Fast Vertical Mining of Sequential Patterns Using Co-occurrence Information

Philippe Fournier-Viger[1]
Antonio Gomariz[2]
Manuel Campos[2]
Rincy Thomas[3]

[1]Shenzhen University, China
[2]University of Murcia, Spain
[3]SCT, India

Most influential paper award 2024

May 7-10 2024 PAKDD

Philippe Fournier-Viger,
Distinguished professor @ Big Data Center @ Shenzhen University

- Canadian researcher, National talent in China
- Stanford list of worldwide highly cited researchers (top 2%)
- Elsevier Highly cited Chinese researcher (> 14,500 citations)
- **Research**: **data mining, pattern mining, applications**

Shenzhen University

Thanks to PAKDD

- The steering committee, award committee, and organizers,
- Wonderful organization!
- **PAKDD**: A great community in pacific-asia and internationally.

- **Attended PAKDD**:
  2014 (Taiwan),
  2015 (Vietnam),
  2017 (Korea),
  2018 (Australia),
  2019 (Macau),
  2020 (Singapore/virtual)
  2024 (Taiwan)

The paper

Fournier-Viger, P., Gomariz, A., Campos, M., Thomas, R. (2014). **Fast Vertical Sequential Pattern Mining Using Co-occurrence Information**. Proc. PAKDD 2014, Springer, LNAI, 8443. pp. 40-52.

- Sequential pattern mining: The task of finding subsequences that appear frequently in some sequences.

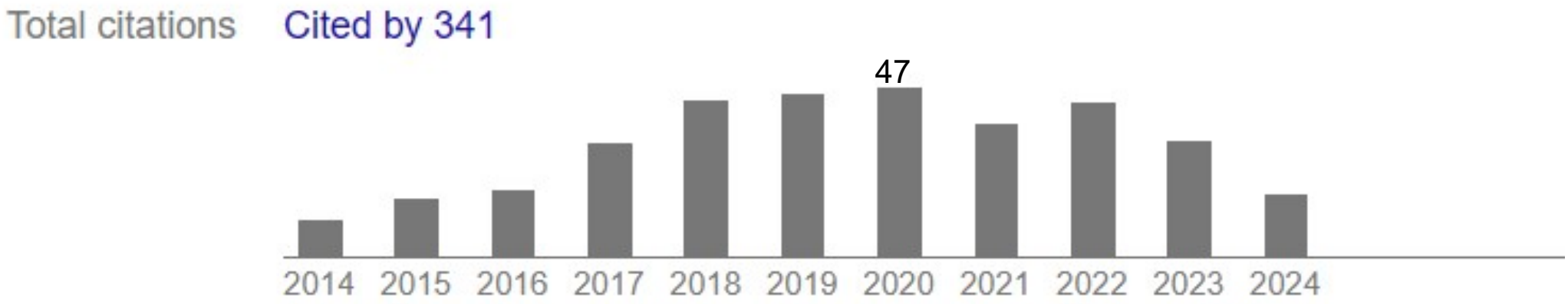
  .... **AAGA**ATGAGTCC ...

  .... ATGAG**AAGA**TCC ...

- Contribution: A new optimization to speed-up **sequential pattern mining algorithms.**

The paper

Fournier-Viger, P., Gomariz, A., Campos, M., Thomas, R. (2014). **Fast Vertical Sequential Pattern Mining Using Co-occurrence Information**. Proc. PAKDD 2014, Springer, LNAI, 8443. pp. 40-52.



Cited for: applications of our algorithms
applications of our optimiziation in other algorithms

How it started?

How it started?

- I was working on **pattern mining**.
- Developed new algorithms.
- I also implemented many classic algorithms: PrefixSpan, SPAM, BIDE+, etc.
- The code is public in my **open-source SPMF software**.

How it started?

- **Rincy Thomas, Ph.D student from India** e-mailed me: Which algorithm is the best? why?
- He eagerly **wanted** to find the answer!
- We had datasets.
- But we did **not** have the code of **all** the state-of-the art algorithms.

How it started?

- **Antonio Gomariz, Ph.D student from Spain** published a paper at PAKDD 2013.
  A. Gomariz, M. Campos, R. Marín and B. Goethals (2013), ClaSP: An Efficient Algorithm for Mining Frequent Closed Sequences.

- He shared his code of many algorithms for SPMF:
  GSP, SPADE, BitSPADE, SPAM, PrefixSpan, CloSpan, ClaSP

How it started?

- **Rincy, Ph.D, did experiments, and made some initial observations**:
  SPAM generates too many candidates.
  The join operation is very costly.
  If we could find a way to reduce that number, we might speed-up the algorithm...

But how?

--> A new optimization (CMAP)

10 years ago... in Tainan city.

Introduction

- **Pattern mining**: to apply algorithms to discover useful patterns in data.

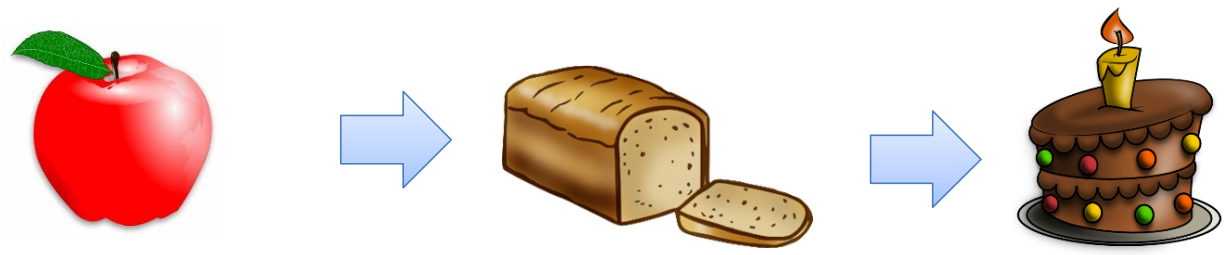
- **Many types of data** can be analyzed:
  graphs, relational databases, time series, sequences, etc.

- In this presentation, we focus on analyzing a common type of data called **discrete sequences** to find interesting patterns in it.

