Fast Vertical Mining of Sequential Patterns Using Co-occurrence Information

Philippe Fournier-Viger[1]

Antonio Gomariz[2]

Manuel Campos[2]

Rincy Thomas[3]

[1]University of Moncton, Canada

[2]University of Murcia, Spain

[3]SCT, India

May 14 2014 – 10:20 AM

PAKDD 2014

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

Some sequential patterns

ID	Pattern	Supp.
p1	$\langle \{a\}, \{f\} \rangle$	3
p2	$\langle \{a\}, \{c\}\{f\} \rangle$	2
p3	$\langle \{b\}, \{f,g\} \rangle$	2
p4	$\langle \{g\}, \{e\} \rangle$	2
p5	$\langle \{c\}, \{f\} \rangle$	2
p6…	$\langle \{b\} \rangle$	4

Pattern-Growth based algorithms

- PrefixSpan, CloSpan, BIDE+…
  - Find frequent patterns containing single items.
  - For each frequent pattern, perform database projection and count frequency of items that could extend the pattern
- Does not generate candidates.
- Drawback: database projection is very costly.

Vertical algorithms

- SPAM, SPADE, bitSPADE, ClaSP, VMSP
  - Read the database to convert it to a vertical representation (**sids lists**)

a	
SID	Itemsets
1	1
2	1,4
3	1
4	

b	
SID	Itemsets
1	1
2	3,4
3	2
4	1

c	
SID	Itemsets
1	2
2	2
3	
4	

d	
SID	Itemsets
1	
2	1
3	
4	

e	
SID	Itemsets
1	5
2	4
3	4
4	

f	
SID	Itemsets
1	3
2	4
3	3
4	2

g	
SID	Itemsets
1	3,4
2	
3	
4	2

  - Perform a depth-first search by joining items to each pattern by **i-extension** and **s-extension**

Vertical algorithms (cont'd)

