Frequent Sequence Mining in MapReduce

Klaus Berberich
(kberberi@mpi-inf.mpg.de)
Sequence Data

- Sequence data is **abundant** and comes in **different forms**
  - **unstructured data** (e.g., text documents, customer complaints)
    - \{ the artist is the creator of beautiful things \}
  - **structured data** (e.g., user sessions logs, ip traffic logs)
    - \{ index a−z about search product−7 search \}
  - **mixed data** (e.g., user purchase histories with reviews interleaved)
    - \{ ... such an epic masterpiece ... ... a poor adaptation of the book ... ... please stop it here ... \}
Problem Statement

- **Input**: Sequence database \( D \)
- **Output**: Sequences that
  - occur in at least \( \sigma \) input sequences (**support threshold**)
  - have a gap of at most \( \gamma \) between consecutive items (**gap constraint**)
  - consist of at most \( \lambda \) items (**length constraint**)

\[
D = \{ \text{a sunny day in new york} \}
\{ \text{sunny new york weather} \}
\{ \text{it is never sunny in york} \}
\{ \text{new york is not like york} \}
\]
Problem Statement

- **Input**: Sequence database $D$
- **Output**: Sequences that
  - occur in at least $\sigma$ input sequences (support threshold)
  - have a gap of at most $\gamma$ between consecutive items (gap constraint)
  - consist of at most $\lambda$ items (length constraint)

\[ D = \{ \text{a sunny day in new york} \}, \{ \text{sunny new york weather} \}, \{ \text{it is never sunny in york} \}, \{ \text{new york is not like york} \} \]

$\sigma = 3$
$\gamma = 0$
$\lambda = 2$

\[ \{ \text{new} \} : 3, \{ \text{york} \} : 4, \{ \text{sunny} \} : 3, \{ \text{new york} \} : 3 \]
Problem Statement

- **Input**: Sequence database $D$
- **Output**: Sequences that
  - occur in at least $\sigma$ input sequences (support threshold)
  - have a gap of at most $\gamma$ between consecutive items (gap constraint)
  - consist of at most $\lambda$ items (length constraint)

$D$

\[
\langle \text{a sunny day in new york} \rangle \quad \sigma = 3 \\
\langle \text{sunny new york weather} \rangle \\
\langle \text{it is never sunny in york} \rangle \\
\langle \text{new york is not like york} \rangle
\]

\[
\langle \text{new} \rangle : 3 \\
\langle \text{york} \rangle : 4 \\
\langle \text{sunny} \rangle : 3 \\
\langle \text{new york} \rangle : 3 \\
\langle \text{sunny york} \rangle : 3
\]
Applications

- **Application areas** of frequent sequence mining include
  - **Natural Language Processing** (e.g., to estimate language models)
  - **Web Search** (e.g., to get insights into users’ behavior)
  - **Digital Humanities** (e.g., to understand the evolution of language)
  - **Business Intelligence** (e.g., to make better recommendations)
Applications

Get up at eight ‘o clock!

Get a potato clock!
Applications

- **Application areas** of frequent sequence mining include
  - **Natural Language Processing** (e.g., to estimate language models)
  - **Web Search** (e.g., to get insights into users’ behavior)
  - **Digital Humanities** (e.g., to understand the evolution of language)
  - **Business Intelligence** (e.g., to make better recommendations)
Applications

- Natural Language Processing (e.g., to estimate language models)
- Web Search (e.g., to get insights into users' behavior)
- Digital Humanities (e.g., to understand the evolution of language)
- Business Intelligence (e.g., to make better recommendations)
Applications

- **Application areas** of frequent sequence mining include
  - **Natural Language Processing** (e.g., to estimate language models)
  - **Web Search** (e.g., to get insights into users’ behavior)
  - **Digital Humanities** (e.g., to understand the evolution of language)
  - **Business Intelligence** (e.g., to make better recommendations)
Applications

Application areas of frequent sequence mining include:

- **Natural Language Processing** (e.g., to estimate language models)
- **Web Search** (e.g., to get insights into users' behavior)
- **Digital Humanities** (e.g., to understand the evolution of language)
- **Business Intelligence** (e.g., to make better recommendations)
Applications

- **Application areas** of frequent sequence mining include
  - **Natural Language Processing** (e.g., to estimate language models)
  - **Web Search** (e.g., to get insights into users’ behavior)
  - **Digital Humanities** (e.g., to understand the evolution of language)
  - **Business Intelligence** (e.g., to make better recommendations)
Applications

Application areas of frequent sequence mining include:

- **Natural Language Processing** (e.g., to estimate language models)
- **Web Search** (e.g., to get insights into users' behavior)
- **Digital Humanities** (e.g., to understand the evolution of language)
- **Business Intelligence** (e.g., to make better recommendations)
Applications

- **Application areas** of frequent sequence mining include
  - **Natural Language Processing** (e.g., to estimate language models)
  - **Web Search** (e.g., to get insights into users’ behavior)
  - **Digital Humanities** (e.g., to understand the evolution of language)
  - **Business Intelligence** (e.g., to make better recommendations)
MapReduce

- **Distributed data processing platform** developed by Google
  - for clusters of **commodity hardware**
  - transparently handles **hardware/software failures**
  - **open-source implementations** available (e.g., Apache Hadoop)