What is a discrete sequence?

A **sequence** is an ordered list of symbols.

Example 1: a sequence can be the items that are purchased by a customer over time:

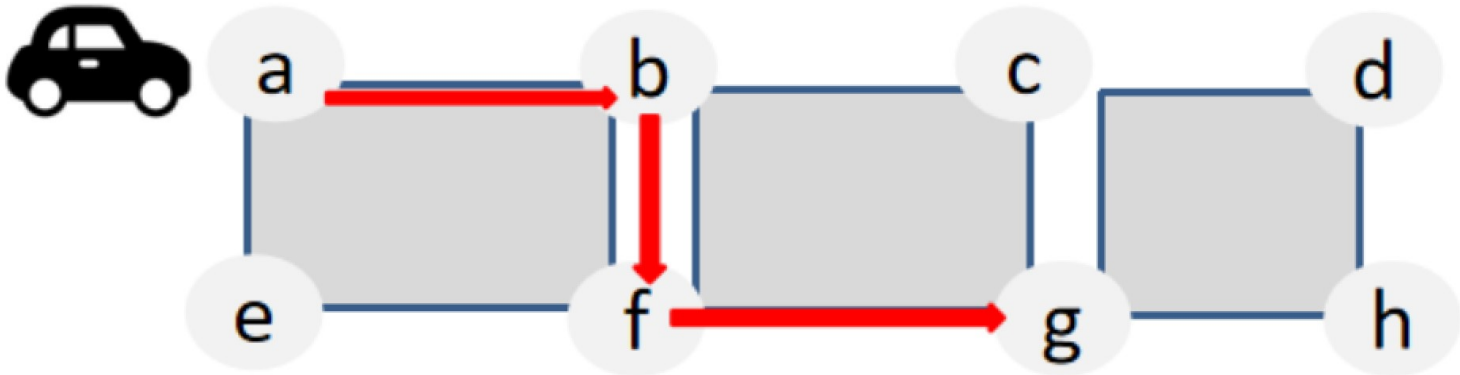


What is a discrete sequence?

Example 2: a sequence of words in a sentence:

I → **go** → **back** → **home**

Example 3: a list of locations visited by a car



Introduction

Sequential pattern mining:

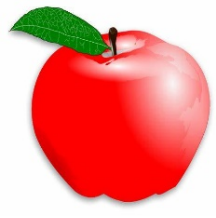
- finding frequent subsequences in a **sequence database**.

Preliminary definitions -->

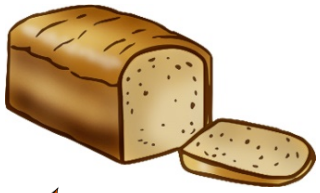
Definition: Items

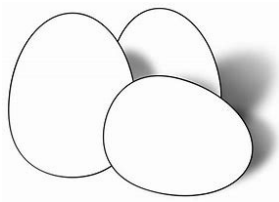
Let there be a **set** of **items** (symbols) called I.

Example: $I = \{a, b, c, d, e\}$

a = apple 

d = dattes

b = bread 

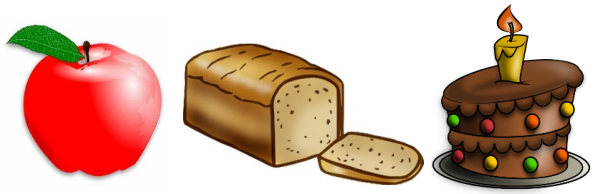
e = eggs 

c = cake

Definition: Itemset

An itemset is a set of **items**.

Example: $\{a, b, c\}$ is an itemset containing 3 items



$\{d, e\}$ is an itemset containing 2 items

- Note: an itemset cannot contain a same item twice.
- An itemset having k items is called a *k-itemset*.

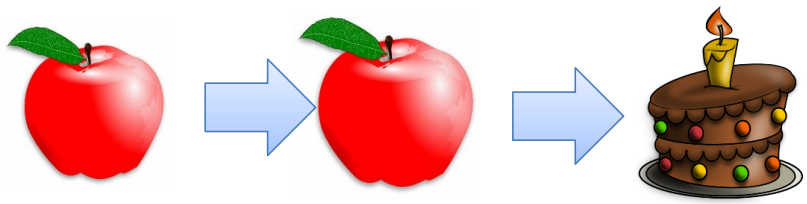
Definition: Sequence

A **discrete sequence** S is a an ordered list of itemsets $S = \langle X_1, X_2, \ldots, X_n \rangle$.

Example 1: $\langle \{a, b\}, \{c\} \rangle$ is a sequence containing two itemsets.

It means that a customer purchased $apple$ and $bread$ at the same time and then purchased $cake$.

Example 2: $\langle \{a\}, \{a\}, \{c\} \rangle$



Example

INPUT:

Sequence database

$S_1 =$	$\langle\{a,b\},\{c\},\ \{a\}\rangle$
$S_2 =$	$\langle\{a,b\},\{b\},\ \{c\}\rangle$
$S_3 =$	$\langle\{b\},\ \{c\},\{d\}\rangle$
$S_4 =$	$\langle\{b\},\ \{a,b\},\{c\}\rangle$

$minsup = 3$

OUTPUT:

all sequential patterns:

$\langle\{a\}\rangle$ support = 3
$\langle\{b\}\rangle$ support = 4
$\langle\{c\}\rangle$ support = 4
$\langle\{a\},\ \{c\}\rangle$ support = 3
$\langle\{b\},\ \{c\}\rangle$ support = 4
$\langle\{a,b\},\ \{c\}\rangle$ support = 3

Example

INPUT:

Sequence database

$S_1 =$	$\langle\{a,b\},\{c\},\{a\}\rangle$
$S_2 =$	$\langle\{a,b\},\{b\},\{c\}\rangle$
$S_3 =$	$\langle\{b\},\{c\},\{d\}\rangle$
$S_4 =$	$\langle\{b\},\{a,b\},\{c\}\rangle$

$minsup = 3$

OUTPUT:

all sequential patterns:

$\langle\{a\}\rangle$ support = 3

$\langle\{b\}\rangle$ support = 4

$\langle\{c\}\rangle$ support = 4

$\langle\{a\},\{c\}\rangle$ support = 3

$\langle\{b\},\{c\}\rangle$ support = 4

$\langle\{a,b\},\{c\}\rangle$ support = 3

A difficult problem!

