# Stream Sequential Pattern Mining with Precise Error Bounds

Luiz F. Mendes[1,2] Bolin Ding[1] Jiawei Han[1]

[1] University of Illinois at Urbana-Champaign [2] Google Inc.

lmendes@google.com {bding3,hanj}@illinois.edu

## Outline

## Introduction

- Sequential pattern mining is an important problem with many real-world applications.
- In recent years, we have seen a new kind of data, referred to as *data stream*: an unbounded sequence in which new elements are generated continuously.
- Additional constraints for mining data streams:
  - Memory usage is limited (cannot store everything)
  - Can only look at each stream component once

## Introduction (cont.)

- Two effective methods for mining sequential patterns from data streams:
  - ***SS-BE*** (Stream Sequence miner using Bounded Error)
    - Guarantees there are no false negatives.
    - Ensures the support count of the false positives is above some pre-defined threshold.
  - ***SS-MB*** (Stream Sequence miner using Memory Bounds)
    - Maximum memory usage after processing any batch can be controlled explicitly.

Source: www.belgravium.com

## Outline

- Introduction
- Problem Definition
- The *SS-BE* Method
- The *SS-MB* Method
- Experimental Results
- Discussion
- Conclusions

## Problem Definition

- Let I = $\{i_1, i_2, \ldots, i_j\}$ be a set of *j* items.
- A sequence is an ordered list of items from I denoted by $<s_1, s_2, \ldots, s_k>$.
- A sequence $<a_1, a_2, \ldots, a_p>$ is a subsequence of another sequence $<b_1, b_2, \ldots, b_q>$ if there exist integers $i_1 < i_2 < \ldots < i_p$ such that $a_1 = b_{i1}, a_2 = b_{i2}, \ldots, a_n = b_{ip}$.
- A *data stream* of sequences is an arbitrarily large list of sequences.
- A sequence *s contains* another sequence *s'* if *s'* is a subsequence of *s*.

## Problem Definition (cont.)

- The *count* of a sequence *s*, denoted by *count(s)*, is defined as the number of sequences that contain *s*.
- The *support* of a sequence *s*, also called *supp(s)*, is *count(s)* divided by the total number of sequences seen.
- If *supp(s)* >= *σ*, where *σ* is a user-supplied minimum support threshold, then *s* is a frequent sequence, or a sequential pattern.
- Goal is to find all the frequent sequential patterns in our data stream (or at least as close as possible in the stream case).

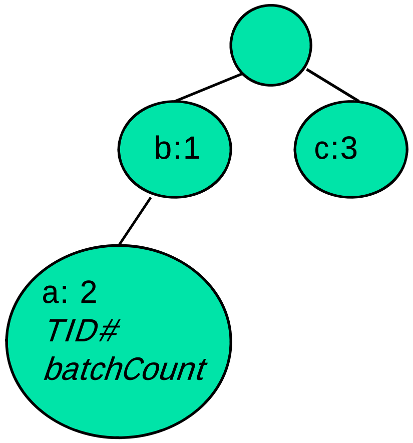
## Problem Definition (cont.)

- Example:
  - Given data stream $D$: $S_1$ = <$a$,$b$,$c$>, $S_2$ = <$a$,$c$>, and $S_3$ = <$b$,$c$>.
  - $\sigma = 0.5$.
  - The set of sequential patterns and their corresponding counts is as follows:
    - <$a$>: 2
    - <$b$>:2
    - <$c$>:3
    - <$a$,$c$>:2
    - <$b$,$c$>:2

## Outline

## *SS-BE* Method

- Input:
  - A data stream $D = S_1, S_2, S_3, \ldots$
  - Minimum support threshold $\sigma$
  - Significance threshold $\epsilon$, $0 <= \epsilon < \sigma$
  - Batch support threshold $\alpha$, $0 <= \alpha <= \epsilon$
  - Batch length $L$
  - Pruning period $\delta$
- Use a tree $T_0$ to store subsequences seen in the stream



## *SS-BE* Method (cont.)

- Algorithm Overview:
  - Break the stream into batches of length *L*.
  - For each arriving batch $B_k$, apply PrefixSpan with support α.
    - Insert each frequent sequence $s_i$ (say it has count $c_i$) into $T_0$ by incrementing *count* of node corresponding to it by $c_i$ and *batchCount* by 1.
      - If a path corresponding to this sequence does not exist in the tree, then one must first be created, setting the *batchCount* and *count* values of the new nodes to 0 and the *TID* values to *k*.