- **Core infrastructure** for Big Data
  - **commercial distributions** of Apache Hadoop (e.g., Cloudera, MapR)
  - **big bets** by “traditional” IT companies (e.g., IBM, Intel, Microsoft)
  - **widely used** in business world (e.g., ebay, Samsung, Bloomberg)
Challenges

- **Volume** of sequence databases (billions or more)
- Taming the **combinatorial explosion**
  - sequence of length $n$ has $O(n^2)$ contiguous subsequences
  - sequence of length $n$ has $O(2^n)$ non-contiguous subsequences
- Sequences with **small support** can be interesting
- Performance of **MapReduce** algorithm critically depends on
  - number of jobs
  - communication cost
  - main memory consumption
Outline

๏ Motivation

๏ Background
  ◦ Suffix-\(\sigma\): Contiguous Sequences (EDBT ’13)
  ◦ MG-FSM: Non-Contiguous Sequences (SIGMOD ’13)

๏ Experiments

๏ Outlook & Summary
Frequent Sequence Mining

- **Apriori principle** (downward closure) exploited by all methods
  - \( \langle a \rangle \) infrequent \( \Rightarrow \) any supersequence (e.g., \( \langle a \ b \rangle \)) is infrequent

- **GSP** [Srikant and Agrawal 1996]
  - candidate generation & pruning via repeated scans of database

- **SPADE** [Zaki 2001]
  - candidate generation & pruning with index built on database

- **PrefixSpan** [Pei et al. 2004]
  - pattern-growth algorithm
Programming model operates on key-value pairs

- **map( )**: \(<k_1, v_1> \rightarrow \text{list}<k_2, v_2>\)
- **compare( )**: \(k_2 \rightarrow \{-1, 0, +1\}\)
- **partition( )**: \(k_2 \rightarrow [0, m]\)
- **reduce( )**: \(<k_2, \text{list}<v_2>> \rightarrow \text{list}<k_3, v_3>\)
MapReduce

**MAP**

\[ M_1 \]

\[ M_n \]

**SHUFFLE**

**REDUCE**

\[ R_1 \]

\[ R_m \]

map()  \rightarrow  partition()  \rightarrow  reduce()
Frequent Sequence Mining in MapReduce (Klaus Berberich)

MapReduce

MAP

MAP

SHUFFLE

REDUCE

map()  partition()  reduce()

\[ d_1 \]

\[ M_1 \quad (\text{me}, 2) \]

\[ d_6 \]

\[ M_n \quad (\text{me}, 5) \]

\[ 1 \quad 1 \]

\[ m \quad m \]

\[ 1 \quad 1 \]

\[ m \quad m \]
Frequent Sequence Mining in MapReduce (Klaus Berberich)
Frequent Sequence Mining in MapReduce (Klaus Berberich)
Naïve Approach

\[ \sigma = 2 \]
\[ \gamma = 0 \]
\[ \lambda = 5 \]

\begin{align*}
1: \langle a \times b \ b \ a \rangle & \quad \text{map()} & \quad \langle a \rangle : 1 \\
& & \langle a \times \rangle : 1 \\
& & \langle a \times b \ b \ b \ a \rangle : 1 \\
& & \langle a \times b \ b \ b \ a \rangle : 1 \\
& & \langle a \times \rangle : 1 \\
& & \langle a \times \rangle : 1 \\
& \cdots
\end{align*}

\textbf{map}(id, sequence): \\
\textbf{for} every \((\gamma, \lambda)\)-subsequence of sequence: \\
\textbf{emit}(subsequence, id)
Naïve Approach

\begin{align*}
\sigma &= 2 \\
\gamma &= 0 \\
\lambda &= 5
\end{align*}

\begin{array}{ll}
\{ ax \} & : <1,1,2> \\
\{ xbx \} & : <1,3> \\
\{ xx \} & : 1 \\
\ldots
\end{array}

\text{reduce()} \rightarrow
\begin{array}{ll}
\{ ax \} & : 2 \\
\{ xbx \} & : 2 \\
\ldots
\end{array}

\text{map}(id, \text{sequence}):
\begin{array}{l}
\text{for every } (\gamma, \lambda)-\text{subsequence of sequence:} \\
\text{emit(subsequence, id)}
\end{array}

\text{reduce}(\text{sequence, list<id>}):
\begin{array}{l}
support = \text{list<id>}.\text{unique()}.\text{count()} \\
\text{if support } \geq \sigma:\ \\
\text{emit(\text{sequence, support})}
\end{array}
Outline

- Motivation
- Background
  - Suffix-\(\sigma\): Contiguous Sequences (EDBT ’13)
  - MG-FSM: Non-Contiguous Sequences (SIGMOD ’13)
- Experiments
- Summary & Outlook
Suffix-\(\sigma\)

- Suffix-\(\sigma\) is based on three key ideas inspired by methods from String Processing (e.g., suffix arrays)
  - emit only sequence suffixes in \(\text{map()}\)
  - partition sequence suffixes based on their first item
  - sort sequence suffixes in reverse lexicographic order

