Mining Maximal Sequential Patterns without Candidate Maintenance

Philippe Fournier-Viger[1]

Cheng-Wei Wu[2]

Vincent Shin-Mu Tseng[2]

[1]University of Moncton, Canada

[2]National Cheng Kung University, Taiwan

UNIVERSITÉ DE MONCTON

16/12/2013

ADMA2013

Introduction

Sequential pattern mining:

- a data mining task with wide applications
- finding frequent subsequences in a **sequence database**.

Example:

minsup = 2

Sequence database

SID	Sequences
1	⟨{a, b},{c},{f, g},{g},{e}⟩
2	⟨{a, d},{c},{b},{a, b, e, f}⟩
3	<{a},{b},{f,g},{e}>
4	⟨{b},{f, g}⟩

→

Some sequential patterns

ID	Pattern	Supp.
p1	⟨{a},{f}⟩	3
p2	⟨{a},{c}{f}⟩	2
p3	⟨{b},{f,g}⟩	2
p4	⟨{g},{e}⟩	2
p5	⟨{c},{f}⟩	2
p6…	⟨{b}⟩	4

Algorithms

Different approaches to solve this problem

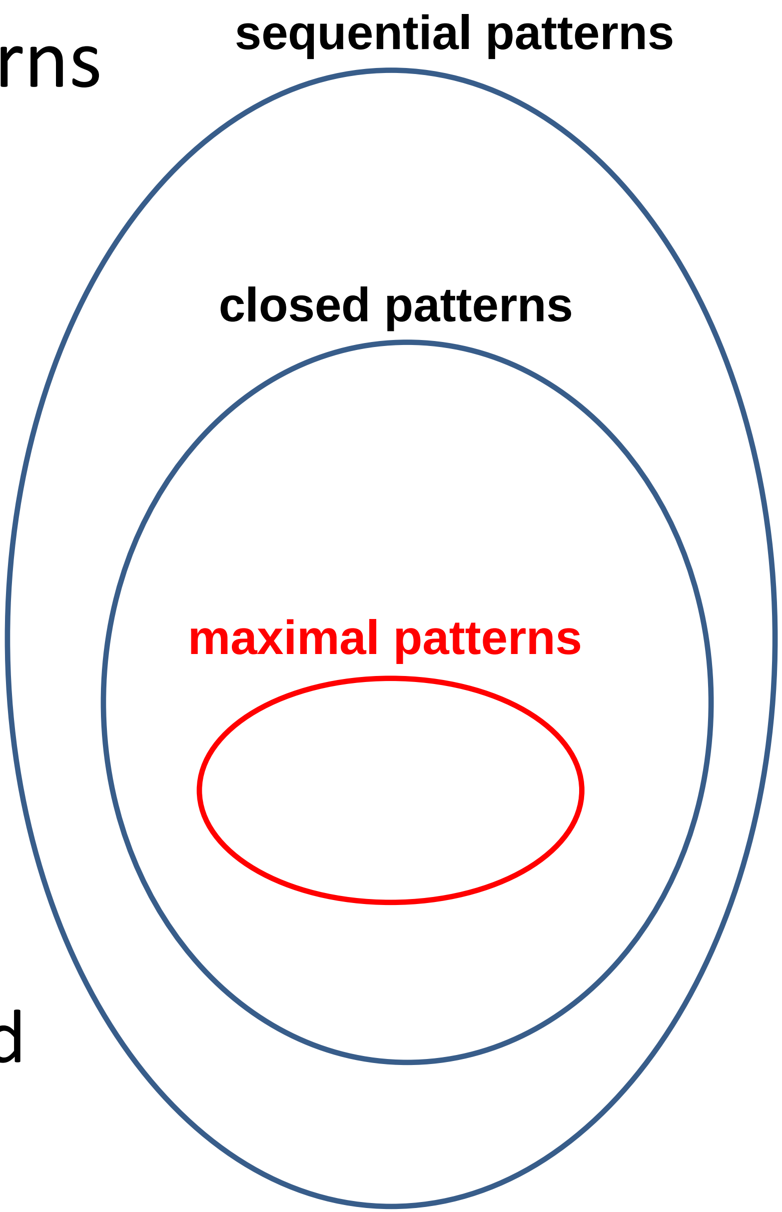
- Apriori-based
  (e.g. GSP)
- Pattern-growth
  (e.g. PrefixSpan)
- Discovery of sequential patterns using a vertical database representation
  (e.g. SPADE and SPAM)