- **A naïve algorithm:** read the database and count the support of **all possible patterns.**
- **Inefficient** because there can be **a huge number of sequential patterns.**
- **For example:**

  $\langle\{a\}\rangle,\ \ \langle\{b\}\rangle,\ \ \langle\{c\}\rangle\ \ldots.$

  $\ldots.\ \langle\{a,b\}\rangle, \langle\{a,c\}\rangle, \langle\{a,d\}\rangle\ldots$

  $\ldots\ \langle\{a\},\{a\}\rangle,\ldots$

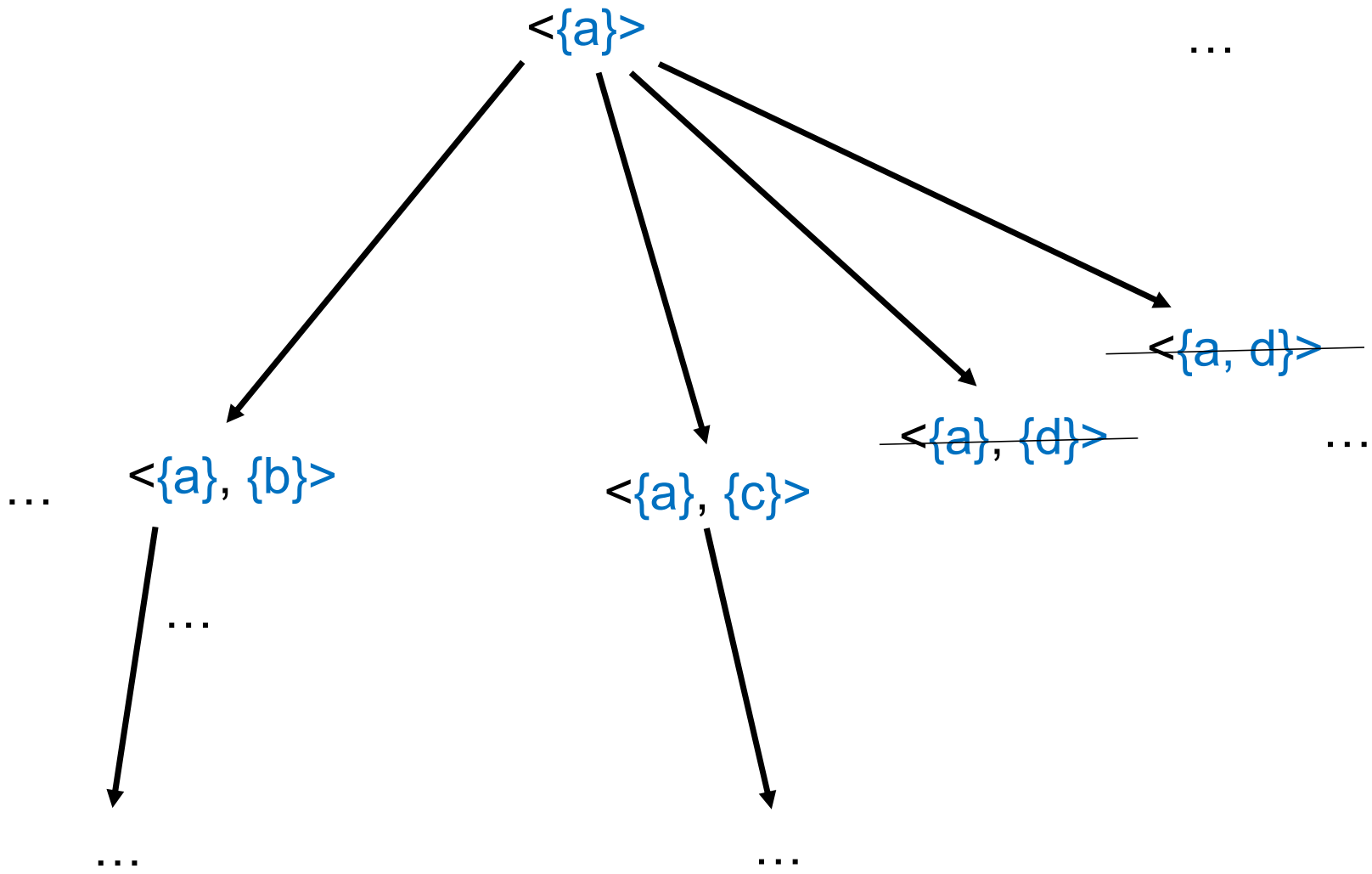
  $\ldots\ \left\langle\{a\},\{a\},\{a\},\langle\{a\},\{a\},\{a\},\{a\}\rangle\right\rangle\ldots.\ \langle\{a,b\}\{a\}\rangle,\ldots.\ \langle\{a\},\{b\}\{a\}\rangle,\ldots.$

- **An efficient algorithm** must find the frequent sequential patterns, without checking all the possibilities.
- **First algorithm**: GSP (1994)

Pattern-Growth algorithms

- FreeSpan (2000), PrefixSpan (2001), CloSpan (2003), BIDE+ (2007)…
  - Start from frequent items.
  - For each frequent pattern, perform database projection and count the frequency of items that could extend the pattern.
- **Strong point:** Do not generate candidates.
- **Drawback:** Database projections can be quite costly.

PrefixSpan



Vertical algorithms

- SPAM, SPADE, bitSPADE, ClaSP, VMSP
  - First, convert the database 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** to create new patterns.

