TKS: Efficient Mining of Top-K Sequential Patterns

Philippe Fournier-Viger[1],
Antonio Gomariz[2], Ted Gueniche[1],
Espérance Mwamikazi[1], Rincy Thomas[3]

[1]University of Moncton, Canada

[2]University of Murcia, Spain

[3] Sha-Shib College of Technology, India

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*},{*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)

How to choose *minsup* the threshold?

- **How ?**
  - too high, too few results
  - too low, too many results,
    performance often exponentially degrades
- **In real-life:**
  - time/storage limitation,
  - the user cannot analyze too many patterns,
  - fine tuning parameters is time-consuming (depends on the dataset)

A solution

- Redefining the **problem of sequential pattern mining** as **mining the top-*k* sequential patterns**.
- **Input:**
  - *k* is the number of patterns to be generated.
- **Output:**
  - the *k* most frequent patterns

Challenges

- An algorithm for top-k sequential pattern mining cannot use a fixed *minsup* threshold to prune the search space.
- Therefore, the problem is more difficult.
- Large search space

TSP

- **TSP** is the state-of-the art algorithm (Tsekov, Yan & Pei, KAIS 2005).
- Discovers **top-k sequential patterns** or top-k closed sequential patterns.
- Uses a **pattern-growth** approach based on **PrefixSpan** (Pei et al., 2001)
  - Scan database to find patterns containing single items.
  - Project database, scan projected databases and append items to grow patterns.
- **Could we make a more efficient algorithm?**

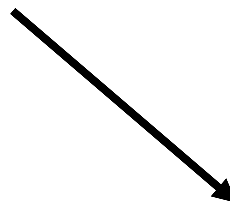
Our proposal

- A new algorithm named **TKS (*T*op-*K* *S*equential pattern miner)**
- It uses a:
  - a vertical representation of the database,
  - the SPAM search procedure to explore the search space of patterns,
  - several optimizations to increase efficiency

The SPAM search procedure

First, creates a vertical representation of the database (sid lists):

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	$\langle\{b\}, \{f, g\}\rangle$



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

The SPAM search procedure (2)

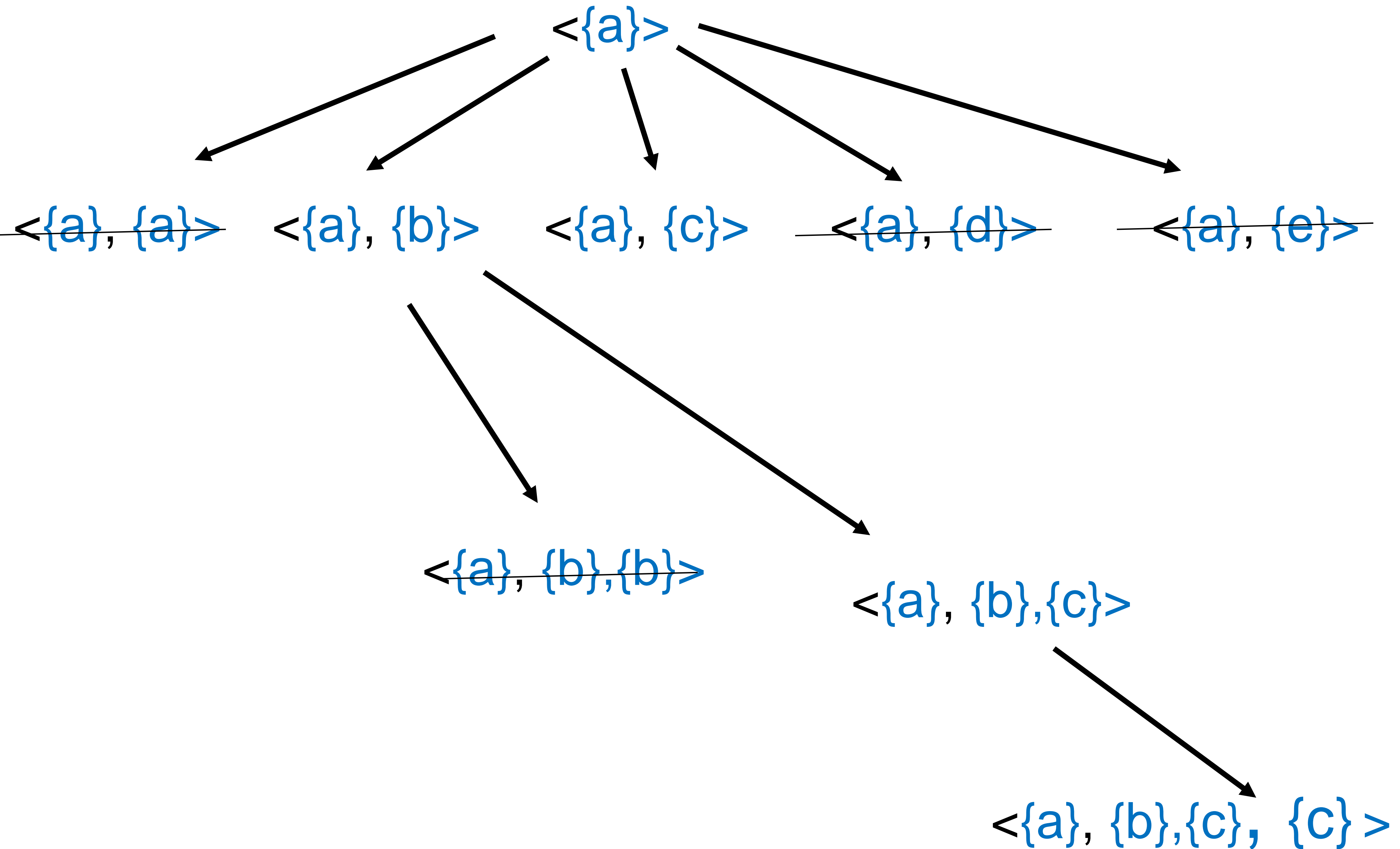
- Then, the algorithm identify frequent patterns containing a single item.
- Then, SPAM append items recursively to each frequent pattern to generate larger patterns.
  - s-extension: <I1, I2, I3… In> with {a} is <I1, I2, I3… In, {a}>
  - i-extension: <I1, I2, I3… In> with {a} is <I1, I2, I3… In U{a}>
- The support of a larger pattern is calculated by intersecting SID lists:

a	
SID	Itemsets
1	1
2	1,4
3	1

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

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

The SPAM search procedure (3)



TKS

Main idea

•set *minsup* = 0.

•use SPAM to explore the pattern search space

•keep a set *L* that contains the current top-*k* patterns found until now.

•when *k* patterns are found, raise *minsup* to the support of the least frequent pattern in *L*.

•after that, for each pattern added to *L*, raise the *minsup* threshold.

TKS (2)

- The resulting algorithm has poor execution time because the search space is too large.
- We therefore need to use additional strategies to improve efficiency.

TKS – Strategy 2

- **Observation:**
  - if we can find patterns having high support first, we can raise *minsup* more quickly to prune the search space.
- **Strategy**
  - We added a set *R* containing the *k* patterns having the highest support that can be used to generate more patterns.
  - The pattern having the highest support is always in this set is extended first.

TKS – choice of data structures (1)

- We found that the choice of data structures for implementing L and R is also very important:
  - *L* : fibonnaci heap : O(1) amortized time for insertion and minimum, and O(log(n)) for deletion.
  - *R*: red-black tree: O(log(n)) worst case time complexity for insertion, deletion, min and max.

TKS – Strategy 3
– discard newly infrequent items

- **Could we reduce the number of candidates?**
- When *minsup* is raised, items that become infrequent are recorded in a hash table.
- Before generating a candidate by appending an item to a pattern, the hash table is checked.
- If the item has become infrequent, the pattern is not generated.
- This avoid making the costly sid list intersection operation for infrequent patterns.

TKS – Strategy 4 – precedence pruning

- **Could we further reduce the number of candidates?**
- A new structure: **Precedence MAP (PMAP)**
  - indicates the number of times that each item follows each other item by **s-extension** and **i-extension**

item	pairs of type <item, support>
a	<a,1, s> <b,2, s>, <b, 2, i>, <c, 2, s>, <d,1, i>, <e,2, s>, <e,1, i>, <f,2, s>, <f,1, i>, <g,1, s>
b	<a,1,s>, <b,1,s>,<c, 1,s>, <e,2,s>, <e,1,i>, <f, 4,s>, <f, 1,i>, <g,2,s>
c	<a,1,s>, <b,1,s>, <e,2,s>, <f,2,s>, <g,1,s>
d	<a,1,s>, <b,1,s>,<c,1,s>, <e,1,s>, <f,1,s>
e	<f,1,i>
f	<e,2,s>, <g,2,i>
g	<e,1,s>