The problem of redundancy

- A lot of redundancy in sequential patterns.
- If a pattern of 20 distinct items is frequent, then all its $2^{20}-1$ subsequences are also frequent!
- **Example**: the pattern {c},{f}, the pattern {a}, the pattern {c} … are frequent if {a},{c},{f} is frequent.
- Because of redundancy,
  – it can be very time-consuming to analyze patterns,
  – it can require much more storage space.

A solution

- **Closed sequential patterns:** patterns that are not included in another pattern having the same support.
  - lossless
  - this set is still quite large for some applications
- **Maximal sequential patterns:** a pattern that is not included in another pattern.
  - lossless
  - generally much smaller than closed patterns



Algorithms

Closed sequential pattern mining

- BIDE, CloSpan, Clasp…

Maximal sequential pattern mining

- approximate solution: **MSPX**
- for strings with no repeating items: **DISMAPS**
- for the general problem: **AprioriAdjust, MSPX, MFSPAN**
  - not memory efficient
  - need to maintain a large set of intermediate candidates in memory during the mining process

Our proposal

MaxSP:

- to discover maximal sequential patterns without maintaining candidates.
- Uses a pattern-growth approach based on *PrefixSpan*
- Integrated a mechanism to check if a pattern is maximal, which has similarities to *Bide*.

MaxSP – search procedure

Basic idea:

1. Discover frequent patterns of size 1 by scanning the database.
2. For each pattern *S*,
   a) make a database projection with *S*.
   b) Scan the projected database to discover each single items that is frequent and can thus be appended to *S* to create a larger patterns *T*.
   c) Perform step 2 recursively for each such pattern.

Output: all sequential patterns

MaxSP – maximality checking

- Consider a pattern *S* = $\{a_1, a_2, \ldots a_n\}$
- Two steps to check if a pattern *S* is maximal:
  - **maximal-forward extension checking**
    - if an item is found in a projected database that can be appended to *S* (after the last item), then *S* is not maximal
  - **maximal-backward extension checking**
    - scan the sequences containing *S*.
    - if an item can be appended to *S* before the first item, or between any two items of *S*, then *S* is not frequent.

Experimental Evaluation

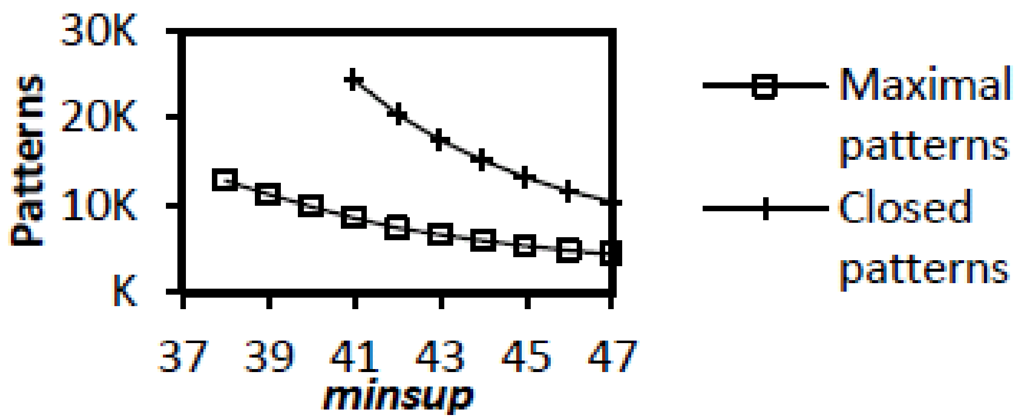
Datasets' characterictics

dataset	sequence count	distinct item count	avg. seq. length (items)	type of data
BMS	59,601	497	2.51 (std = 4.85)	web click stream
Snake	163	20	60 (std = 0.59)	protein sequences
Sign	730	267	51.99 (std = 12.3)	language utterances
Leviathan	5,834	9,025	33.81 (std= 18.6)	book
FIFA	20,450	2,990	34.74 (std = 24.08)	web click stream