**Reference:** K. Berberich and S. Bedathur: *Computing n-Gram Statistics in MapReduce*, EDBT 2013
Sequence Suffixes

- Suffix-$\sigma$ emits **only sequence suffixes** in `map()`
  - each represents **multiple contiguous sequences** corresponding to its prefixes (e.g., `a x b b a` represents `a`, `a x`, `a x b`, etc.)
  - crucial for reducing **communication cost**

$1 : \{a x b b a\} \rightarrow \text{map()} \rightarrow$

\[
\begin{align*}
\{a x b b a\} & : 1 \\
\{x b b a\} & : 1 \\
\{b b a\} & : 1 \\
\{b a\} & : 1 \\
\{a\} & : 1 
\end{align*}
\]
Partitioning

- **Suffix-σ partitions** sequence suffixes based on their **first item**
  - brings together all suffixes that represent **same contiguous sequence**
  - crucial for computation in a **single MapReduce job**

\[
\begin{align*}
\{ a x b b a \} : 1 \\
\{ x b b a \} : 1 \\
\{ b b a \} : 1 \\
\{ b a \} : 1 \\
\{ a \} : 1 \\
\end{align*}
\]
Sorting & Aggregation

- `compare()`

- `reduce()`

- Suffix-σ sorts sequence suffixes in reverse lexicographic order
  - allows for bookkeeping using a stack of bounded height λ
  - crucial for low main-memory consumption
Sorting & Aggregation

- **compare()**

- **reduce()**

- **Suffix-$\sigma$ sorts** sequence suffixes in **reverse lexicographic order**
  - allows for bookkeeping using a stack of bounded height $\lambda$
  - crucial for **low main-memory consumption**
Sorting & Aggregation

- compare() →

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ a x b }</td>
<td>&lt;2&gt;</td>
</tr>
<tr>
<td>{ a x a }</td>
<td>&lt;1&gt;</td>
</tr>
<tr>
<td>{ a b x }</td>
<td>&lt;3, 4&gt;</td>
</tr>
<tr>
<td>{ a b }</td>
<td>&lt;6, 7&gt;</td>
</tr>
<tr>
<td>{ a }</td>
<td>&lt;1, 5&gt;</td>
</tr>
</tbody>
</table>

- reduce() →

- Suffix-σ sorts sequence suffixes in reverse lexicographic order
  - allows for bookkeeping using a stack of bounded height λ
  - crucial for low main-memory consumption
Sorting & Aggregation

- compare() →

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>{a x b}</td>
<td>2</td>
</tr>
<tr>
<td>{a x a}</td>
<td>1</td>
</tr>
<tr>
<td>{a b x}</td>
<td>3,4</td>
</tr>
<tr>
<td>{a b}</td>
<td>6,7</td>
</tr>
<tr>
<td>{a}</td>
<td>1,5</td>
</tr>
</tbody>
</table>

- reduce() →

- Suffix-σ sorts sequence suffixes in reverse lexicographic order
- allows for bookkeeping using a stack of bounded height $\lambda$
- crucial for low main-memory consumption
Sorting & Aggregation

- **compare()**: 
  - \( \langle a \times b \rangle : <2> \)
  - \( \langle a \times a \rangle : <1> \)
  - \( \langle a \times b \times \rangle : <3, 4> \)
  - \( \langle a \times b \rangle : <6, 7> \)
  - \( \langle a \rangle : <1, 5> \)

- **reduce()**: 
  - \( b \) : \{2\}
  - \( x \) : \{}
  - \( a \) : \{}

- **Suffix-\( \sigma \) sorts** sequence suffixes in **reverse lexicographic order**
  - allows for bookkeeping using a stack of bounded height \( \lambda \)
  - crucial for **low main-memory consumption**
Suffix-\(\sigma\) sorts sequence suffixes in reverse lexicographic order:

- allows for bookkeeping using a stack of bounded height \(\lambda\)
- crucial for low main-memory consumption

\[
\begin{array}{|c|c|}
\hline
\{ \text{a x b} \} & <2> \\
\{ \text{a x a} \} & <1> \\
\{ \text{a b x} \} & <3,4> \\
\{ \text{a b} \} & <6,7> \\
\{ \text{a} \} & <1,5> \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|}
\hline
\text{b} & \{2\} \\
\text{x} & \{\} \\
\text{a} & \{\} \\
\hline
\end{array}
\]
Sorting & Aggregation

- **compare()**

- **reduce()**

- **Suffix-σ sorts** sequence suffixes in **reverse lexicographic order**
  - allows for bookkeeping using a stack of bounded height \( \lambda \)
  - crucial for **low main-memory consumption**

\[
\begin{array}{c}
\langle a \times b \rangle : \langle 2 \rangle \\
\langle a \times a \rangle : \langle 1 \rangle \\
\langle a \ b \ x \rangle : \langle 3, 4 \rangle \\
\langle a \ b \rangle : \langle 6, 7 \rangle \\
\langle a \rangle : \langle 1, 5 \rangle \\
\end{array}
\]

\[
\begin{array}{c}
b \{2\} \\
x \{\} \\
a \{\} \\
\langle a \times b \rangle : 1 \\
\end{array}
\]
Sorting & Aggregation