Extensions

- The **i-extension** of $\langle\{\mathbf{a}\}\rangle$ with **c** is:

$$\langle\{\mathbf{a}, \mathbf{c}\}\rangle$$

- The **s-extension** of $\langle\{\mathbf{a}\}\rangle$ with **c** is:

$$\langle\{\mathbf{a}\}, \{\mathbf{c}\}\rangle$$

Vertical algorithms (cont'd)

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

a	
SID	Itemsets
1	1
2	1,4
3	1

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

$\langle\{a\}, \{b\}\rangle$

SID	Itemsets
1	1
2	1, 3
3	2

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

Our proposal

- **Co-occurrence Map (CMAP):** a new structure to store co-occurrence information.
- A **pruning mechanism** to **speed-up 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	$\emptyset$

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

- **full matrix** implementation
  - size : n x n where n is the number of items
- **sparse matrix** implementation
  - store only pairs of items that co-occurs in a hashmap
  - generally much smaller because few items co-occurs in most datasets

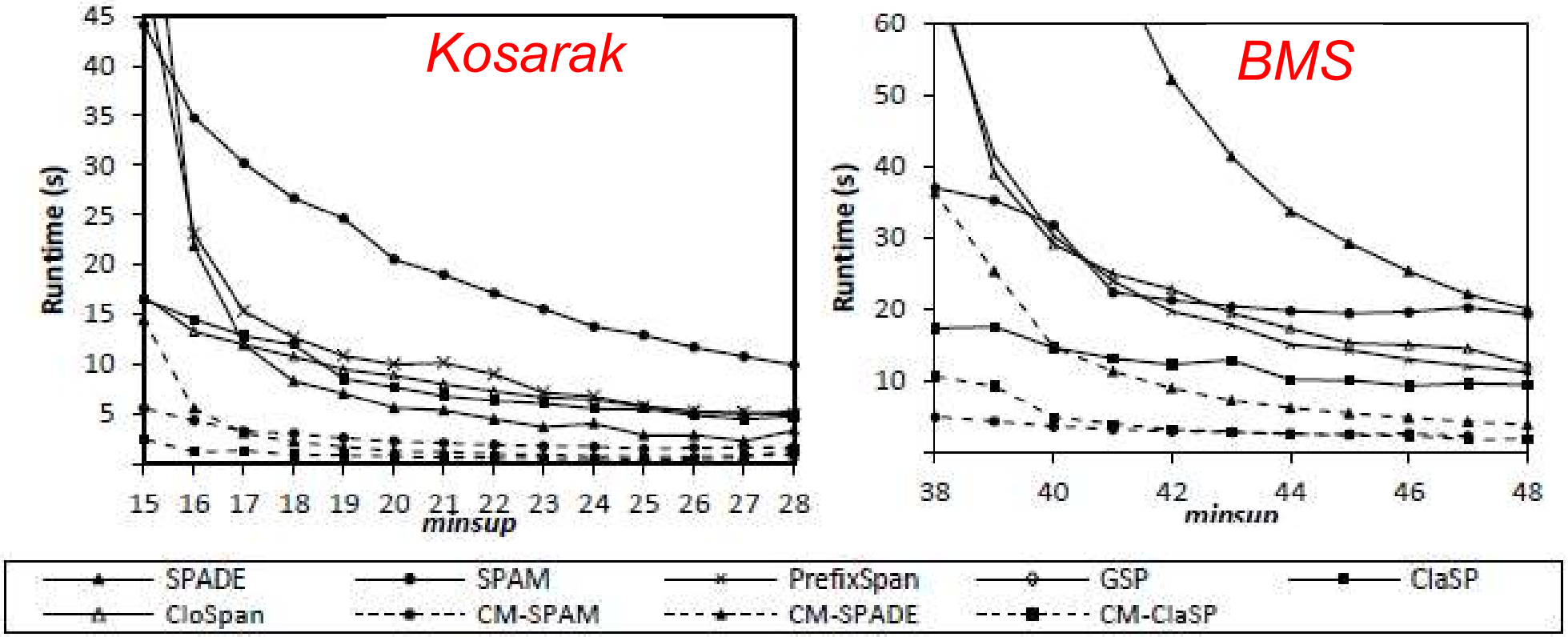
Experimental Evaluation

Datasets' characterictics

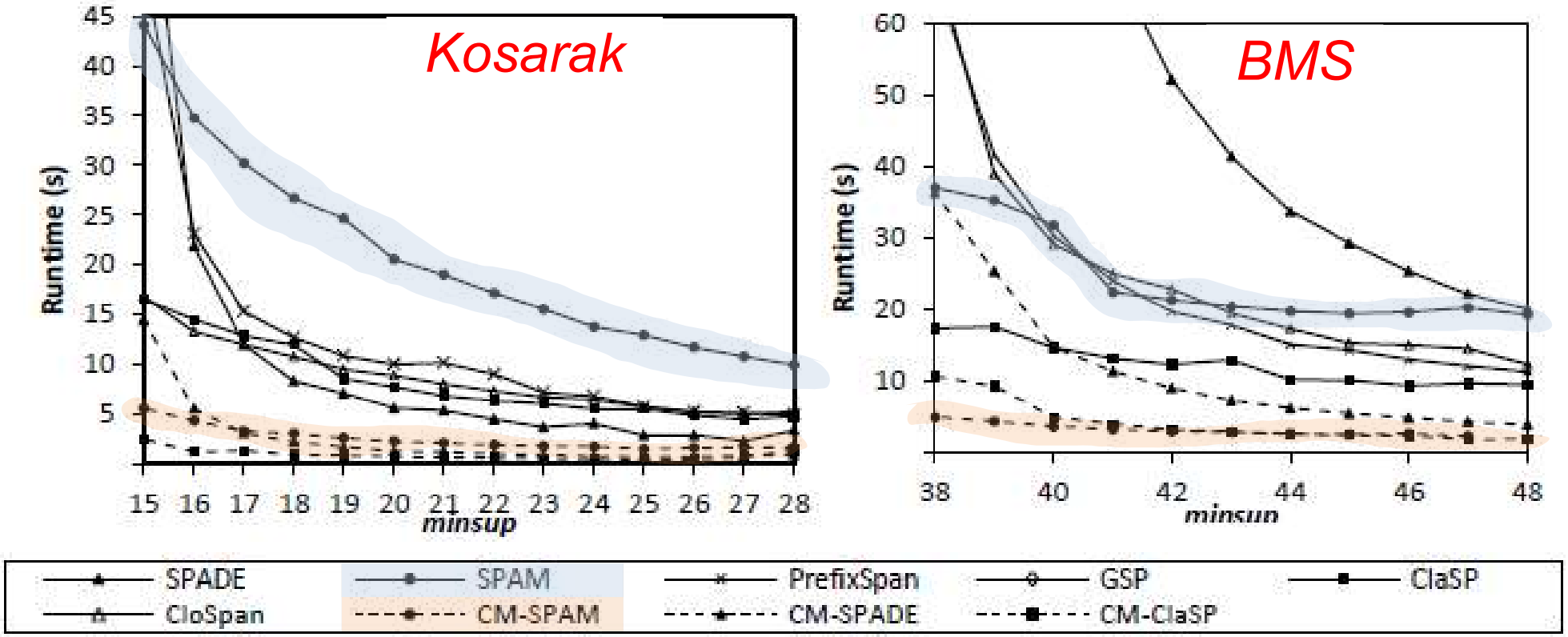
dataset	sequence count	distinct item count	avg. seq. length (items)	type of data
Leviathan	5834	9025	33.81 (std= 18.6)	book
Sign	730	267	51.99 (std = 12.3)	language utterances
Snake	163	20	60 (std = 0.59)	protein sequences
FIFA	20450	2990	34.74 (std = 24.08)	web click stream
BMS	59601	497	2.51 (std = 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
- Java, Windows 7, 5 GB of RAM