– Calculate the support of a pattern by join operation of sid lists

a	
SID	Itemsets
1	1
2	1,4
3	1

support = 3

b	
SID	Itemsets
1	1
2	3,4
3	2
4	1

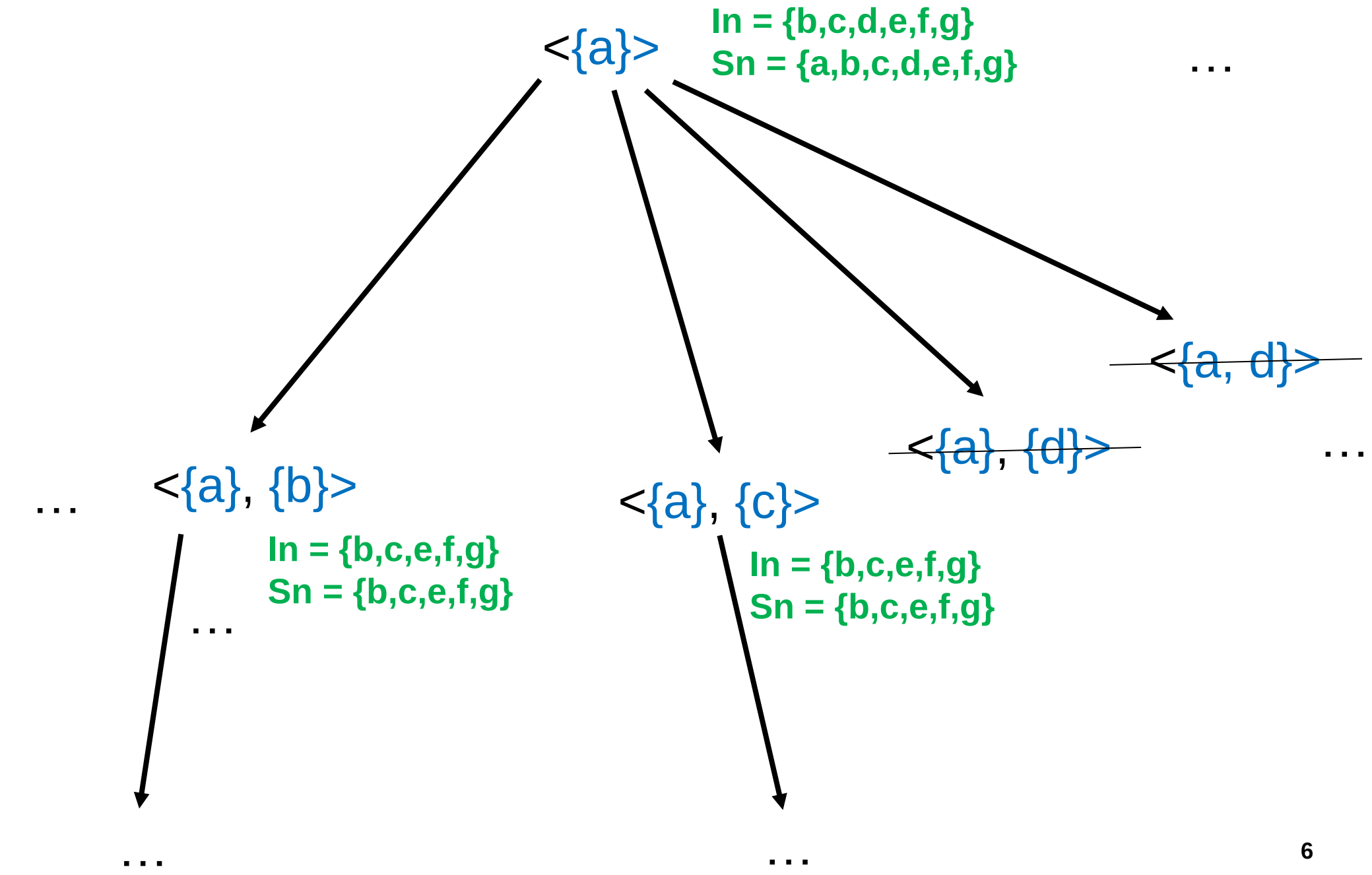
support = 4

<{a}, {b}>	
SID	Itemsets
1	1
2	1, 3
3	2

support = 3

- Does not require to scan the database more than once.
- **Drawback**: generate a huge amount of candidates
- Could we improve performance by pruning candidates?

SPAM


<{a}>
In = {b,c,d,e,f,g}
Sn = {a,b,c,d,e,f,g}
...
<{a, d}>
<{a}, {d}>
...
<{a}, {b}>
...
In = {b,c,e,f,g}
Sn = {b,c,e,f,g}
...
<{a}, {c}>
In = {b,c,e,f,g}
Sn = {b,c,e,f,g}
...
...

SPAM

SPAM(*SDB*, *minsup*)
1. Scan *SDB* to create *V*(*SDB*) and identify F_1, the list of frequent items.
2. **FOR** each item s $\in F_1$,
3. **SEARCH**($\langle s \rangle$, F_1, $\{e \in F_1 \mid e \succ_{lex} s\}$, *minsup*).

SEARCH(*pat*, S_n, I_n, *minsup*)
1. Output pattern *pat*.
2. $S_{temp} := I_{temp} := \emptyset$
3. **FOR** each item $j \in S_n$,
4. **IF** the s-extension of *pat* is frequent **THEN** $S_{temp} := S_{temp} \cup \{i\}$.
5. **FOR** each item $j \in S_{temp}$,
6. **SEARCH**(the s-extension of *pat* with *j*, S_{temp}, $\{e \in S_{temp} \mid e \succ_{lex} j\}$, *minsup*).
7. **FOR** each item $j \in I_n$,
8. **IF** the i-extension of *pat* is frequent **THEN** $I_{temp} := I_{temp} \cup \{i\}$.
9. **FOR** each item $j \in I_{temp}$,
10. **SEARCH**(i-extension of *pat* with *j*, S_{temp}, $\{e \in I_{temp} \mid e \succ_{lex} j\}$, *minsup*).