- compare() ▶️

- reduce() ▶️

- Suffix-σ sorts sequence suffixes in reverse lexicographic order
  - allows for bookkeeping using a stack of bounded height $\lambda$
  - crucial for low main-memory consumption

|     |     | |     |     | |     |     |
|-----|-----| |     |     | |     |     |
| {a x b} : <2> | {a x b} : 1 |
| {a x a} : <1> |
| {a b x} : <3,4> |
| {a b} : <6,7> |
| {a} : <1,5> |

x { }

a { }
Suffix-\( \sigma \) sorts sequence suffixes in reverse lexicographic order
- allows for bookkeeping using a stack of bounded height \( \lambda \)
- crucial for low main-memory consumption
Sorting & Aggregation

- **compare()** →

- **reduce()** →

  - Suffix-σ *sorts* sequence suffixes in **reverse lexicographic order**
  - allows for bookkeeping using a stack of bounded height $\lambda$
  - crucial for **low main-memory consumption**
Sorting & Aggregation

- **compare()**

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>⟨a x b⟩</td>
<td>2</td>
</tr>
<tr>
<td>⟨a x a⟩</td>
<td>1</td>
</tr>
<tr>
<td>⟨a b x⟩</td>
<td>3, 4</td>
</tr>
<tr>
<td>⟨a b⟩</td>
<td>6, 7</td>
</tr>
<tr>
<td>⟨a⟩</td>
<td>1, 5</td>
</tr>
</tbody>
</table>

- **reduce()**

- **Suffix-σ sorts** sequence suffixes in reverse lexicographic order
  - allows for bookkeeping using a stack of bounded height \( \lambda \)
  - crucial for **low main-memory consumption**
Sorting & Aggregation

- compare() →

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ a x b }</td>
<td>&lt;2&gt;</td>
</tr>
<tr>
<td>{ a x a }</td>
<td>&lt;1&gt;</td>
</tr>
<tr>
<td>{ a b x }</td>
<td>&lt;3,4&gt;</td>
</tr>
<tr>
<td>{ a b }</td>
<td>&lt;6,7&gt;</td>
</tr>
<tr>
<td>{ a }</td>
<td>&lt;1,5&gt;</td>
</tr>
</tbody>
</table>

- reduce() →

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ a x b }</td>
<td>1</td>
</tr>
<tr>
<td>{ a x a }</td>
<td>1</td>
</tr>
<tr>
<td>a</td>
<td>{1}</td>
</tr>
<tr>
<td>x</td>
<td>{2}</td>
</tr>
<tr>
<td>a</td>
<td>{}</td>
</tr>
</tbody>
</table>

- **Suffix-σ sorts** sequence suffixes in reverse lexicographic order
  - allows for bookkeeping using a stack of bounded height \( \lambda \)
  - crucial for **low main-memory consumption**
Sorting & Aggregation

- **compare()**
  - \( \langle a \times b \rangle : <2> \)
  - \( \langle a \times a \rangle : <1> \)
  - \( \langle a \times a \rangle : <1> \)
  - \( \langle a \times a \rangle : <1> \)
  - \( \langle a \times b \rangle : <2> \)
  - \( \langle a \times a \rangle : <1> \)
  - \( \langle a \times b \rangle : <2> \)
  - \( \langle a \times a \rangle : <1> \)

- **reduce()**

- **Suffix-\( \sigma \) sorts** sequence suffixes in **reverse lexicographic order**
  - allows for bookkeeping using a stack of bounded height \( \lambda \)
  - crucial for **low main-memory consumption**
Sorting & Aggregation

- **compare()**

  -  \(\{ \text{a x b} \} : <2>\)
  -  \(\{ \text{a x a} \} : <1>\)
  -  \(\{ \text{a b x} \} : <3,4>\)
  -  \(\{ \text{a b} \} : <6,7>\)
  -  \(\{ \text{a} \} : <1,5>\)

- **reduce()**

  -  \(x \{1,2\}\)
  -  \(a \{\}\)

- **Suffix-\(\sigma\) sorts** sequence suffixes in **reverse lexicographic order**
  - allows for bookkeeping using a stack of bounded height \(\lambda\)
  - crucial for **low main-memory consumption**
Sorting & Aggregation

- **compare()**
  - \{ a x b \} : <2>
  - \{ a x a \} : <1>
  - \{ a b x \} : <3,4>
  - \{ a b \} : <6,7>
  - \{ a \} : <1,5>

- **reduce()**
  - \{ a x b \} : 1
  - \{ a x a \} : 1
  - \{ a b \} : 2
  - \{ a \} : ∅

- Suffix-\(\sigma\) sorts sequence suffixes in reverse lexicographic order
  - allows for bookkeeping using a stack of bounded height \(\lambda\)
  - crucial for **low main-memory consumption**
Sorting & Aggregation

- `compare()`: Sorts sequence suffixes in reverse lexicographic order.
  - Suffix-σ sorts sequence suffixes in reverse lexicographic order.
  - Allows for bookkeeping using a stack of bounded height λ.
  - Crucial for low main-memory consumption.

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>a x b</td>
<td>2</td>
</tr>
<tr>
<td>a x a</td>
<td>1</td>
</tr>
<tr>
<td>a b x</td>
<td>3, 4</td>
</tr>
<tr>
<td>a b</td>
<td>6, 7</td>
</tr>
<tr>
<td>a</td>
<td>1, 5</td>
</tr>
</tbody>
</table>

- `reduce()`: Merges sorted sequence suffixes.

