Distributed frequent sequence mining with declarative subsequence constraints

Alexander Renz-Wieland
April 26, 2017
• Sequence: succession of items
  • Words in text
  • Products bought by a customer
  • Nucleotides in DNA molecules
• Sequence: succession of *items*
  • Words in text
  • Products bought by a customer
  • Nucleotides in DNA molecules

1: Obama lives in Washington
2: Gates lives in Medina
3: The IMF is based in Washington
• Sequence: succession of *items*
  • Words in text
  • Products bought by a customer
  • Nucleotides in DNA molecules

• Goal: find **frequent sequences**

1: Obama **lives in** Washington
2: Gates **lives in** Medina
3: The IMF is based **in Washington**

→ **lives in** (2), **in Washington** (2),
  lives (2), in (2), Washington (2)
• Sequence: succession of *items*
  • Words in text
  • Products bought by a customer
  • Nucleotides in DNA molecules

• Goal: find *frequent sequences*

• Item hierarchy

1: Obama lives in Washington
2: Gates lives in Medina
3: The IMF is based in Washington

→ lives in (2), in Washington (2),
  lives (2), in (2), Washington (2),
  PERSON lives in LOCATION (2), ...
• Sequence: succession of *items*
  - Words in text
  - Products bought by a customer
  - Nucleotides in DNA molecules

• Goal: find *frequent sequences*

• Item hierarchy

• Subsequences

1: Obama lives in Washington
2: Gates lives in Medina
3: The IMF is based in Washington

Subsequences of input sequence 1:


(15 subsequences, with hierarchy: 190)
• Sequence: succession of *items*
  • Words in text
  • Products bought by a customer
  • Nucleotides in DNA molecules

• Goal: find *frequent sequences*

• Item hierarchy

• Subsequences

• Subsequence constraints

1: Obama lives in Washington
2: Gates lives in Medina
3: The IMF is based in Washington

item constraint, gap constraint, length constraint, ...
• Sequence: succession of *items*
  • Words in text
  • Products bought by a customer
  • Nucleotides in DNA molecules

• Goal: find *frequent sequences*

• Item hierarchy

• Subsequences

• Subsequence constraints

• Declarative constraints:
  (Beedkar and Gemulla, 2016)

1: Obama lives in Washington
2: Gates lives in Medina
3: The IMF is based in Washington

item constraint, gap constraint, length constraint, ...

“relational phrases between entities”
→ lives in (2)
• Sequence: succession of *items*
  • Words in text
  • Products bought by a customer
  • Nucleotides in DNA molecules

• Goal: find *frequent sequences*

• Item hierarchy

• Subsequences

• Subsequence constraints

• Declarative constraints: (Beedkar and Gemulla, 2016)

• Scalable algorithms

1: Obama lives in Washington
2: Gates lives in Medina
3: The IMF is based in Washington

item constraint, gap constraint, length constraint, ...

“relational phrases between entities”

→ **lives in** (2)
Outline

Preliminaries

Naïve approach

Proposed algorithm
  Partitioning
  Shuffle
  Local mining

Experimental evaluation
Outline

Preliminaries

Naïve approach

Proposed algorithm
  Partitioning
  Shuffle
  Local mining

Experimental evaluation
Problem definition

• Given
  • Input sequences
  • Item hierarchy
  • Constraint $\pi$
  • Minimum support threshold $\sigma$

• *Candidate sequences* of input sequence $T$:
  • Subsequences of $T$ that conform with constraint $\pi$

• Find *frequent sequences*
  • Every sequence that is a candidate sequence of at least $\sigma$ input sequences
Related work

Sequential algorithms DESQ-COUNT and DESQ-DFS (Beedkar and Gemulla, 2016)

Two distributed algorithms for Hadoop MapReduce:

• MG-FSM (Miliaraki et al., 2013; Beedkar et al., 2015)
  • Maximum gap and maximum length constraints
  • No hierarchies

• LASH (Beedkar and Gemulla, 2015)
  • Maximum gap and maximum length constraints
  • Hierarchies
Outline

Preliminaries

Naïve approach

Proposed algorithm
  Partitioning
  Shuffle
  Local mining

Experimental evaluation
Naïve approach

• “Word count”
  • Generate candidate sequences $\rightarrow$ count $\rightarrow$ filter

• Can improve by using single item frequencies
Naïve approach

• “Word count”
  - Generate candidate sequences $\rightarrow$ count $\rightarrow$ filter

• Can improve by using single item frequencies

• \textit{Problem}: a sequence of length $n$ has $O(2^n)$ subsequences
  (without considering hierarchy)
  - Typically less due to constraints, but still a problem