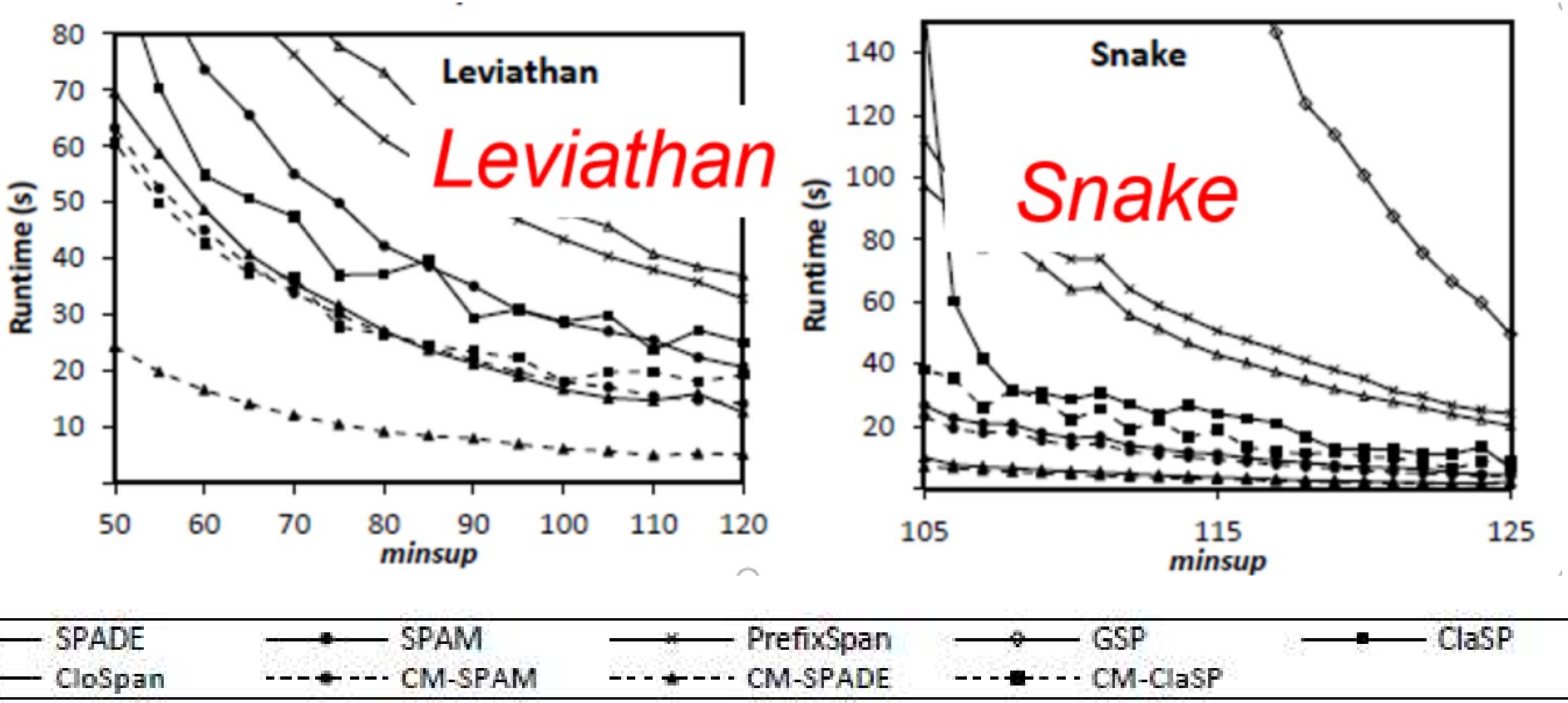
Execution times



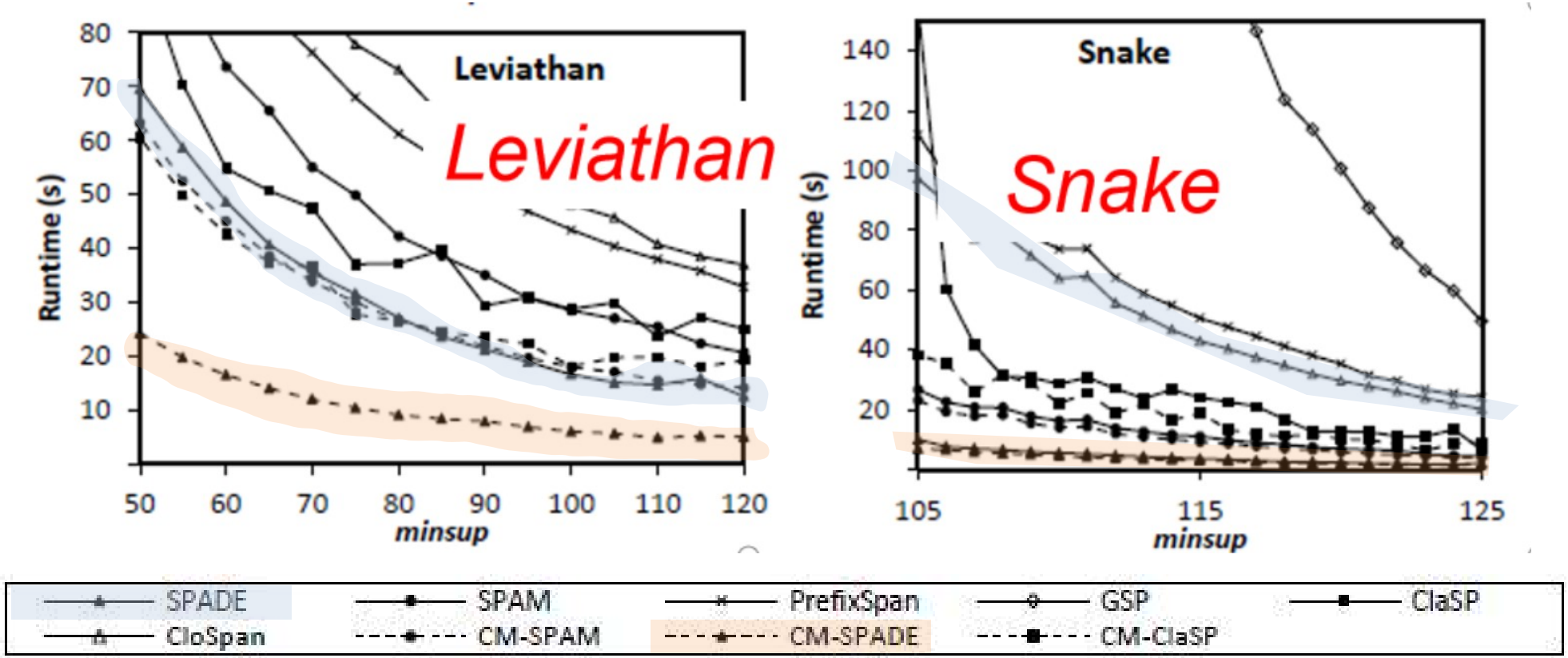
Execution times



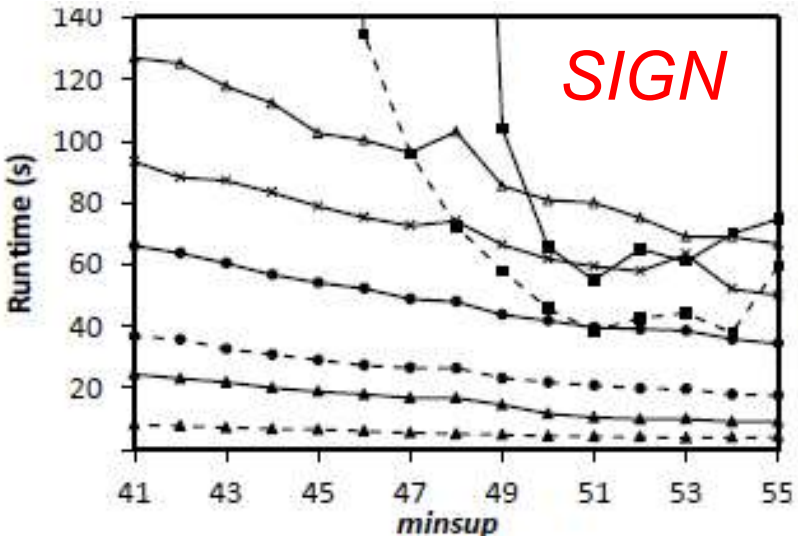
Execution times

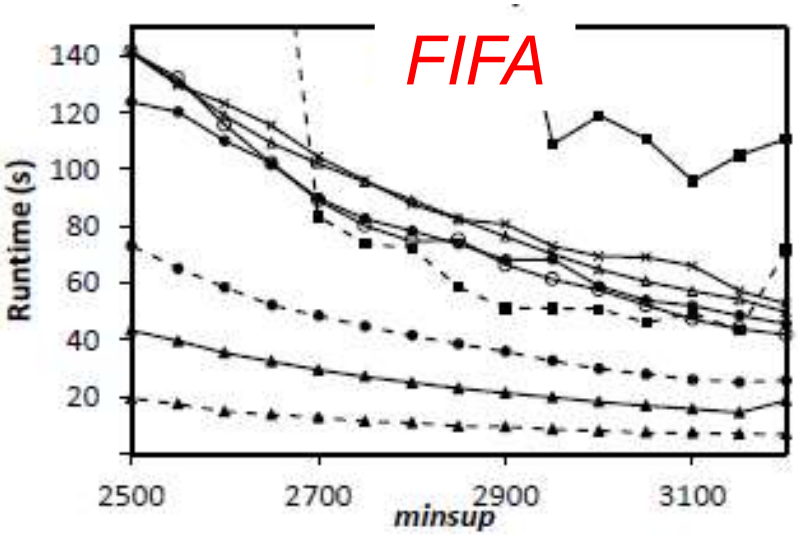


Execution times



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

Some applications of CM-SPAM

- **Malware classification:**
  Nawaz, M. S., Fournier-Viger, P., Nawaz, M. Z., Chen, G., Wu, Y. (2022) MalSPM: Metamorphic Malware Behavior Analysis and Classification using Sequential Pattern Mining. Computers & Security, Elsever
- **Protein structures analysis and classification:**
  Nawaz, S. M., Fournier-Viger, P., He, Y., Zhang, Q. (2023). PSAC-PDB: Analysis and Classification of Protein Structures. Computers in Biology and Medicine, Elsevier, 158: 106814 (2023)
- **E-learning**
- **...**

SPMF

An Open-Source Data Mining Library

Introduction
Algorithms
Download
Documentation
Datasets
FAQ
License
Contributors
Citations
Performance
Developers's guide
Videos
Other resources

Introduction

SPMF is an **open-source software** and **data mining library** written in **Java**, specialized in **patte**
patterns in data) .