## *SS-BE* Method (cont.)

- When the number of batches seen is a multiple of the pruning period δ, prune the tree by eliminating all sequences (nodes) where:
  - $[count + (\lceil \alpha L \rceil - 1) * B'] <= \epsilon * (BL)$

    where *B* is the number of batches elapsed since the last pruning before the sequence was inserted in the tree, and *B'* is the number of these batches that did not modify the count of the sequence in the tree (note that *B'* = *B* - *batch_count*).
- When we find that a node can be pruned, the entire sub-tree rooted at that node can be pruned as well.

# *SS-BE* Method (cont.)

- Finally, suppose the user requests the set of frequent sequences after *N* sequences have been seen in the stream.
- Simply traverse the tree outputting all sequences corresponding to nodes having *count* >= (σ-ϵ)*N*.
  - There are no false negatives.
  - The false positives are guaranteed to have real support count at least (σ-ϵ) *N*.

## *SS-BE* Example Execution

- Suppose $L$ = 4, σ = 0.75, ϵ = 0.5, α = 0.4, and δ = 2.
- Data stream $D$:
  - <*a*,*b*,*c*>
  - <*a*,*c*>
  - <*a*,*b*>
  - <*b*,*c*>

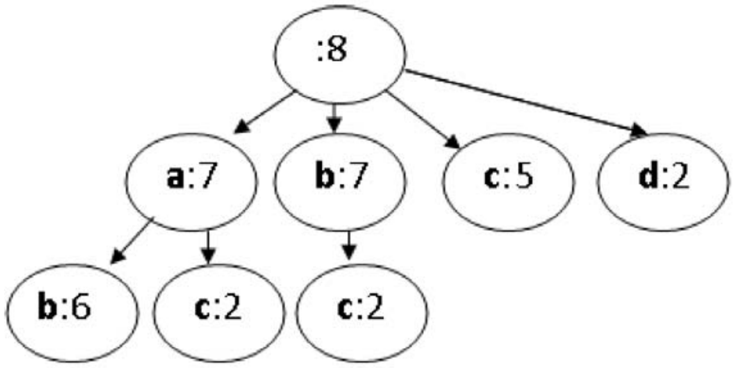
  Batch $B_1$

  - <*a*,*b*,*c*,*d*>
  - <*c*,*a*,*b*>
  - <*d*,*a*,*b*>
  - <*a*,*e*,*b*>

  Batch $B_2$

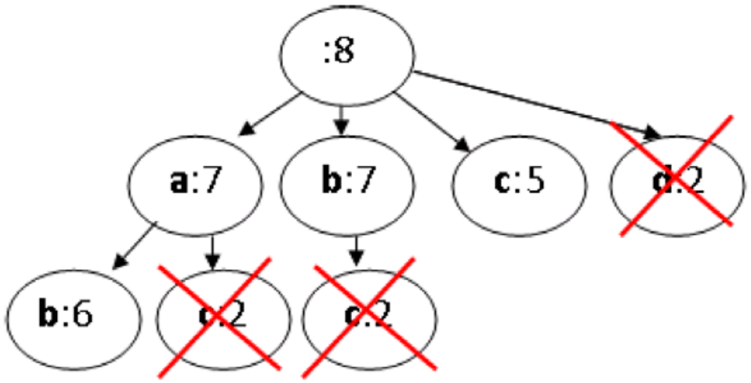
# *SS-BE* Example Execution (cont.)

- Apply PrefixSpan to $B_1$ with minimum support 0.4. The frequent sequences found are:
  - <*a*>:3, <*b*>:3, <*c*>:3, <*a*,*b*>:2, <*a*,*c*>:2, and <*b*,*c*>:2
- The algorithm then moves on to $B_2$. The frequent sequences found are:
  - <*a*>:4, <*b*>:4, <*c*>:2, <*d*>:2, and <*a*,*b*>:4



# *SS-BE* Example Execution (cont.)

- Because the pruning period is 2, we must now prune the tree.
- For each node, *B* is the number of batches elapsed since the last pruning before that node was inserted in the tree, and *B'* = *B* – *batchCount*.
- We prune all nodes satisfying:
  - $count + B' (\lceil \alpha L \rceil - 1) <= B \in L$
    - -> $count + B' <= 4$



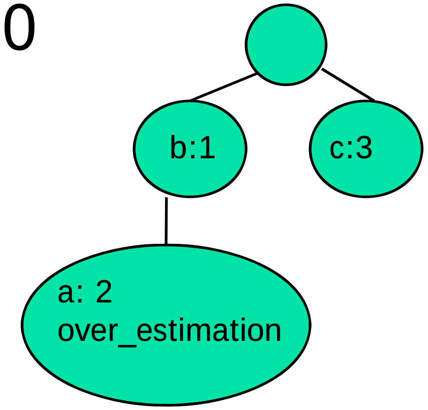
# *SS-BE* Example Execution (cont.)