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>a x b</td>
<td>1</td>
</tr>
<tr>
<td>a x a</td>
<td>1</td>
</tr>
<tr>
<td>a x</td>
<td>2</td>
</tr>
</tbody>
</table>
Sorting & Aggregation

- compare()

- reduce()

- Suffix-σ sorts sequence suffixes in reverse lexicographic order
  - allows for bookkeeping using a stack of bounded height λ
  - crucial for low main-memory consumption
### Sorting & Aggregation

- **compare()**

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>(&lt;a \times b&gt;)</td>
<td>2</td>
</tr>
<tr>
<td>(&lt;a \times a&gt;)</td>
<td>1</td>
</tr>
<tr>
<td>(&lt;a \times &gt;)</td>
<td>2</td>
</tr>
<tr>
<td>(&lt;a&gt;)</td>
<td>1, 5</td>
</tr>
<tr>
<td>(&lt;a \times b&gt;)</td>
<td>1</td>
</tr>
<tr>
<td>(&lt;a \times a&gt;)</td>
<td>1</td>
</tr>
<tr>
<td>(&lt;a \times&gt;)</td>
<td>2</td>
</tr>
</tbody>
</table>

- **reduce()**

<table>
<thead>
<tr>
<th>Element</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>{}</td>
</tr>
<tr>
<td>a</td>
<td>{1, 2}</td>
</tr>
</tbody>
</table>

- Suffix-\(\sigma\) sorts sequence suffixes in **reverse lexicographic order**
  - allows for bookkeeping using a stack of bounded height \(\lambda\)
  - crucial for **low main-memory consumption**
### Sorting & Aggregation

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ a x b }</td>
<td>&lt;2&gt;</td>
<td></td>
</tr>
<tr>
<td>{ a x a }</td>
<td>&lt;1&gt;</td>
<td></td>
</tr>
<tr>
<td>{ a b x }</td>
<td>&lt;3,4&gt;</td>
<td></td>
</tr>
<tr>
<td>{ a b }</td>
<td>&lt;6,7&gt;</td>
<td></td>
</tr>
<tr>
<td>{ a }</td>
<td>&lt;1,5&gt;</td>
<td></td>
</tr>
</tbody>
</table>

- **compare()**

- **reduce()**

- Suffix-\(\sigma\) sorts sequence suffixes in **reverse lexicographic order**
  - allows for bookkeeping using a stack of bounded height \(\lambda\)
  - crucial for **low main-memory consumption**
Outline

- Motivation
- Background
- **Suffix-σ: Contiguous Sequences (EDBT ’13)**
- **MG-FSM: Non-Contiguous Sequences (SIGMOD ’13)**
- Experiments
- Outlook & Summary
MG-FSM

- MG-FSM is based on three key ideas
  - **item-based non-disjoint partitioning** of input sequences
  - **rewrite** (trim, split, aggregate) sequences in each partition
  - **mine & filter sequences in isolation** on each partition

**Reference:** I. Miliaraki, K. Berberich, R. Gemulla, and S. Zoupananos: *Mind the Gap: Large-Scale Frequent Sequence Mining*, SIGMOD 2013
MG-FSM

MG-FSM is based on three key ideas

- **item-based non-disjoint partitioning** of input sequences
- **rewrite** (trim, split, aggregate) sequences in each partition
- **mine & filter sequences in isolation** on each partition

**Reference:** I. Miliaraki, K. Berberich, R. Gemulla, and S. Zoupanos: *Mind the Gap: Large-Scale Frequent Sequence Mining*, SIGMOD 2013
MG-FSM is based on three key ideas:

- **item-based non-disjoint partitioning** of input sequences
- **rewrite** (trim, split, aggregate) sequences in each partition
- **mine & filter sequences in isolation** on each partition

MG-FSM

- MG-FSM is based on three key ideas
  - item-based non-disjoint partitioning of input sequences
  - rewrite (trim, split, aggregate) sequences in each partition
  - mine & filter sequences in isolation on each partition

Reference: I. Miliaraki, K. Berberich, R. Gemulla, and S. Zoupanos: Mind the Gap: Large-Scale Frequent Sequence Mining, SIGMOD 2013
Partitioning

MG-FSM uses an **item-based non-disjoint partitioning** of input

\[
\sigma = 2 \\
\gamma = 1 \\
\lambda = 2
\]

\[\text{D} = \begin{array}{c}
\{ a \ b \ a \} \\
\{ b \ b \ a \} \\
\{ a \ a \} \\
\{ b \ a \ c \} \\
\{ a \ d \ c \} \\
\{ b \ a \} \\
\end{array}\]

\[
\begin{array}{c}
a : 6 \\
b : 4 \\
c : 2 \\
d : 1 \\
\end{array}\]
MG-FSM uses an **item-based non-disjoint partitioning** of input.

\[
\begin{align*}
\sigma &= 2 \\
\gamma &= 1 \\
\lambda &= 2
\end{align*}
\]

\[
D = \{(a, b, a), (b, b, a), (a, a), (b, a, c), (a, d, c), (b, a)\}
\]

\[
\begin{align*}
a &: 6 \\
b &: 4 \\
c &: 2 \\
d &: 1
\end{align*}
\]
MG-FSM uses an **item-based non-disjoint partitioning** of input.
Partitioning