It is distributed under the **GPL v3 license**.

It offers implementations of **251 data mining algorithms** for:

- **association rule mining,**
- **itemset mining,**
- **sequential pattern**
- **sequential rule mining,**
- **sequence prediction**,
- **periodic pattern mining,**
- **episode mining**
- **high-utility pattern mining,**
- **time-series mining.**
- **clustering** and **classification**,

SPMF can be used as a **standalone program** with a simple user interface or from the **command line**.

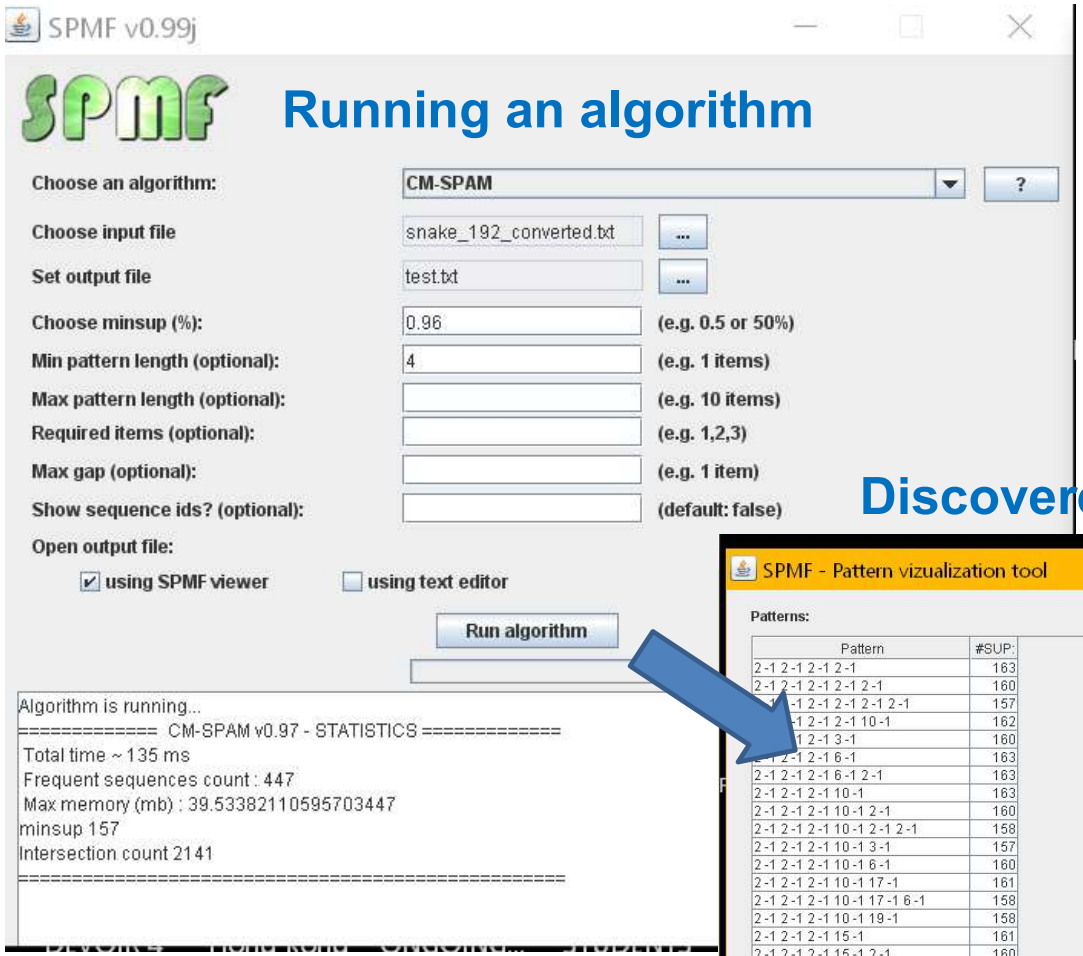
Moreover, the **source code** of each algorithm can be easily integrated in other **Java** software.

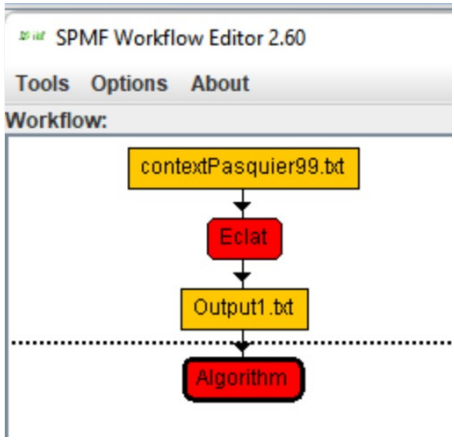
Besides, some unofficial wrappers are available for other languages such as Python, R and Weka.

SPMF is fast and lightweight (no dependencies to other libraries).