Fig. 3. The pseudocode of SPAM

SPADE

Generates candidates by merging patterns from the same equivalence class.

Example: <{a},{b}>, <{a},{c}>, <{a},{e}>

can be used to generate:

<{a},{b,c}>, <{a},{b,e}>, <{a},{c,e}>, <{a},{b},{c}>, <{a},{b},{e}>, <{a},{c},{b}>, <{a},{c},{e}>…

SPADE

SPADE(*SDB*, *minsup*)
1. Scan *SDB* to create $V(SDB)$ and identify F_1 the list of frequent items.
2. **ENUMERATE**(F_1).

ENUMERATE(an equivalence class F)
1. **FOR** each pattern $A_i \in F$
2. Output A_i
3. $T_i := \emptyset$.
4. **FOR** each pattern $A_j \in F$, with $j \geq i$
5. R = MergePatterns(A_i, A_j)
6. **FOR** each pattern $r \in R$
7. **IF** $sup(R) \geq minsup$ **THEN**
8. $T_i := T_i \cup \{R\}$;
9. **ENUMERATE**(T_i)

Fig. 4. The pseudocode of SPADE

Our proposal

- **Co-occurrence Map (CMAP):** a new structure to store co-occurrence information.
- **Pruning mechanisms for vertical algorithms**:
  - SPAM/ClaSP
  - SPADE
  - …

CMAP definition

- A structure **$CMAP_i$** stores every items that succeeds each item by **i-extension** at least *minsup* times.
- A similar structure **$CMAP_s$** stores every items that succeeds each item by **s-extension** at least *minsup* times.

$CMAP_i$	
item	is succeeded by (i-extension)
a	$\{b\}$
b	$\emptyset$
c	$\emptyset$
e	$\emptyset$
f	$\{g\}$
g	$\emptyset$

$CMAP_s$	
item	is succeeded by (s-extension)
a	$\{b, c, e, f\}$
b	$\{e, f, g\}$
c	$\{e, f\}$
e	$\emptyset$
f	$\{e, g\}$
g	$\emptyset$

This figure shows $CMAP_i$ and $CMAP_s$ for **minsup = 2**

Pruning properties

- **Pruning an i-extension:** The i-extension of a pattern ***p*** with an item ***x*** is infrequent if there exist an item ***i*** in the last itemset of *p* such that **(i,x)** is not in $\mathbf{CMAP_i}$.
- **Pruning an s-extension:** The s-extension of a pattern ***p*** with an item ***x*** is infrequent if there exist an item ***i*** in *p* such that **(i,x)** is not in $\mathbf{CMAP_s}$.

Pruning properties (cont'd)

- The previous properties can be generalized.
- **Pruning a prefix:**
  - Let ***p*** be a pattern.
  - If an **s-extension** of ***p*** with an item ***x*** is pruned, then no patterns having ***p*** as prefix and containing ***x*** can be frequent.
  - If an **i-extension** of ***p*** with an item ***x*** is pruned, then no i-extensions of ***p*** containing ***x*** can be frequent.

Integration in SPADE/SPAM/ClaSP

- **CM-SPADE**
  - for each candidate, check pruning (i-extension or s- extension).
  - **Note**: only necessary to check for the two last items in CMAPs.
- **CM-SPAM / CM-Clasp**
  - check each candidate for pruning (i-extension or s- extensions).
  - can also perform prefix pruning

CMAP implementation

- **n** items

- matrix implementation
  - size : n x n
- hasmap implementation
  - store only pairs of items that co-occurs
  - generally much smaller because few items co-occurs in most datasets