MG-FSM uses an **item-based non-disjoint partitioning** of input.
MG-FSM uses an **item-based non-disjoint partitioning** of input.

\[ \begin{align*}
\sigma &= 2 \\
\gamma &= 1 \\
\lambda &= 2 
\end{align*} \]

![Diagram showing partitioning with symbols and sequences]

- D
  - \{ a b a \}
  - \{ b b a \}
  - \{ a a \}
  - \{ b a c \}
  - \{ a d c \}
  - \{ b a \}
  - a : 6
  - b : 4
  - c : 2
  - d : 1

- [Partitioned Sequences]
  - \{ a a \}
  - \{ a b \}
  - \{ a c \}
  - \{ b a \}
  - \{ b a \}

- [Partitioned Sequences]
  - \{ a \}
  - \{ b \}
  - \{ c \}
  - \{ a a \}

- [Partitioned Sequences]
  - \{ a \}
  - \{ b \}
  - \{ c \}
  - \{ a a \}

- [Partitioned Sequences]
  - \{ a \}
  - \{ b \}
  - \{ c \}

- [Partitioned Sequences]
  - \{ a \}
  - \{ b \}
  - \{ c \}
  - \{ a a \}
Partitioning

MG-FSM uses an **item-based non-disjoint partitioning** of input.

\[ \sigma = 2 \]
\[ \gamma = 1 \]
\[ \lambda = 2 \]
Partitioning

MG-FSM uses an **item-based non-disjoint partitioning** of input

\[\sigma = 2\]
\[\gamma = 1\]
\[\lambda = 2\]

\(a\) but not \(b, c, \ldots\)
\(b\) but not \(c, d, \ldots\)
\(c\) but not \(d, \ldots\)
MG-FSM uses an item-based non-disjoint partitioning of input

\[ \sigma = 2 \]
\[ \gamma = 1 \]
\[ \lambda = 2 \]

\[ \sigma = 2 \]
\[ \gamma = 1 \]
\[ \lambda = 2 \]

\[ a \]
but not \( b, c, \ldots \)

\[ b \]
but not \( c, d, \ldots \)

\[ c \]
but not \( d, \ldots \)
MG-FSM uses an **item-based non-disjoint partitioning** of input

\[
\sigma = 2 \\
\gamma = 1 \\
\lambda = 2
\]

\[
\begin{align*}
\{ a b a \} & \\
\{ b b a \} & \\
\{ a a \} & \\
\{ b a c \} & \\
\{ a d c \} & \\
\{ b a \} & \\
\{ a \} & \\
\{ b \} & \\
\{ a c \} & \\
\end{align*}
\]

\[
\begin{align*}
\{ a \} & \\
\{ b \} & \\
\{ c \} & \\
\{ d \} & \\
\end{align*}
\]

\[
\begin{align*}
a & \quad \text{but not } b, c, \ldots \\
b & \quad \text{but not } c, d, \ldots \\
c & \quad \text{but not } d, \ldots
\end{align*}
\]
MG-FSM uses an item-based non-disjoint partitioning of input.

\[
\begin{align*}
\sigma &= 2 \\
\gamma &= 1 \\
\lambda &= 2 \\
\end{align*}
\]

\[
\begin{align*}
\{ \text{a b a} \} \\
\{ \text{b b a} \} \\
\{ \text{a a} \} \\
\{ \text{b a c} \} \\
\{ \text{a d c} \} \\
\{ \text{b a} \} \\
\end{align*}
\]

\[
\begin{align*}
\{ \text{a} \} \\
\{ \text{b} \} \\
\{ \text{c} \} \\
\{ \text{a a} \} \\
\{ \text{a b} \} \\
\{ \text{a c} \} \\
\{ \text{b a} \} \\
\end{align*}
\]

\[
\begin{align*}
\text{a} &\quad \text{but not b, c, ...} \\
\text{b} &\quad \text{but not c, d, ...} \\
\text{c} &\quad \text{but not d, ...}
\end{align*}
\]
Rewriting

- Pivot: b

- **Pivot sequence**: Any relevant non-trivial sequence including b

- **Irrelevant items** (i.e., less frequent): c, d, ...

- Rewriting reduces *communication cost* (e.g., by removing items) and makes partition *easier to mine* (e.g., by splitting sequences)

- Any rewriting (trim, split, aggregate) *provably correct*, i.e., we still obtain the same set of pivot sequences and frequencies

\[ \sigma = 2 \]
\[ \gamma = 1 \]
\[ \lambda = 2 \]
Trimming

- Replace **irrelevant items** (i.e., less frequent) by **blank symbol** 
- Replace **unreachable items** (from all pivots) by **blank symbol** 
- Remove **trailing and leading blank symbols**
- Runs of blank symbol coalesced and compressed using **run-length encoding** (e.g., \_ \_ \_ represented as \-2)

\[\begin{align*}
\sigma &= 2 \\
\gamma &= 1 \\
\lambda &= 2
\end{align*}\]
Trimming