TKS – Strategy 4 – precedence pruning

- **Example:**
  - Consider a pattern <{a}, {b}> and an item c.
  - For minsup =2, <{a}, {b} , {c}> is not frequent

item	pairs of type <item, support>
a	<a,1, s> <b,2, s>, <b, 2, i>, <c, 2, s>, <d,1, i>, <e,2, s>, <e,1, i>, <f,2, s>, <f,1, i>, <g,1, s>
b	<a,1,s>, <b,1,s>,<c, 1,s>, <e,2,s>, <e,1,i>, <f, 4,s>, <f, 1,i>, <g,2,s>
c	<a,1,s>, <b,1,s>, <e,2,s>, <f,2,s>, <g,1,s>
d	<a,1,s>, <b,1,s>,<c,1,s>, <e,1,s>, <f,1,s>
e	<f,1,i>
f	<e,2,s>, <g,2,i>
g	<e,1,s>

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
Bible	36369	13905	21.64 (std = 12.2)	book
Sign	730	267	51.99 (std = 12.3)	sign language utterances
Snake	163	20	60 (std = 0.59)	protein sequences
FIFA	20450	2990	34.74 (std = 24.08)	web click stream

- TKS vs TSP
- All algorithms implemented in Java
- Windows 7, 1 GB of RAM

Experiment 1 – influence of *k*

Results for *k* =1000, 2000, 3000

Dataset	Algorithm	Execution Time (s)			Maximum Memory Usage (MB)		
		k=1000	*k*=2000	*k*=3000	*k*=1000	*k*=2000	*k*=3000
Leviathan	**TKS**	**10**	**23**	**38**	**302**	**424**	**569**
	TSP	103	191	569	663	856	974
Bible	**TKS**	**16**	**43**	**65**	**321**	**531**	**658**
	TSP	88	580	227	601	792	957
Sign	**TKS**	**0.5**	**0.8**	**1.2**	**46**	**92**	**134**
	TSP	4.8	7.6	9.1	353	368	383
Snake	**TKS**	**1.1**	**1.63**	**1.8**	**19**	**38**	**44**
	TSP	19	33	55	446	595	747
FIFA	**TKS**	**15**	**34**	**95**	**436**	**663**	**796**
	TSP	182	O.O.M.	O.O.M.	979	O.O.M.	O.O.M.