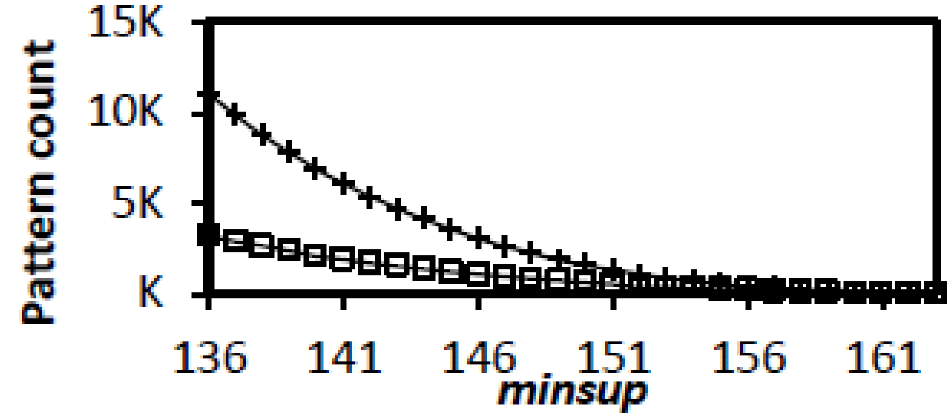
- MaxSP vs BIDE
- All algorithms implemented in Java
- Windows 7, 1 GB of RAM

