one go, which proves Lemma 4.4.

PROOF. For brevity, we drop subscript (w, γ, λ) from our notation.

First, observe that if k is unreachable, we have for all indexes $i_- < k$ and $i_+ > k$, $d(i^-, k \mid T) = \infty$ and $d(k, i^+ \mid T) = \infty$, which implies $d(i_-, i_+ \mid T) = \infty$. Our definition of distance thus implies that there is no reachable index sequence of T that "crosses" k, that is, simultaneously contains indexes less than k and indexes larger than k. Now pick any sequence $S \in G(T)$ and any of its index sequences $I \in I(S \mid T)$. If I consists only of indexes smaller than k, then $T'(I) = T(I) = S$. Otherwise, I consists only of indexes larger than k. Then $T'(I') = T(I) = S$, where I' is obtained from I by decrementing every index by one. Thus $S \in G(T)$ implies $S \in G(T')$.

We now show that no additional sequences are generated from T'. Suppose to the contrary that there exists a sequence S of length at most λ that is generated from T' but not from T. Then $I(S \mid T) = \emptyset$ but $I(S \mid T') \neq \emptyset$. Pick any $I' \in I(S \mid T')$, and denote by i'_- and i'_+ the smallest and largest index in I', respectively. We must have $i'_- < k$ and $i'_+ \geq k$; otherwise the preceding arguments imply that T would have also generated S. Since $I' \in I(S \mid T')$, we obtain $d(i'_-, i'_+ \mid T') \leq \lambda$. Assume for now that Eq. (8) asserted earlier indeed holds. Then $d(i'_-, i'_+ \mid T') = d(i_-, i_+ \mid T)$, where $i_- = i'_-$ and $i_+ = i'_+ + 1$ denote the corresponding indexes in T. Since we showed before that $d(i_-, i_+ \mid T) = \infty$, we conclude that $d(i'_-, i'_+ \mid T') = \infty > \lambda$, a contradiction. Thus $S \in G(T')$ implies $S \in G(T)$.

It remains to show that Eq. (8) holds. Observe that the distance between two indexes i and j depends only on the set $\{ v : i \leq v \leq j \}$ of in-between indexes and, in particular, depends on s_v only through the *properties* of s_v, that is, whether s_v is relevant and whether it is a pivot. Pick any $i_- \neq k$ and $i_+ > i_-$, $i_+ \neq k$, and denote by i'_- and i'_+ the corresponding indexes in T'. We need to show that

$$d(i_-, i_+ \mid T) = d(i'_-, i'_+ \mid T'). \tag{9}$$

If $i_+ < k$ or $i_- > k$, $T(i_- \cdots i_+) = T'(i'_- \cdots i'_+)$ and Eq. (9) follows immediately. Otherwise, we have $i^- < k$ and $i^+ > k$ and $d(i^-, i^+ \mid T) = \infty$. First, $d(i'_-, k \mid T') = \infty$ since: (1) $T(i_- \cdots [k-1]) = T'(i'_- \cdots [k-1])$; (2) any difference between $d(i_-, k \mid T)$ and $d(i'_-, k \mid T')$ thus depends on the k-th item in each input sequence; (3) the k-th item

is neither a pivot in T nor in T' (otherwise index k would be reachable in T) so that $d(i'_-, k \mid T') = d(i_-, k \mid T)$; and (4) $d(i_-, k \mid T) = \infty$. Using similar arguments, we can show that $d(k, i'_+ \mid T') = \infty$ and therefore $d(i'_-, i'_+ \mid T') = \infty$ as desired. □

B. PROOF OF LEMMA 4.5 (PREFIX/SUFFIX REDUCTION)

LEMMA 4.5 (PREFIX/SUFFIX REDUCTION).

$$G_{w,\gamma,\lambda}(\llcorner^{l_1} T \llcorner^{l_2}) = G_{w,\gamma,\lambda}(T).$$