$\rightarrow$ Need a better approach
Outline

Preliminaries

Naïve approach

Proposed algorithm
  Partitioning
  Shuffle
  Local mining

Experimental evaluation
Overview

- Two main stages
- Partition candidate sequences
- Similar approach used in MG-FSM and LASH
Overview

node /one.osf
node /two.osf
node n

stage /one.osf: process input sequences
stage /two.osf: shuffle
stage /three.osf: local mining

input sequences
intermediary information
partitions
frequent sequences
Outline

Preliminaries

Naïve approach

Proposed algorithm

Partitioning
  Shuffle
  Local mining

Experimental evaluation
Partitioning

- Partition candidate sequences
- *Item-based partitioning*
- Pivot item

\[\text{T: abcd, ab, abc, abd, abcd, abd, abcd, abd, bd, bc, bcd, bd}\]

\[\text{P_a: ab, abc, abd, abcd}\]
\[\text{P_b: b, bc, bd, bcd}\]

\[\text{P_c: abc, bc}\]
\[\text{P_d: abd, abcd, bd, bcd}\]

\[f(a) > f(b) > f(c) > f(d)\] → reduces variance in partition sizes
Partitioning

- Partition candidate sequences

- *Item-based partitioning*

- Pivot item
  - First item
Partitioning

- Partition candidate sequences

- Item-based partitioning

- Pivot item
  - First item

\[ T: \text{abcd} \rightarrow \text{ab, abc, abd, abcd, } b, bc, bcd, bd \]

- \( P_a: \text{ab, abc, abd, abcd} \)
- \( P_b: \text{b, bc, bd, bcd} \)

\[ f(a) > f(b) > f(c) > f(d) \rightarrow \text{reduces variance in partition sizes} \]
• Partition candidate sequences

• **Item-based partitioning**

• Pivot item
  • First item
    
    T: abcd
    \[ \mathcal{P}_a: \text{ab, abc, abd, abcd} \]
    \[ \mathcal{P}_b: \text{b, bc, bcd, bd} \]

• Least frequent item
  
  T: abcd
  \[ \mathcal{P}_b: \text{ab, b} \]
  \[ \mathcal{P}_c: \text{abc, bc} \]
  \[ \mathcal{P}_d: \text{abd, abcd, bd, bcd} \]

with \( f(a) > f(b) > f(c) > f(d) \)
Partitioning

• Partition candidate sequences

• Item-based partitioning

• Pivot item
  • First item
    
    T: abcd

    \[ \mathcal{P}_a: \text{ab, abc, abd, abcd} \]
    \[ \mathcal{P}_b: \text{b, bc, bcd, bd} \]

• Least frequent item
  
  T: abcd

  \[ \mathcal{P}_b: \text{ab, b} \]
  \[ \mathcal{P}_c: \text{abc, bc} \]
  \[ \mathcal{P}_d: \text{abd, abcd, bd, bcd} \]

  with \( f(a) > f(b) > f(c) > f(d) \)

→ reduces variance in partition sizes
Overview

One partition per pivot item.
One partition per pivot item.

An input sequence is *relevant* for zero or more partitions.
One partition per pivot item.

An input sequence is *relevant* for zero or more partitions. **Next: what to shuffle?**
Outline

Preliminaries

Naïve approach

Proposed algorithm
  Partitioning
  Shuffle
  Local mining

Experimental evaluation
• Goal: from an input sequence, communicate candidate sequences to relevant partitions

• Two main options
  • Send input sequence

  • Send candidate sequences
• Goal: from an input sequence, communicate candidate sequences to relevant partitions

• Two main options
  • Send input sequence
    + compact when many candidate sequences
    - need to compute candidate sequences twice
  • Send candidate sequences
• Goal: from an input sequence, communicate candidate sequences to relevant partitions

• Two main options
  • Send input sequence
    + compact when many candidate sequences
    - need to compute candidate sequences twice
  • Send candidate sequences
    + compact when candidate sequences are short and few per partition
Shuffle

• Goal: from an input sequence, communicate candidate sequences to relevant partitions

• Two main options
  • Send input sequence
    + compact when many candidate sequences
    - need to compute candidate sequences twice
  • Send candidate sequences
    + compact when candidate sequences are short and few per partition

→ Focus on sending candidate sequences
→ Try to represent them compactly
A compact representation for candidate sequences

- Goal: compactly represent set of candidate sequences
- Trick: exploit shared structure

\{caabe, caaBe, caAbe, caABe, cAabe, cAaBe, cAAbe, cAABe, cbe, cBe\}

Naïve NFA

- Compressed NFA
A compact representation for candidate sequences

• Goal: compactly represent set of candidate sequences
• Trick: exploit shared structure

\{caabe, caaBe, caAbe, caABe, cAabe, cAaBe, cAAbe, cAABe, cbe, cBe\}
A compact representation for candidate sequences

- Goal: compactly represent set of candidate sequences
- Trick: exploit shared structure

\{caabe, caaBe, caAbe, caABe, cAabe, cAaBe, cAAbe, cAABe, cbe, cBe\}

- Naïve NFA
A compact representation for candidate sequences

- Goal: compactly represent set of candidate sequences
- Trick: exploit shared structure