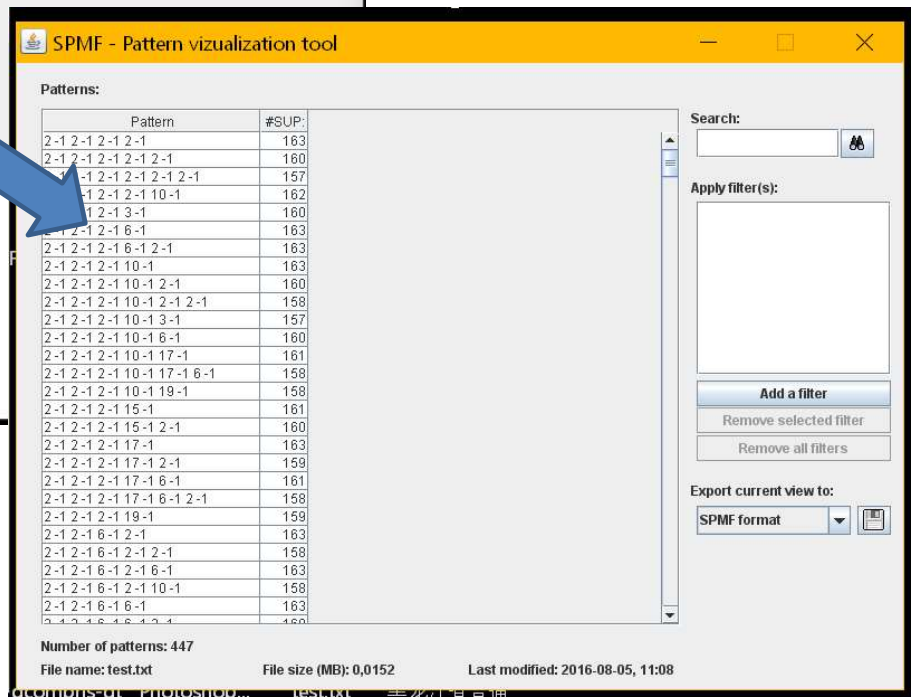
The current **version** is **v2.58** and was released the **25th November 2022.**

Running an algorithm





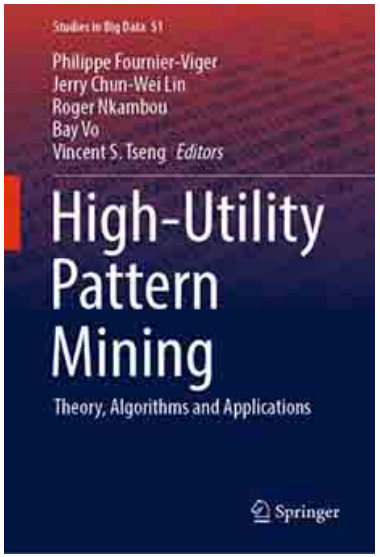
Discovered patterns

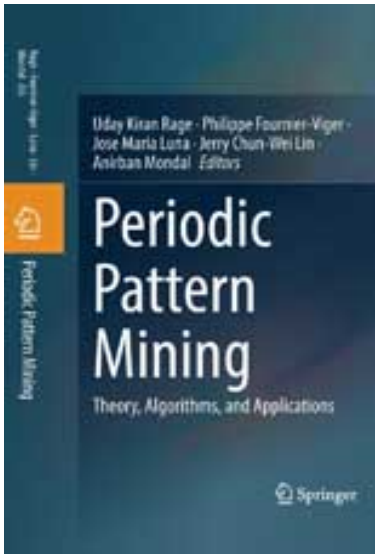


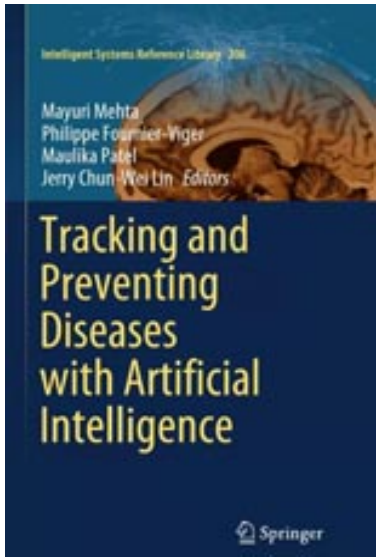
Applications of SPMF

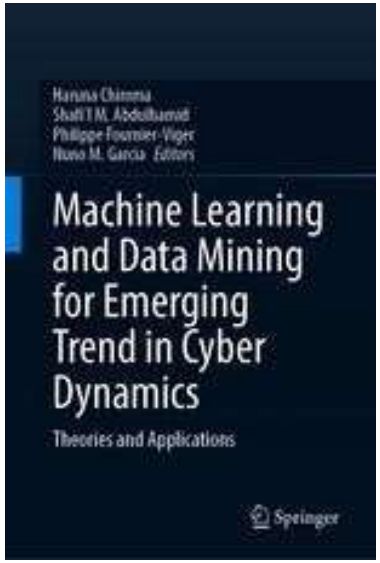
- Web usage mining
- E-learning
- Stream mining
- Library recommendation,
- Predicting location in social networks
- restaurant recommendation,
- Classifying edits on Wikipedia
- Web page recommendation
- Insider thread detection on the cloud
- Linguistics
- Analyzing DOS attack in network data
- Anomaly detection in medical treatment
- Discovery of Antigen patterns
- Load forecasting
- Agricultural machinery maintenance
- Authorship attribution
- Mnufacturing simulations
- Retail sale forecasting
- Mining source code
- Forecasting crime incidents
- Analyzing medical pathways
- Optimizing join indexes in data ware
- Smartphone usage log mining
- Opinion mining on the web
- Intelligent and cognitive agents
- Reducing energy consumption
- Music Analysis
- Chemistry
- Text retrieval
- Train journey prediction
- Fault detection in execution traces
- ….

Edited books









Workshop

UDML 2024 @ PAKDD 2024
Special issue in **Expert Systems**

Trends and opportunities

- Finding patterns in complex data,
  e.g. graph, sequence, with time
- Finding patterns that are statistically significant,
- Finding patterns in data streams,
- Designing parameter-free algorithms....
- Bridging the gap with machine learning
  – patterns as features, to create embeddings, ...

Conclusion:

How to obtain this award?

- Choose an important topic,
- A novel idea,
- Clear improvement over the state-of-the-art,
- Write the paper well,
- Publish the source code and datasets,
- Promote the paper in different ways.
- Have some luck!

Who will get it in **2034**... ? :-)

Thank you. Questions?

E-mail:
philfv@qq.com
Wechat:
philfv

Open source Java data mining software, 280 algorithms
http://www.phillippe-fournier-viger.com/spmf/

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, 280 algorithms
http://www.phillippe-fournier-viger.com/spmf/