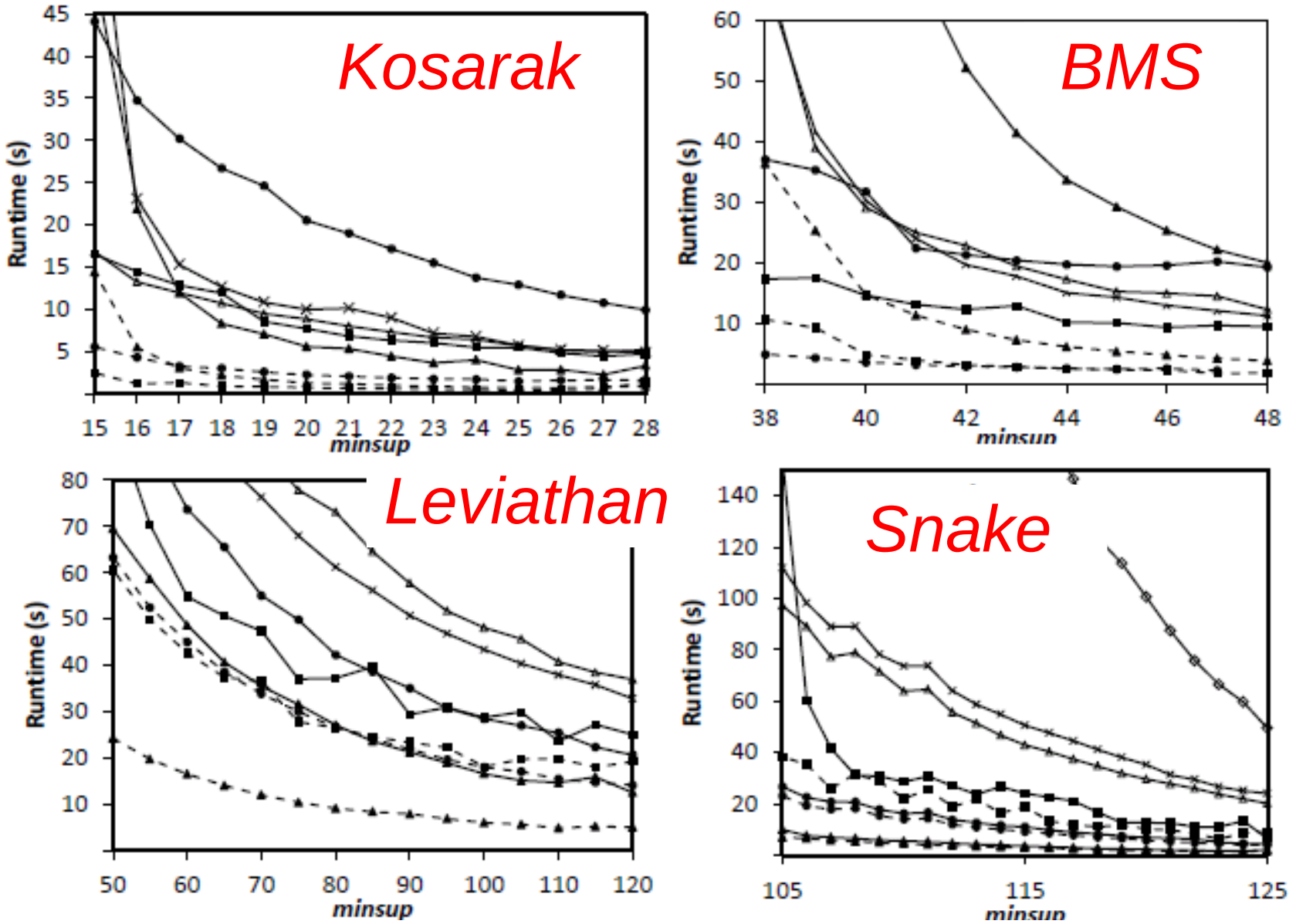
Experimental Evaluation

Datasets' characterictics

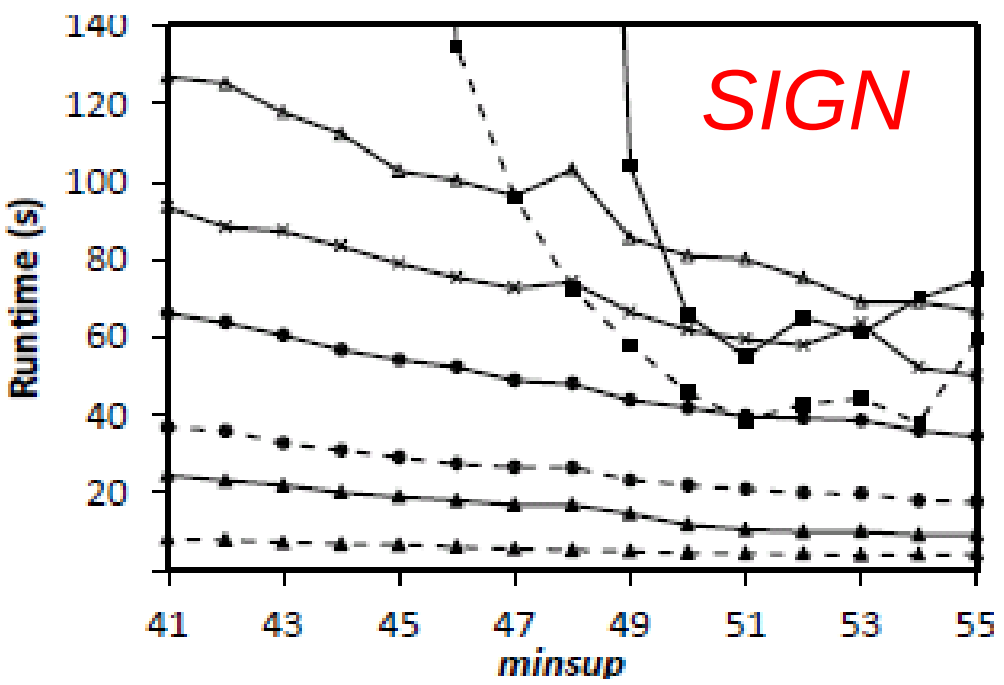
dataset	sequence count	distinct item count	avg. seq. length (items)	type of data
Leviathan	5834	9025	33.81 (std= 18.6)	book
Sign	730	267	51.99 (std = 12.3)	language utterances
Snake	163	20	60 (std = 0.59)	protein sequences