- When the user requests the set of sequential patterns, the algorithm outputs all sequences corresponding to nodes having count at least $(\sigma-\epsilon)N = (0.75 - 0.5) * 8 = 2$.
- The output sequences and counts are:
  - <*a*>: 7
  - <*b*>: 7
  - <*c*>: 5
  - <*a*, *b*>:6
- There are no false negatives and only one false positive: <*c*>

## Outline

## *SS-MB* Method

- Input:
  - A data stream $D = S_1, S_2, S_3, \ldots$
  - Minimum support threshold σ
  - Significance threshold ∊, 0 <= ∊ < σ
  - Batch length *L*
  - Maximum number of nodes in the tree *m*
- Use a tree $T_0$ to store subsequences seen in the stream
- Use variable *min*, initially set to 0



## *SS-MB* Method (cont.)

- Algorithm Overview:
  - Break the stream into batches of length $L$.
  - For each arriving batch $B_k$, apply PrefixSpan with support $\epsilon$.
    - Insert each frequent sequence $s_i$ (say it has count $c_i$) into $T_0$ by incrementing *count* of node corresponding to it by $c_i$.
      - If a path corresponding to this sequence does not exist in the tree, then one must first be created, setting the *over_estimation* and *count* values of the new nodes to *min*.

## *SS-MB* Method (cont.)

- After processing each batch, we check whether the number of nodes in the tree exceeds *m*.
- While this is true, we remove from the tree the node of minimum count, and set *min* to equal the count of the last node removed.

## *SS-MB* Method (cont.)

- Finally, suppose the user requests the set of frequent sequences after *N* sequences have been seen in the stream.
- Simply traverse the tree outputting all sequences corresponding to nodes having *count* > (σ-ϵ)*N*.
  - Nodes having *(count – over-estimation)* >= σ*N* are *guaranteed* to be frequent.
  - If *min* <= (σ-ϵ)*N*, then the algorithm *guarantees* there are no false negatives.

## *SS-MB* Example Execution

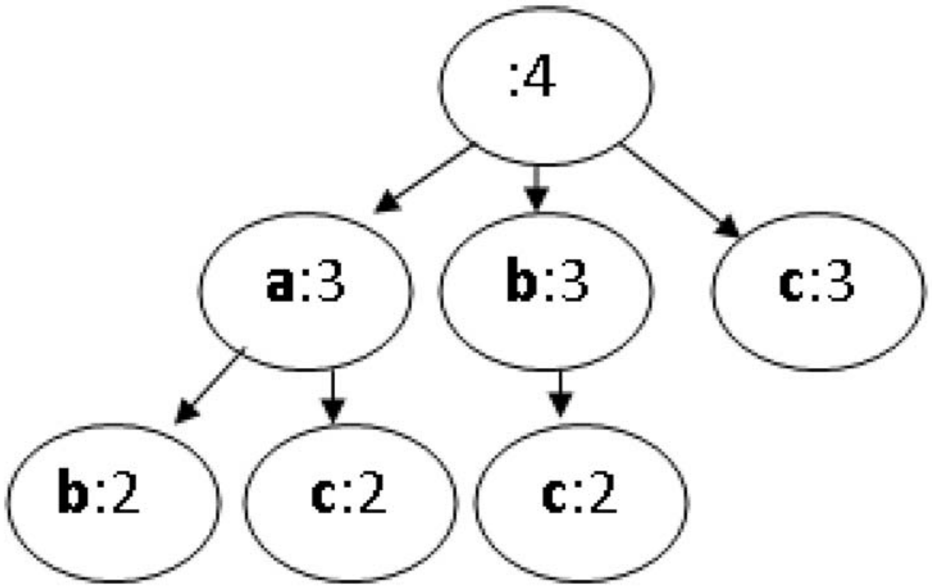
- Suppose $L$ = 4, σ = 0.75, ϵ = 0.5, and $m$ = 7.
- Data stream $D$:
  - <*a*,*b*,*c*>
  - <*a*,*c*>
  - <*a*,*b*>
  - <*b*,*c*>

    Batch $B_1$
  - <*a*,*b*,*c*,*d*>
  - <*c*,*a*,*b*>
  - <*d*,*a*,*b*>
  - <*a*,*e*,*b*>