Pattern count

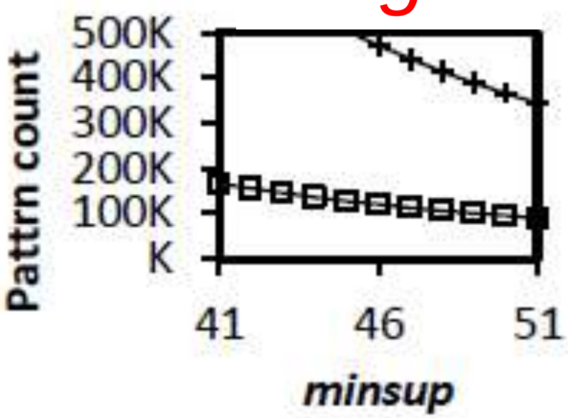
BMS



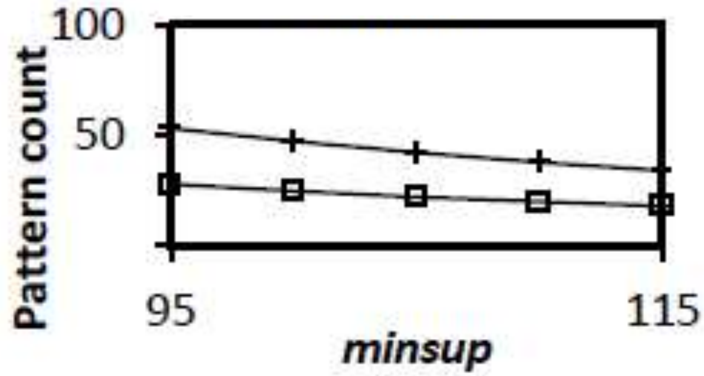
Snake



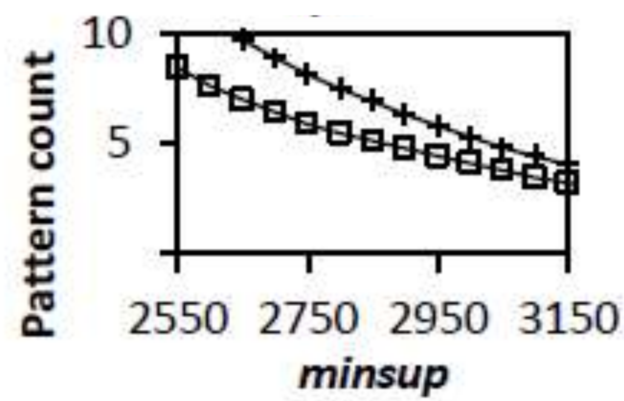
Sign



Leviathan



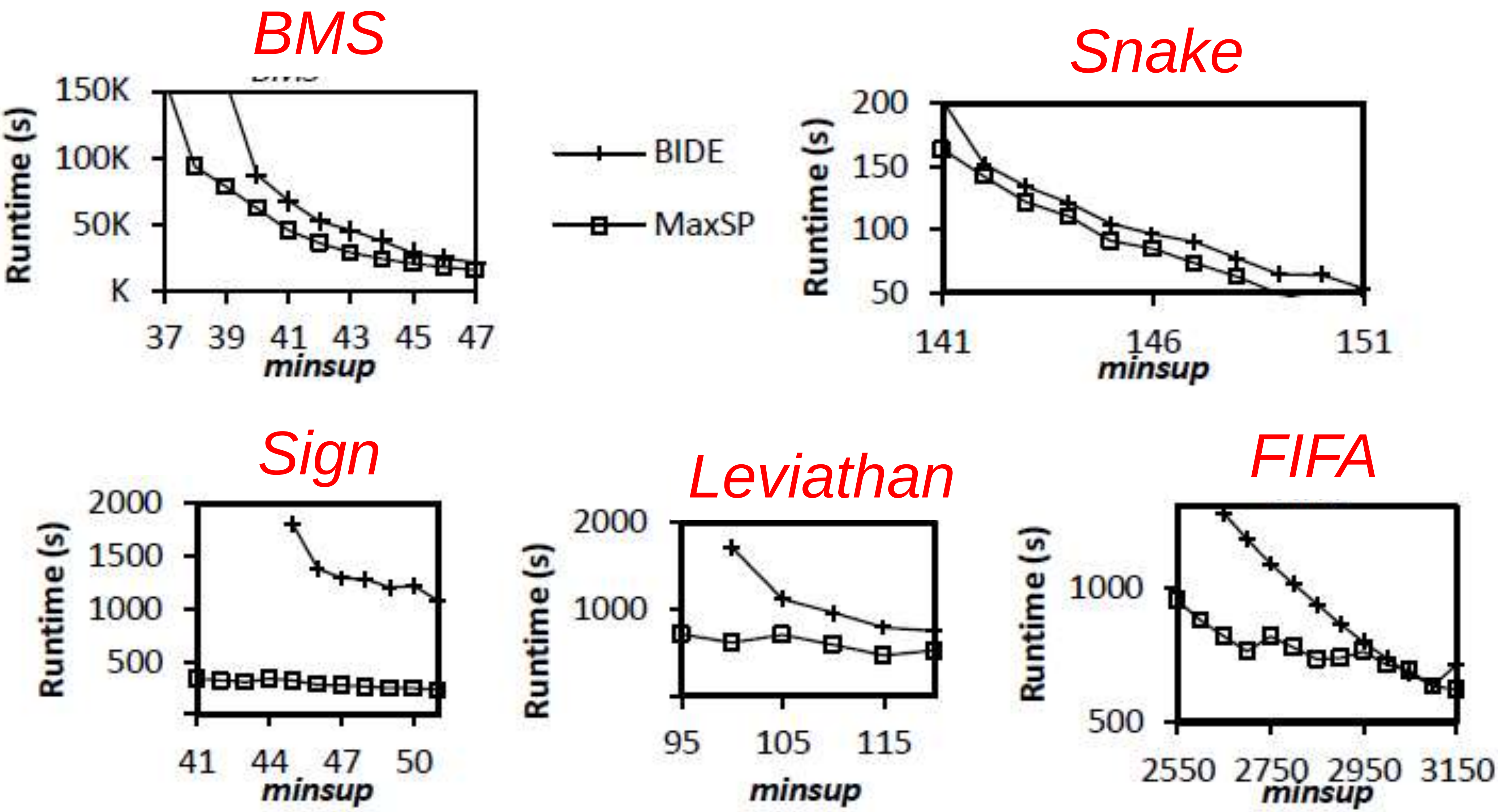
FIFA



Much less maximal sequential patterns than closed patterns

eg.: Snake – 28 %, Sign = 25 %

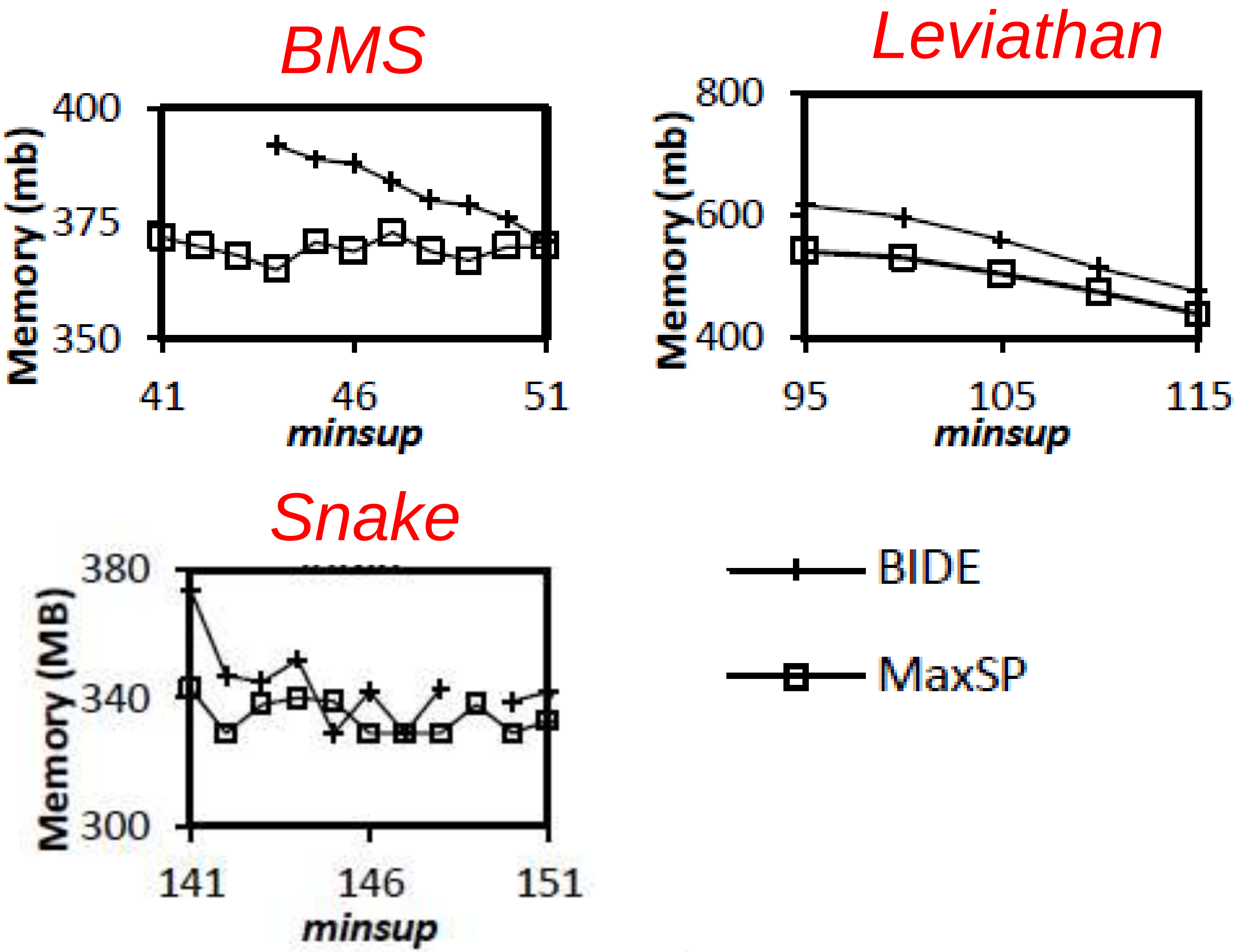
Execution time



MaxSP is up to five times faster than BIDE

Two reasons: easier to meet pruning conditions and write operations to disk

Memory Usage



MaxSP consumes less memory
(less sequences need to be scanned, less patterns need to be created)

Conclusion

- **MaxSP**
  - a new pattern-growth algorithm to discover maximal sequential patterns
  - avoid the problem of candidate maintenance (can output pattern to disk immediately)