Replace irrelevant items (i.e., less frequent) by blank symbol _
Replace unreachable items (from all pivots) by blank symbol _
Remove trailing and leading blank symbols
Runs of blank symbol coalesced and compressed using run-length encoding (e.g., _ _ _ _ represented as 2)
Trimming

- Replace irrelevant items (i.e., less frequent) by blank symbol _
- Replace unreachable items (from all pivots) by blank symbol _
- Remove trailing and leading blank symbols
- Runs of blank symbol coalesced and compressed using run-length encoding (e.g., _ _ _ _ represented as \(-2\))
Trimming

Replace **irrelevant items** (i.e., less frequent) by **blank symbol _**

Replace **unreachable items** (from all pivots) by **blank symbol _**

Remove **trailing and leading blank symbols**

Runs of blank symbol coalesced and compressed using **run-length encoding** (e.g., _ _ _ represented as −2)

\[
\begin{align*}
\sigma &= 2 \\
\gamma &= 1 \\
\lambda &= 2 \\
\end{align*}
\]

\[
\begin{array}{c}
\{ \ b \ } \\
\{ \ a \ b \ b \ a \ } \\
\{ \ a \ b \ b \ a \ _ \ _ \ _ \ b \ } \\
\{ \ b \ a \ _ \ _ \ a \ b \ } \\
\{ \ a \ b \ _ \ _ \ a \ b \ a \ } \\
\end{array}
\]

\[
\text{b but not c, d, ...}
\]
Trimming

- Replace irrelevant items (i.e., less frequent) by blank symbol _
- Replace unreachable items (from all pivots) by blank symbol _
- Remove trailing and leading blank symbols
- Runs of blank symbol coalesced and compressed using run-length encoding (e.g., _ _ _ represented as −2)

\[ \sigma = 2 \]
\[ \gamma = 1 \]
\[ \lambda = 2 \]
Split sequences at runs of $\gamma + 1$ or more blank symbols if there is no overlap in left and right pivot sequences.
Splitting

\[ \lambda = 2 \]
\[ \gamma = 1 \]
\[ \sigma = 2 \]

\[ \langle b \rangle \]
\[ \langle a \ b \ b \ a \rangle \]
\[ \langle a \ b \ b \ a \rangle \quad \langle b \rangle \]
\[ \langle b \ a \rangle \quad \langle a \ b \rangle \]
\[ \langle a \ b \ _ \ _ \ a \ b \ a \rangle \]

- **Split sequences** at runs of \( \gamma + 1 \) or more blank symbols if there is no overlap in left and right pivot sequences

\[ b \]
but not \( c, d, \ldots \]
Aggregation

- Aggregate repeated sequences to save communication cost
- Remove trivial sequences (i.e., those containing only the pivot)

\[
\lambda = 2 \\
\gamma = 1 \\
\sigma = 2
\]

- \( b \)
- but not \( c, d, \ldots \)
Aggregation

- Aggregate repeated sequences to save communication cost
- Remove trivial sequences (i.e., those containing only the pivot)
Aggregation

- Aggregate repeated sequences to save communication cost
- Remove trivial sequences (i.e., those containing only the pivot)

\[ \lambda = 2 \]
\[ \gamma = 1 \]
\[ \sigma = 2 \]

\[ \langle a \ b \rangle : 1 \]
\[ \langle b \ a \rangle : 1 \]
\[ \langle a \ b \ b \ a \rangle : 2 \]
\[ \langle a \ b \ _ \ _ \ a \ b \ a \rangle : 1 \]

b

but not c, d, ...
Outline

- Motivation
- Background
  - Suffix-σ: Contiguous Sequences (EDBT ’13)
  - MG-FSM: Non-Contiguous Sequences (SIGMOD ’13)
- Experiments
- Outlook & Summary
Datasets

- **The New York Times Annotated Corpus (NYT)**
  - 1.8 million newspaper articles, 1987 – 2007, ~3 GBytes
  - 1.5 million distinct items
  - 53 million sentences consisting of 19 items on average

- **ClueWeb09-B (CW)**
  - 50 million web documents, 2009, ~246 GBytes
  - 7.3 million distinct items
  - 1.1 billion sentences consisting of 19 items on average
Setup

- **Hadoop cluster** running Cloudera CDH3u0 on Debian Linux
  - 1 master node: 2 x Xeon E5-2640, 64 GB RAM, 2 x 1 TB SATA HDD
  - 10 slave nodes: 2 x Xeon E5-2640, 64 GB RAM, 8 x 2 TB SATA HDD
  - connected via 10 GBit Ethernet

- Implementations operate on **compressed integer sequences** (using variable-byte encoding); datasets preprocessed accordingly
Contiguous Sequences

- How do Suffix-σ and MG-FSM compare against Naïve?

<table>
<thead>
<tr>
<th>Time [s]</th>
<th>Naïve</th>
<th>Suffix-σ</th>
<th>MG-FSM</th>
</tr>
</thead>
<tbody>
<tr>
<td>(100, 0, 5)</td>
<td>1,288</td>
<td>217</td>
<td>132</td>
</tr>
<tr>
<td>(10, 0, 5)</td>
<td>1,466</td>
<td>211</td>
<td>168</td>
</tr>
<tr>
<td>(10, 0, 50)</td>
<td>17,880</td>
<td>334</td>
<td>202</td>
</tr>
</tbody>
</table>
Non-Contiguous Sequences

How effective are MG-FSM and its rewriting strategies?