PROOF. We first show that $S \in G_{w,\gamma,\lambda}(T)$ implies $S \in G_{w,\gamma,\lambda}(T')$. Let $S \in G_{w,\gamma,\lambda}(T)$ and pick any $I \in I_{w,\gamma,\lambda}(S \mid T)$, where we continue to use the notation from Appendix A. Denote by i^- and i^+ the smallest and largest indexes in I. We have $i^- > l_1$ and $i^+ < |T| - l_2$, since all items in prefix $\llcorner^{l_1}$ or suffix $\llcorner^{l_2}$ are w-irrelevant and thus not part of a pivot sequence. Denote by I' the index set obtained from I by decrementing all indexes by l_1. Then I' is a (w, γ, λ)-reachable index sequence for T' with $T'(I') = T(I) = S$ so that $S \in G_{w,\gamma,\lambda}(T')$, as claimed. Using similar arguments, one can show that $S \in G_{w,\gamma,\lambda}(T')$ implies $S \in G_{w,\gamma,\lambda}(T)$, completing the proof. □

C. PROOF OF LEMMA 5.1 (SUPPORT MONOTONICITY)

LEMMA 5.1 (SUPPORT MONOTONICITY). *Let S and S' be two sequences such that $S \subseteq_{\gamma^-} S'$. For all sequence databases $\mathscr{D}$, we have*

$$\mathrm{Sup}_\gamma(S, \mathscr{D}) \supseteq \mathrm{Sup}_\gamma(S', \mathscr{D}).$$

PROOF. Consider any input sequence $T = t_1 \dots t_n \in \mathscr{D}$ such that $S' \subseteq_\gamma T$. We need to show that this implies $S \subseteq_\gamma T$. Since $S' \subseteq_\gamma T$, there is a set of indexes $i'_1 < \dots < i'_{|S'|}$ such that: (i) $S_k = T_{i'_k}$ for $1 \le k \le |S'|$, and (ii) $i'_{k+1} - i'_k - 1 \le \gamma$ for $1 \le k < |S'|$. Furthermore, since $S \subseteq_{\gamma^-} S'$, there is a set of indexes $j_1 < \dots < j_{|S|}$ such that: (i) $S_k = S'_{j_k}$ for $1 \le k \le |S|$, and (ii) $j_{k+1} - j_k - 1 \le \gamma^-$ for $1 \le k < |S|$. Now consider the set of indexes $i_k = i'_{j_k}$. We have $S = T_{i_1} \cdots T_{i_{|S|}}$ by construction so that S is a ∞-subsequence of T; this proves the lemma for $\gamma = \infty$. To finish the proof, observe that when $\gamma < \infty$, we have $\gamma^- = 0$ so that $j_{k+1} = j_k + 1$ and therefore $i_{k+1} - i_k - 1 = i'_{j_{k+1}} - i'_{j_k} - 1 = i'_{j_k+1} - i'_{j_k} - 1 \le \gamma$. □

D. PROOF OF LEMMA 5.5

LEMMA 5.5. *Every globally maximal sequence S with $|S| \ge 2$ is also locally maximal.*

PROOF. Let S with $|S| \ge 2$ be globally maximal and set $w = p(S)$. We have $f(S, \mathscr{D}) \ge \sigma$ and, since $\mathscr{D}$ and $\mathscr{P}_w$ are w-equivalent, $S \in F_w(\mathscr{P}_w)$. We have to show that $S \in F_w^{\max}(\mathscr{P}_w)$ as well. Suppose to the contrary that $S \notin F_w^{\max}(\mathscr{P}_w)$. By the definition of local maximality, there must be a sequence $S' \supset_{\gamma^-} S$ with $S' \in F_w(\mathscr{P}_w)$. Since $S' \in F_w(\mathscr{P}_w)$, we have $p(S') = w$, $|S'| \ge 2$ and $f(S'; \mathscr{P}_w) \ge \sigma$. Since furthermore $\mathscr{D}$ and $\mathscr{P}_w$ are w-equivalent, it follows that $f(S'; \mathscr{P}_w) = f(S'; \mathscr{D})$ and thus $f(S'; \mathscr{D}) \ge \sigma$. But then $S' \in F_{\sigma,\gamma,\lambda}(\mathscr{D})$ so that S cannot be globally maximal, a contradiction. □