    Batch $B_2$

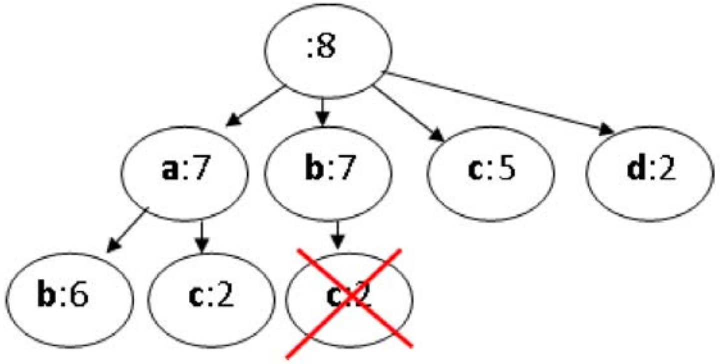
# *SS-MB* Example Execution (cont.)

- Apply PrefixSpan to $B_1$ with minimum support 0.5. The frequent sequences found are:
  - <*a*>:3, <*b*>:3, <*c*>:3, <*a*,*b*>:2, <*a*,*c*>:2, and <*b*,*c*>:2



# *SS-MB* Example Execution (cont.)

- The algorithm then moves on to $B_2$. The frequent sequences found are:
  - <*a*>:4, <*b*>:4, <*c*>:2, <*d*>:2, and <*a*,*b*>:4
- Because there are now 8 nodes in the tree and the maximum is 7, we must remove the sequence having minimum count from the tree.
  - sequence <*b*,*c*> is removed
  - *min* is set to this sequence's *count*, 2.



## *SS-MB* Example Execution (cont.)

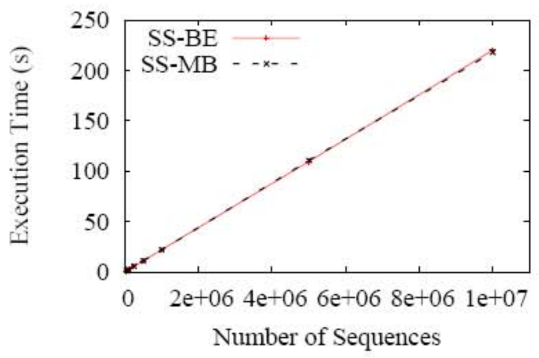
- When the user requests the set of sequential patterns, the algorithm outputs all sequences corresponding to nodes having count above $(\sigma-\epsilon)N = (0.75 - 0.5) * 8 = 2$.
- The output sequences and counts are:
  - <*a*>: 7
  - <*b*>: 7
  - <*c*>: 5
  - <*a*, *b*>:6
- Because *min* = 2 <= $(\sigma-\epsilon)N = 2$, the algorithm *guarantees* that there are no false negatives. In this case, there is only one false positive: <*c*>

## Outline

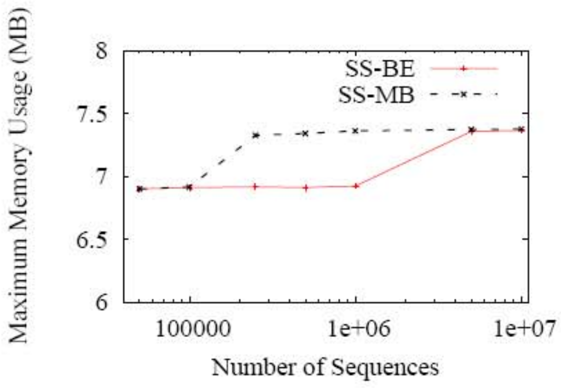
- Introduction
- Problem Definition
- The *SS-BE* Method
- The *SS-MB* Method
- Experimental Results
- Discussion
- Conclusions