<table>
<thead>
<tr>
<th>Time [s]</th>
<th>Naïve</th>
<th>MG-FSM</th>
<th>+ Trim</th>
<th>+ Split</th>
<th>+ Aggregate</th>
</tr>
</thead>
<tbody>
<tr>
<td>(100, 1, 5)</td>
<td></td>
<td>13,500</td>
<td>662</td>
<td>254</td>
<td>247</td>
</tr>
</tbody>
</table>
Non-Contiguous Sequences

How scalable is MG-FSM? \((1000, 1, 5)\) on 50% of CW

![Graph showing the scalability of MG-FSM with different numbers of machines.](image-url)
Outline

- Motivation
- Background
  - Suffix-σ: Contiguous Sequences (EDBT ’13)
  - MG-FSM: Non-Contiguous Sequences (SIGMOD ’13)
- Experiments
- Outlook & Summary
Outlook

- Closed/maximal sequences to reduce to output

\[
\begin{align*}
\langle a \ b \ c \rangle &: 2 \\
\langle a \ b \rangle &: 2 \\
\langle b \ c \rangle &: 3 \\
\end{align*}
\]

- Temporal sequences with temporal gaps ("within $\gamma$ days")

\[
\begin{align*}
\langle a@t2 \ b@t4 \ c@t6 \rangle & \quad \rightarrow \quad \langle a \ c \rangle &: 2 \\
\langle a@t4 \ x@t5 \ y@t6 \ c@t8 \rangle & \quad \rightarrow \quad \gamma &= 4
\end{align*}
\]

- Generalized sequences (e.g., along part-of-speech tags)

\[
\begin{align*}
\langle a:dt \ furious:ad \ speech:nn \rangle \\
\langle the:dt \ boring:ad \ speech:nn \rangle & \quad \rightarrow \quad \langle dt \ ad \ speech:nn \rangle \\
\langle a:dt \ angry:ad \ speech:nn \rangle
\end{align*}
\]
Outlook

- **Closed/maximal sequences** to reduce to output
  
  \[
  \langle a \ b \ c \rangle : 2 \\
  \langle a \ b \rangle : 2 \\
  \langle b \ c \rangle : 3 
  \]

- **Temporal sequences** with temporal gaps ("within $\gamma$ days")
  
  \[
  \langle a@t2 \ b@t4 \ c@t6 \rangle \\
  \langle a@t4 \ x@t5 \ y@t6 \ c@t8 \rangle \\
  \]

- **Generalized sequences** (e.g., along part-of-speech tags)
  
  \[
  \langle a:dt \ \text{furious:ad} \ \text{speech:nn} \rangle \\
  \langle \text{the:dt} \ \text{boring:ad} \ \text{speech:nn} \rangle \\
  \langle a:dt \ \text{angry:ad} \ \text{speech:nn} \rangle \\
  \]

\[ \gamma = 4 \]
Outlook

- **Closed/maximal sequences** to reduce to output
  \[
  \langle a \ b \ c \rangle: 2 \\
  \langle a \ b \rangle: 2 \\
  \langle b \ c \rangle: 3
  \]

- **Temporal sequences** with temporal gaps ("within $\gamma$ days")
  \[
  \langle a@t2 \ b@t4 \ c@t6 \rangle \rightarrow \langle a \ c \rangle: 2 \\
  \langle a@t4 \ x@t5 \ y@t6 \ c@t8 \rangle \\
  y = 4
  \]

- **Generalized sequences** (e.g., along part-of-speech tags)
  \[
  \langle a:dt \ furious:ad \ speech:nn \rangle \\
  \langle the:dt \ boring:ad \ speech:nn \rangle \rightarrow \langle dt \ ad \ speech:nn \rangle \\
  \langle a:dt \ angry:ad \ speech:nn \rangle
  \]
Outlook

- **Closed/maximal sequences** to reduce to output
  
  \[
  \{ a \ b \ c \} : 2 \\
  \{ a \ b \} : 2 \\
  \{ b \ c \} : 3
  \]

- **Temporal sequences** with temporal gaps ("within $\gamma$ days")

  \[
  \{ a@t2 \ b@t4 \ c@t6 \} \\
  \{ a@t4 \ x@t5 \ y@t6 \ c@t8 \}
  \]

  \( y = 4 \)

- **Generalized sequences** (e.g., along part-of-speech tags)

  \[
  \{ a:dt \ \text{furious:ad} \ \text{speech:nn} \} \\
  \{ \text{the:dt} \ \text{boring:ad} \ \text{speech:nn} \} \rightarrow \{ \text{dt ad speech:nn} \} \\
  \{ a:dt \ \text{angry:ad} \ \text{speech:nn} \}
  \]
Summary

- Frequent sequence mining is an important building block with diverse applications including natural language processing.

- **Suffix-σ** and **MG-FSM** as two efficient & scalable algorithms to mine contiguous and non-contiguous sequences in MapReduce.

- **Code available:**
  - [https://github.com/kberberi/mpiingrams](https://github.com/kberberi/miipiingrams)