E. PROOF OF LEMMA 5.6

LEMMA 5.6. *Let S be a spurious sequence. Then there is a* primary-witness sequence *S^+ which satisfies:*

(1) $S \subset_{\gamma^-} S^+$, $p(S^+) > p(S)$, and S^+ is frequent,
(2) S^+ is locally maximal,
(3) there is no intermediate sequence S^* with $p(S^*) < p(S^+)$ and $S \subset_{\gamma^-} S^* \subset_{\gamma^-} S^+$.

PROOF. Since S is spurious, there must be some globally maximal sequence $S' \supset_{\gamma^-} S$ satisfying (1); Lemma 5.5 implies that S' also satisfies (2). If S' satisfies (3), we take $S^+ = S'$ and are done.

Otherwise, let $w = p(S)$. Pick any sequence S' that satisfies (1) and (2); the prior discussion shows that there is such a sequence. Suppose that S' contains only one distinct item w' that is larger than w. We show that S' then satisfies (3). Suppose to the contrary that S' contains a subsequence S^* satisfying $S \subset_{\gamma^-} S^* \subset_{\gamma^-} S'$ and $w^* = p(S^*) < p(S') = w'$. Since S' contains only one distinct item that is "larger" than w, we must have $w^* = w$. Since S' is frequent and by Lemma 5.1 any γ^--subsequence of a frequent sequence is itself frequent, we conclude that S^* is frequent. Putting both together, we have $S^* \in F_w(\mathscr{P}_w)$. But then S is not locally maximal and thus not a spurious sequence, a contradiction.

Now assume that S' satisfies (1) and (2) but not (3). We show that there must be a "smaller" sequence S^- such that $p(S) < p(S^-) < p(S')$ and S^- satisfies (1) and (2). If S^- also satisfies (3), we are done. If not, we iterate this process by taking the just-obtained sequence S^- for S'. Since after every iteration $p(S) < p(S^-) < p(S')$, S^- will eventually satisfy (3); by the preceding discussion, this happens at the latest when S^- contains only one item larger than w. The lemma thus follows.

It remains to show that S^- exists. Let $w' = p(S')$ and let S^* be any subsequence of S' violating (3), that is, $w^* = p(S^*) < w'$ and $S \subset_{\gamma^-} S^* \subset_{\gamma^-} S'$. Using arguments as earlier, we find that S^* satisfies (1). If S^* also satisfies (2), set $S^- = S^*$. Otherwise, S^* is not locally maximal. But then there is a locally maximal sequence $S_2^* \in F_{w^*}(\mathscr{P}_{w^*})$ with $S^* \subset_{\gamma^-} S_2^*$. Clearly, $S \subset_{\gamma^-} S_2^*$ as well. Since additionally $p(S_2^*) = w^* > w$, we conclude that S_2^* satisfies (1) and (2), and set $S^- = S_2^*$. □

REFERENCES

Ramesh C. Agarwal, Charu C. Aggarwal, and V. V. V. Prasad. 2001. A tree projection algorithm for generation of frequent item sets. *J. Parallel Distrib. Comput.* 61, 3, 350–371.

Rakesh Agrawal, Tomasz Imielinski, and Arun N. Swami. 1993. Mining association rules between sets of items in large databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'93).* 207–216.

Kaustubh Beedkar and Rainer Gemulla. 2015. LASH: Large-scale sequence mining with hierarchies. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'15).* To appear.

James Bennett and Stan Lanning. 2007. The Netflix prize. In *Proceedings of the KDD Cup and Workshop (KDD'07).*

Klaus Berberich and Srikanta Bedathur. 2013. Computing n-gram statistics in MapReduce. In *Proceedings of the International Conference on Extending Database Technology (EDBT'13).* 101–112.

Thorsten Brants and Alex Franz. 2006. Web 1T 5-gram version 1. Linguistic Data Consortium, Philadelphia. https://catalog.ldc.upenn.edu/LDC2006T13.