FIFA	20450	2990	34.74 (std = 24.08)	web click stream
BMS	59601	497	2.51 (std = 4.85)	web click stream
Kosarak10k	10000	10094	8.14 (std = 22)	web click stream

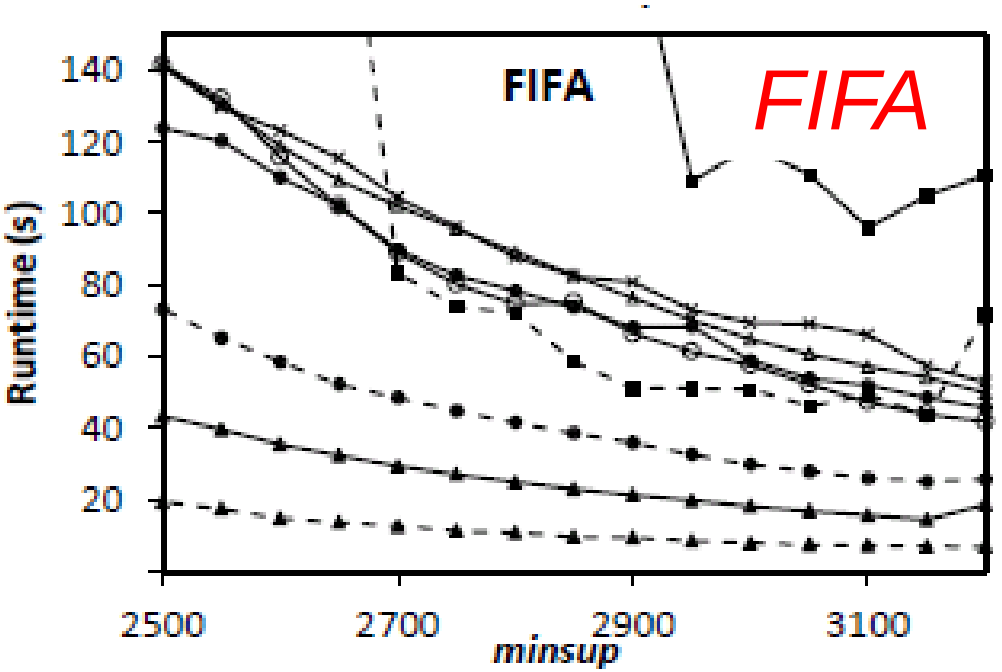
- CM-SPADE, CM-SPAM, CM-Clasp,
  vs SPADE, SPAM, ClaSP, PrefixSpan, GSP, CloSpan
- All algorithms implemented in Java
- Windows 7, 5 GB of RAM