# Experimental Results

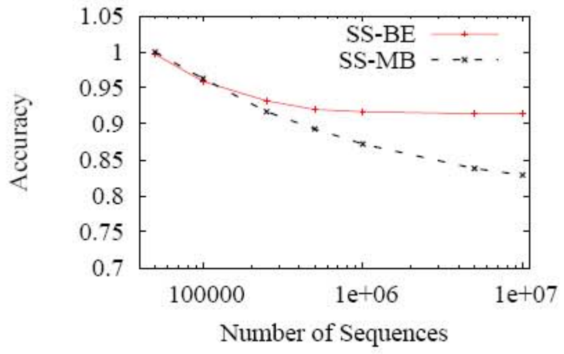
- Varying the number of sequences



(a) Execution Time



(b) Maximum Memory Usage

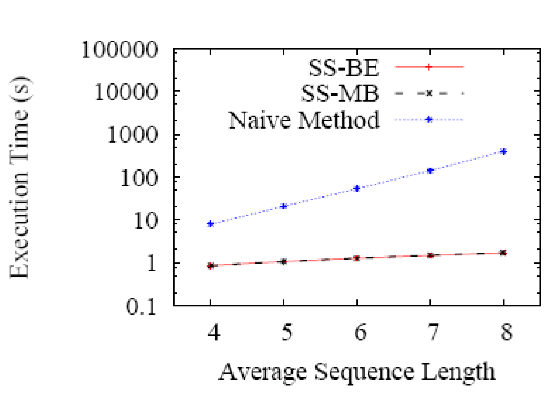


(c) Accuracy

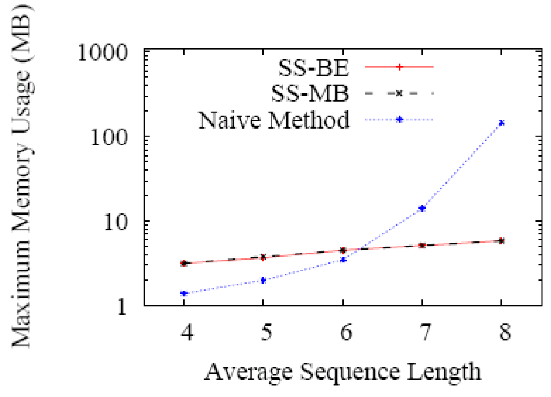
- Number of distinct items: 100
- Average sequence length: 10
- Minimum support threshold σ: 0.01
- Significance threshold ∊ : 0.00999
- Batch length *L*: 50,000
- Batch support threshold α (in *SS-BE*): 0.00995
- Prune period δ (in *SS-BE*): 4 batches
- Maximum number of nodes in the tree *m* (in *SS-MB*): the smallest possible value such that the algorithm still guaranteed that all true sequential patterns were output (on average, the ratio of *m* to the number of true sequential patterns was 1.115)

# Experimental Results (cont.)

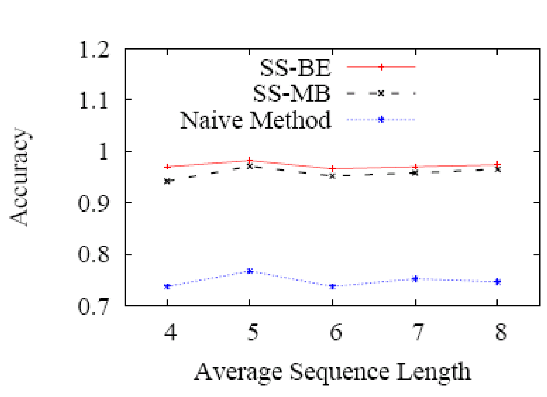
- Varying the average sequence length



(d) Execution Time



(e) Maximum Memory Usage



(f) Accuracy

- Number of distinct items: 100
- Total number of sequences: 100,000
- Minimum support threshold σ: 0.01
- Significance threshold ∊ : 0.0099
- Batch length *L*: 50,000
- Batch support threshold α (in *SS-BE*): 0.0095
- Prune period δ (in *SS-BE*): 1 batch
- Maximum number of nodes in the tree *m* (in *SS-MB*): the smallest possible value such that the algorithm still guaranteed that all true sequential patterns were output (on average, the ratio of *m* to the number of true sequential patterns was 1.054)
- We compare with a naïve method that finds all the possible subsequences for each sequence that arrives in the data stream, inserting each one into a tree like $T_0$, that is also pruned periodically.

## Outline

## Discussion

- Main advantage of *SS-BE* is that it always *guarantees* no false negatives, and also places a bound on the support of the false positives.
- However, no precise relationship between the significance threshold parameter ∊ and the maximum memory usage
    - may pick a value for ∊ too large or too small
- By exploiting all of the available memory in the system, *SS-MB* may be able to achieve greater accuracy than *SS-BE* in some cases.

## Outline

## Conclusions

- *SS-BE* always ensures there are no false negatives, while also guaranteeing that the true support of the false positives is above some pre-defined threshold.
- *SS-MB* is only guaranteed to have no false negatives if at the end of the algorithm $min <= (\sigma-\epsilon)N$.
- Our proposed methods are effective solutions to the stream sequential pattern mining problem:
  - The running time of each algorithm scales linearly as the number of sequences grows.
  - The maximum memory usage is restricted in both cases through the pruning strategies adopted.
  - Our experiments show that both methods produce a very small number of false positives.

## Thanks and Questions