Thorsten Brants, Ashok C. Popat, Peng Xu, Franz J. Och, Jeffrey Dean, and Google Inc. 2007. Large language models in machine translation. In *Proceedings of the Conference on Empirical Methods on Natural Language Processing (EMNLP'07).* 858–867.

Gregory Buehrer, Srinivasan Parthasarathy, Shirish Tatikonda, Tahsin Kurc, and Joel Saltz. 2007. Toward terabyte pattern mining: An architecture-conscious solution. In *Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'07).* 2–12.

ClueWeb. 2009. ClueWeb09 dataset. http://lemurproject.org/clueweb09/.

Shengnan Cong, Jiawei Han, and David Padua. 2005. Parallel mining of closed sequential patterns. In *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD'05).* 562–567.

Jeffrey Dean and Sanjay Ghemawat. 2004. MapReduce: Simplified data processing on large clusters. In *Proceedings of the 6th Symposium on Operating Systems Design and Implementation (OSDI'04).* 137–150.

Philippe Fournier-Viger, Cheng-Wei Wu, Antonio Gomariz, and Vincent S. Tseng. 2014. VMSP: Efficient vertical mining of maximal sequential patterns. In *Proceedings of the 27th Canadian Conference on Artificial Intelligence (AI'14)*. 83–94.

Philippe Fournier-Viger, Cheng-Wei Wu, and Vincent S. Tseng. 2013. Mining maximal sequential patterns without candidate maintenance. In *Proceedings of the 9th International Conference on Advanced Data Mining and Applications (ADMA'13)*. 169–180.

Fosca Giannotti, Mirco Nanni, and Dino Pedreschi. 2006. Efficient mining of temporally annotated sequences. In *Proceedings of the SIAM International Conference on Data Mining (SDM'06)*. 346–357.

Antonio Gomariz, Manuel Campos, Roque Marin, and Bart Goethals. 2013. ClaSP: An efficient algorithm for mining frequent closed sequences. In *Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'13)*. 50–61.

Valerie Guralnik and George Karypis. 2004. Parallel tree-projection-based sequence mining algorithms. *Parallel Comput.* 30, 4, 443–472.

Jiawei Han, Hong Cheng, Dong Xin, and Xifeng Yan. 2007. Frequent pattern mining: Current status and future directions. *Data Mining Knowl. Discov.* 15, 1, 55–86.

Jiawei Han, Jian Pei, Behzad Mortazavi-Asl, Qiming Chen, Umeshwar Dayal, and Mei-Chun Hsu. 2000. FreeSpan: Frequent pattern-projected sequential pattern mining. In *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD'00)*. 355–359.

Jiawei Han, Jian Pei, Yiwen Yin, and Runying Mao. 2004. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data Mining Knowl. Discov.* 8, 1, 53–87.

Samuel Huston, Alistair Moffat, and W. Bruce Croft. 2011. Efficient indexing of repeated n-grams. In *Proceedings of the ACM International Conference on Web Search and Data Mining (WSDM'11)*. 127–136.

Ravi Kant, Srinivasan H. Sengamedu, and Krishnan S. Kumar. 2012. Comment spam detection by sequence mining. In *Proceedings of the ACM International Conference on Web Search and Data Mining (WSDM'12)*. 183–192.

Christian Kohlschutter, Peter Fankhauser, and Wolfgang Nejdl. 2010. Boilerplate detection using shallow text features. In *Proceedings of the ACM International Conference on Web Search and Data Mining (WSDM'10)*. 441–450.

Chun Li and Jianyong Wang. 2008. Efficiently mining closed subsequences with gap constraints. In *Proceedings of the SIAM International Conference on Data Mining (SDM'08)*. 313–322.

Haoyuan Li, Yi Wang, Dong Zhang, Ming Zhang, and Edward Y. Chang. 2008. PFP: Parallel FP-growth for query recommendation. In *Proceedings of the ACM Conference on Recommender Systems (RecSys'08)*. 107–114.