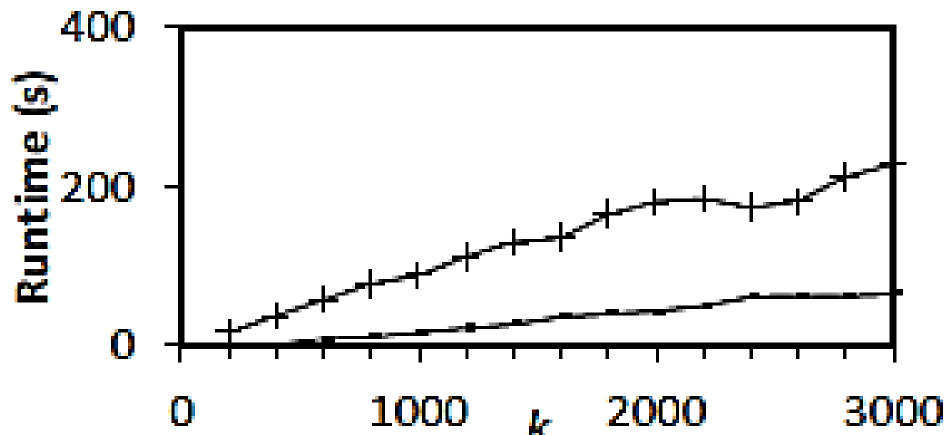
TKS: up to an order of magnitude faster

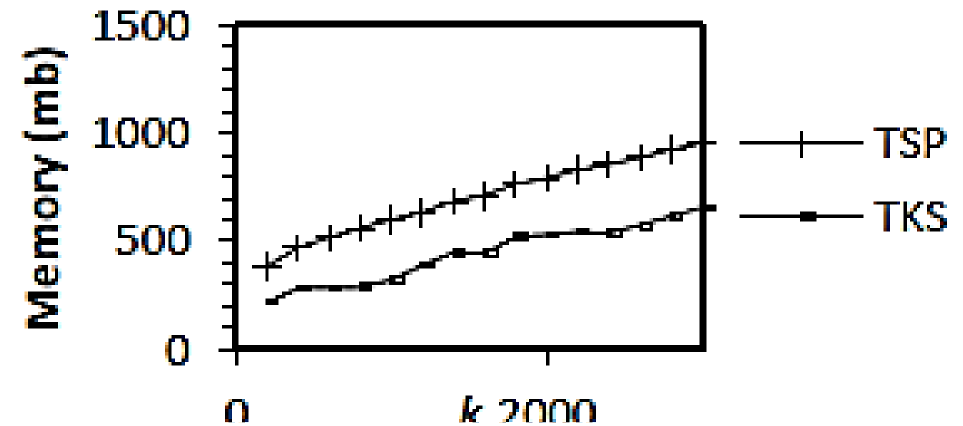
up to an order of magnitude less memory

For example, on **Snake**, TKS uses 13 times less memory and is 25 times faster

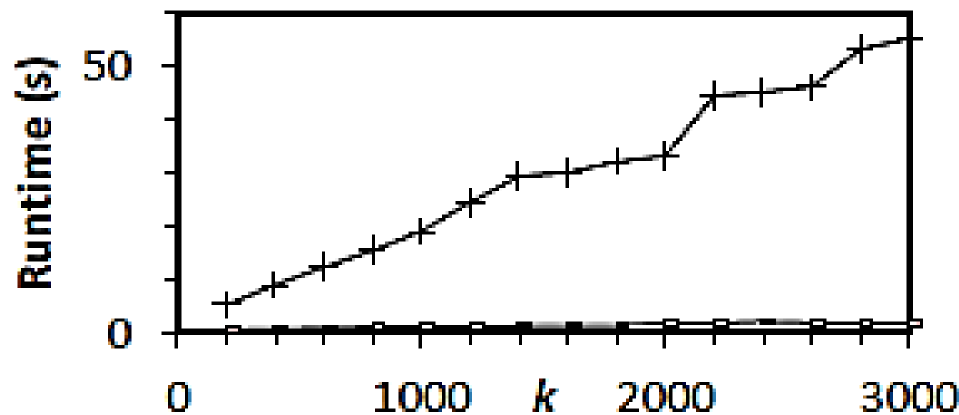
Experiment 1 – influence of *k*

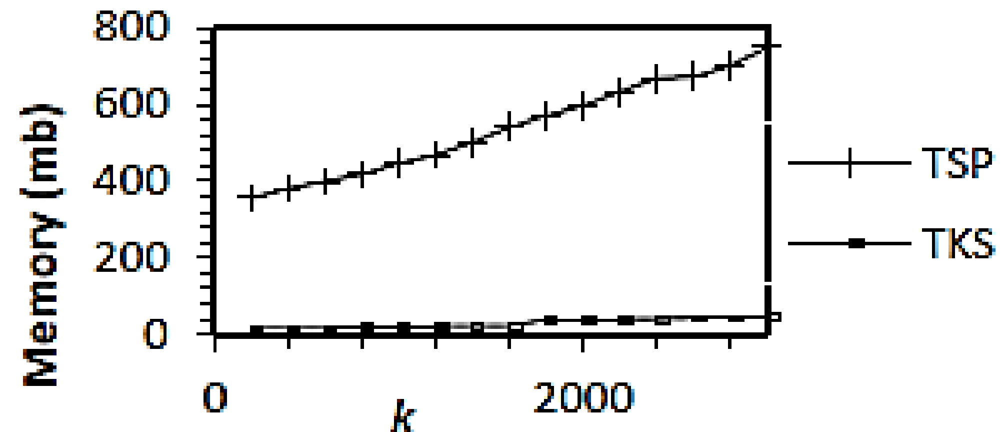
Bible





Snake