Execution times



Execution times (cont'd)





Observations:

- **CM-SPADE:** best performance except for **Kosarak** and **BMS**
- **CM-SPAM:** best performance on **Kosarak**, **BMS**
- **CM-ClaSP:** best performance for closed pattern mining on all datasets except **FIFA** and **SIGN** where **CloSpan** performs better for low minsup values.

Candidate reduction (%)

	BMS	Kosarak	Leviathan	Snake	Sign	Fifa
CM-SPAM	78 to 93 %	94 to 98 %	50 to 51 %	28%	63 %	61 to 68 %
CM-SPADE	75 to 76 %	98 %	50 %	25 to 26 %	69 %	63 to 69 %
CM-ClaSP	79 to 93%	75 %	50 to 52 %	18 %	63 %	67 to 68 %

- **Number of candidates pruned**: 50% to 98 % for all but the Snake dataset.
- Reason: Snake is very dense.
- The number of pruned candidates decreases when *minsup* is set lower.

Memory overhead (MB / entries)

	BMS	Kosarak	Leviathan	Snake	Sign	Fifa
minsup	38	16	60	105	43	2500
CMAP Size (hashmap)	**0.5 MB**	**33.1 MB**	**15 MB**	64 KB	3.19 MB	**0.4 MB**
CMAP Size (matrix)	0.9 MB	388 MB	310 MB	**1.7 KB**	**0.2 MB**	34.1 MB
Pair count (hashmap)	50,885	58,772	41,677	144	17,887	2,500
Pair count (matrix)	247,009	101,888,836	81,450,625	400	71,289	8,940,100

- **matrix implementation**:
  - smallest overhead for: **Snake, Sign**
- **hashmap implementation**:
  - smallest overhead usage for : **BMS, Leviathan, Kosarak, FIFA**
  - always less than 35 MB

Conclusion

- Candidate pruning is important for vertical sequential pattern mining algorithms.
- Our proposal:
  - pruning based on item co-occurrences
  - a structure **CMAP** and pruning properties
  - integration in three algorithms: **SPAM, SPADE, ClaSP**
  - resulting algorithms outperforms state-of-the-art algorithms for (closed) sequential pattern mining.
- Source code and datasets available as part of the **SPMF data mining library (**GPL 3).

Open source Java data mining software, 66 algorithms
http://www.phillippe-fournier-viger.com/spmf/

Thank you. Questions?

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

This work has been funded by an NSERC grant from the government of Canada.

Applications of SPMF

- Web usage mining
- Stream mining
- Optimizing join indexes in data warehouses
- E-learning
- Smartphone usage log mining
- Opinion mining on the web
- Insider thread detection on the cloud
- Classifying edits on Wikipedia
- Linguistics
- Library recommendation,
- restaurant recommendation,
- web page recommendation
- Analyzing DOS attack in network data
- Anomaly detection in medical treatment
- Text retrieval
- Predicting location in social networks
- Manufacturing simulations
- Retail sale forecasting
- Mining source code
- Forecasting crime incidents
- Analyzing medical pathways
- Intelligent and cognitive agents
- Chemistry

Fournier-Viger, P., Gomariz, A., Campos, M., Thomas, R. (2014). Fast Vertical Sequential Pattern Mining Using Co-occurrence Information. Proc. 18th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD 2014) Part 1, Springer, LNAI, 8443. pp. 40-52.