Wenqing Lin, Xiaokui Xiao, and Gabriel Ghinita. 2014. Large-scale frequent subgraph mining in MapReduce. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE'14)*. 844–855.

Adam Lopez. 2008. Statistical machine translation. *ACM Comput. Surv.* 40, 3.

Congnan Luo and Soon Myoung Chung. 2005. Efficient mining of maximal sequential patterns using multiple samples. In *Proceedings of the SIAM International Conference on Data Mining (SDM'05)*. 415–426.

Heikki Mannila, Hannu Toivonen, and A. Inkeri Verkamo. 1997. Discovery of frequent episodes in event sequences. *Data Mining Knowl. Discov.* 1, 3, 259–289.

MG-FSM 2014. MG-FSM source code. http://dws.informatik.unimannheim.de/en/resources/software/mg-fsm/.

Iris Miliaraki, Klaus Berberich, Rainer Gemulla, and Spyros Zoupanos. 2013. Mind the gap: Large-scale frequent sequence mining. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'13)*. 797–808.

Ndapandula Nakashole, Martin Theobald, and Gerhard Weikum. 2011. Scalable knowledge harvesting with high precision and high recall. In *Proceedings of the ACM International Conference on Web Search and Data Mining (WSDM'11)*. 227–236.

Srinivasan Parthasarathy, Mohammed J. Zaki, Mitsunori Ogihara, and Wei Li. 2001. Parallel data mining for association rules on shared-memory systems. *Knowl. Inf. Syst.* 3, 1, 1–29.

Jian Pei, Jiawei Han, Behzad Mortazavi-Asl, Jianyong Wang, Helen Pinto, Qiming Chen, Umeshwar Dayal, and Mei-Chun Hsu. 2004. Mining sequential patterns by pattern-growth: The PrefixSpan approach. *IEEE Trans. Knowl. Data Engin.* 16, 1424–1440.

Ramakrishnan Srikant and Rakesh Agrawal. 1996. Mining sequential patterns: Generalizations and performance improvements. In *Proceedings of the International Conference on Extending Database Technology (EDBT'96)*. 3–17.

Jaideep Srivastava, Robert Cooley, Mukund Deshpande, and Pang-Ning Tan. 2000. Web usage mining: Discovery and applications of usage patterns from Web data. *SIGKDD Explor. Newsl.* 1, 2, 12–23.

Andreas Stolcke. 2002. SRILM - An extensible language modeling toolkit. In *Proceedings of the Annual Conference of the International Speech Communication Association (INTERSPEECH'02)*.

Niket Tandon, Gerard de Melo, and Gerhard Weikum. 2011. Deriving a web-scale common sense fact database. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI'11)*.

The New York Times. 2008. The New York Times annotated corpus. https://catalog.ldc.upenn.edu/LDC2008T19.

Jianyong Wang and Jiawei Han. 2004. BIDE: Efficient mining of frequent closed sequences. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE'04)*. 79–90.

Ian H. Witten, Alistair Moffat, and Timothy C. Bell. 1999. *Managing Gigabytes: Compressing and Indexing Documents and Images* 2nd Ed. Morgan Kaufmann.

Xifeng Yan, Jiawei Han, and Ramin Afshar. 2003. CloSpan: Mining closed sequential patterns in large databases. In *Proceedings of the SIAM International Conference on Data Mining (SDM'03)*. 166–177.

Mohammed J. Zaki. 2000. Sequence mining in categorical domains: Incorporating constraints. In *Proceedings of the Conference on Information and Knowledge Management (CIKM'00)*. 422–429.

Mohammed J. Zaki. 2001a. Parallel sequence mining on shared-memory machines. *J. Parallel Distrib. Comput.* 61, 3, 401–426.

Mohammed J. Zaki. 2001b. SPADE: An efficient algorithm for mining frequent sequences. *Mach. Learn.* 42, 1–2, 31–60.

ChengXiang Zhai. 2008. Statistical language models for information retrieval a critical review. *Foundat. Trends Inf. Retr.* 2, 3, 137–213.

Received June 2014; revised January 2015; accepted March 2015