\{caabe, caaBe, caAbe, cAABe, cAabe, cAaBe, cAABe, cAABe, cbe, cBe\}

- Naïve NFA

- Compressed NFA
Shuffling NFAs

Constructing NFAs

• Per input sequence, build one NFA for each relevant partition
• Naïve: generate all candidate sequences, compress
• Better: build directly from *Finite State Transducer*
Constructing NFAs

• Per input sequence, build one NFA for each relevant partition
• Naïve: generate all candidate sequences, compress
• Better: build directly from *Finite State Transducer*

Serialization

• Send structure and items
Shuffling NFAs

Constructing NFAs

- Per input sequence, build one NFA for each relevant partition
- Naïve: generate all candidate sequences, compress
- Better: build directly from *Finite State Transducer*

Serialization

- Send structure and items
- Many “simple” NFAs

\[
\text{start} \xrightarrow{\{a\}} 0 \xrightarrow{\{b\}} 1 \xrightarrow{\{c\}} 2 \xrightarrow{\{\}\text{zero.osf}/\text{one.osf}/\text{two.osf}/\text{three.osf}} 3
\]
Outline

Preliminaries

Naïve approach

Proposed algorithm

Partitioning

Shuffle

Local mining

Experimental evaluation
Overview

- node /one.osf
- node /two.osf
- node

---

**stage /one.osf:** process input sequences

**stage /two.osf:** shuffle

**stage /three.osf:** local mining

**input sequences** → **intermediary information** → **partitions** → **frequent sequences**

---

node 1

node 2

... nodes

---

stage 1: process input sequences

stage 2: shuffle

stage 3: local mining

---

**Done: How to partition? What to shuffle?**
Overview

input sequences → intermediary information → partitions → frequent sequences

node 1
node 2
...
node n

stage 1: process input sequences
stage 2: shuffle
stage 3: local mining

Done: How to partition? What to shuffle?
Next: How to process the partitions?
Local mining

• Partition for pivot item $p$
  • Given: list of NFAs
  • Goal: mine frequent sequences with pivot item $p$

• Pattern-growth approach (Pei et al., 2001)
Outline

Preliminaries

Naïve approach

Proposed algorithm
  Partitioning
  Shuffle
  Local mining

Experimental evaluation
Experimental setup

- Implementation
  - In Java and Scala
  - For Apache Spark

- Experiments on cluster with 8 worker nodes
  - 8 cores per node
  - 64 GB memory per node

- Here: two datasets
  - 50 million sentences from New York Times
  - Product reviews of 21 million Amazon users
Non-traditional constraints

• Constraints that cannot be expressed with traditional methods
• Compare to count-based approach
Non-traditional constraints

- Constraints that cannot be expressed with traditional methods
- Compare to count-based approach

![Graph showing total run time (seconds) for different constraints with Count and DDIN compared]
Non-traditional constraints

- Constraints that cannot be expressed with traditional methods
- Compare to count-based approach

→ DDIN not slower for selective constraints $N_1, N_2, N_3,$ and $A_2$
→ DDIN up to $50\times$ faster for unselective constraints $N_4, N_5, A_1, A_3,$ and $A_4$
Traditional constraints

• Compare to LASH, state-of-the art distributed algorithm
• Maximum gap and maximum length constraints, hierarchies
Traditional constraints

- Compare to LASH, state-of-the-art distributed algorithm
- Maximum gap and maximum length constraints, hierarchies

LASH (Hadoop) vs. DDIN (Spark)

DDIN generally competitive to LASH, despite being more general

The fewer candidate sequences, the better DDIN
Traditional constraints

- Compare to LASH, state-of-the-art distributed algorithm
- Maximum gap and maximum length constraints, hierarchies

→ DDIN generally competitive to LASH, despite being more general
→ The fewer candidate sequences, the better DDIN
More findings

• Scales linearly
  • Tested effect of dataset size, weak and strong scalability

• Main limitation
  • Many candidate sequences with no common structure
  • Better approach: send input sequence
Conclusion

• Distributed algorithm for frequent sequence mining with declarative subsequence constraints

• Item-based partitioning, shuffles candidate sequences as NFA

• Can mine a wide range of constraints

• Outperforms naïve approach, competitive to LASH, scales linearly
Conclusion

- Distributed algorithm for frequent sequence mining with declarative subsequence constraints
- Item-based partitioning, shuffles candidate sequences as NFA
- Can mine a wide range of constraints
- Outperforms naïve approach, competitive to LASH, scales linearly

Thank you!


Jian Pei, Jiawei Han, Behzad Mortazavi-Asl, Helen Pinto, Qiming Chen, Umeshwar Dayal, and Mei-Chun Hsu. Prefixspan: mining sequential patterns efficiently by prefix-projected pattern growth. ICDE ’01, pages 215–224. IEEE, 2001.