  - outperforms the BIDE algorithm in execution time and memory, and has better scalability.
- Source code and datasets available as part of the **SPMF data mining library (**GPL 3).

SPMF **Open source Java data mining software**, 55 algorithms
http://www.phillippe-fournier-viger.com/spmf/

Thank you. Questions?

SPMF

Open source Java data mining software, 55 algorithms
http://www.phillippe-fournier-viger.com/spmf/

Example

SID	Sequences
1	$\langle\{a, b\},\{c\},\{f, g\},\{g\},\{e\}\rangle$
2	$\langle\{a, d\},\{c\},\{b\},\{a, b, e, f\}\rangle$
3	$\langle\{a\},\{b\},\{f\},\{e\}\rangle$
4	$\langle\{b\},\{f, g\}\rangle$

Fig. 1. A sequence database

Pattern	Sup.	Pattern	Sup.
$\langle\{a\}\rangle$	3⟩ ⟩ C	$\langle\{b\},\{g\},\{e\}\rangle$	2 CM
$\langle\{a\},\{g\}\rangle$	2 ⟩	$\langle\{b\},\{f\}\rangle$	4 C ⟩
$\langle\{a\},\{g\},\{e\}\rangle$	2⟩ CM	$\langle\{b\},\{f, g\}\rangle$	2 CM
$\langle\{a\},\{f\}\rangle$	3⟩ C	$\langle\{b\},\{f\},\{e\}\rangle$	2 CM
$\langle\{a\},\{f\},\{e\}\rangle$	2⟩ CM	$\langle\{b\},\{e\}\rangle$	3 C
$\langle\{a\},\{c\}\rangle$	2⟩	$\langle\{c\}\rangle$	2⟩
$\langle\{a\},\{c\},\{f\}\rangle$	2 CM	$\langle\{c\},\{f\}\rangle$	2⟩
$\langle\{a\},\{c\},\{e\}\rangle$	2 CM	$\langle\{c\},\{e\}\rangle$	2⟩
$\langle\{a\},\{b\}\rangle$	2⟩	$\langle\{e\}\rangle$	3⟩
$\langle\{a\},\{b\},\{f\}\rangle$	2⟩ CM	$\langle\{f\}\rangle$	4⟩
$\langle\{a\},\{b\},\{e\}\rangle$	2⟩ CM	$\langle\{f, g\}\rangle$	2⟩
$\langle\{a\},\{e\}\rangle$	3⟩ C	$\langle\{f\},\{e\}\rangle$	2⟩
$\langle\{a, b\}\rangle$	2⟩ CM	$\langle\{g\}\rangle$	3⟩
$\langle\{b\}\rangle$	4⟩	$\langle\{g\},\{e\}\rangle$	2⟩
$\langle\{b\},\{g\}\rangle$	3⟩ C ⟩		

C = Closed M = Maximal

Fig. 2. Sequential patterns found for *minsup* = 2 (right)