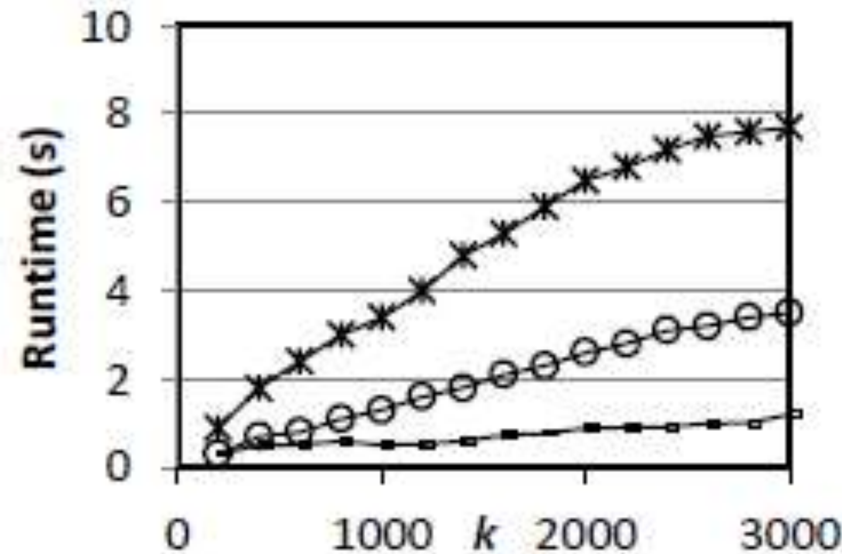
TKS has better scalability w.r.t *k*

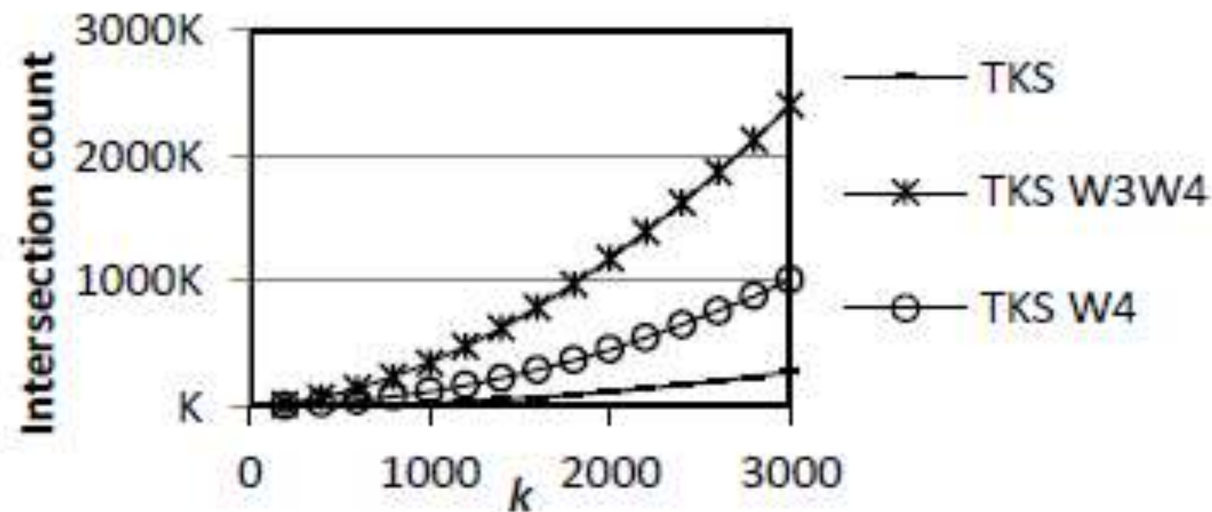
Experiment 2 – optimizations

Four versions of TKS:

- TKS
- TKS W2 (without exploring most promising patterns)
- TKS W3 (without discarding newly infrequent items)
- TKS W3W4 (without PMAP and discarding infrequent items)

Sign

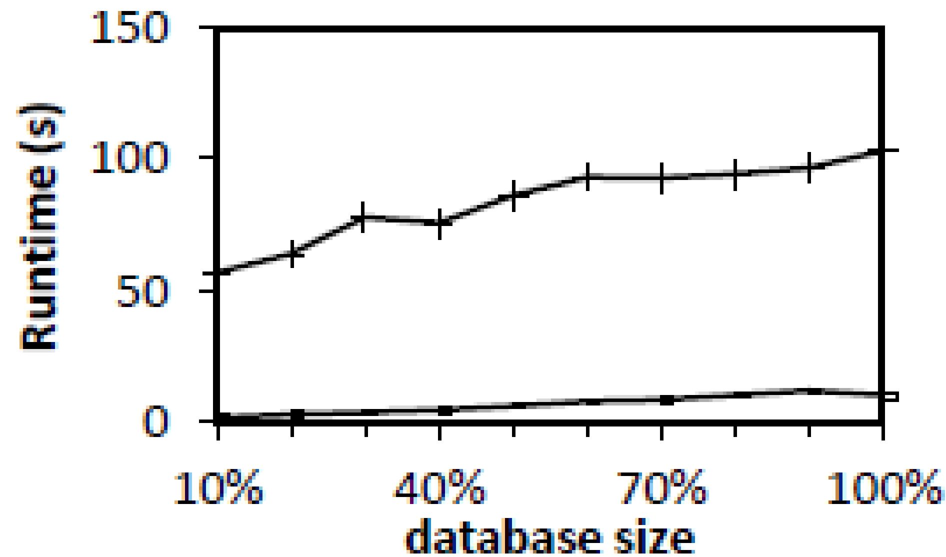


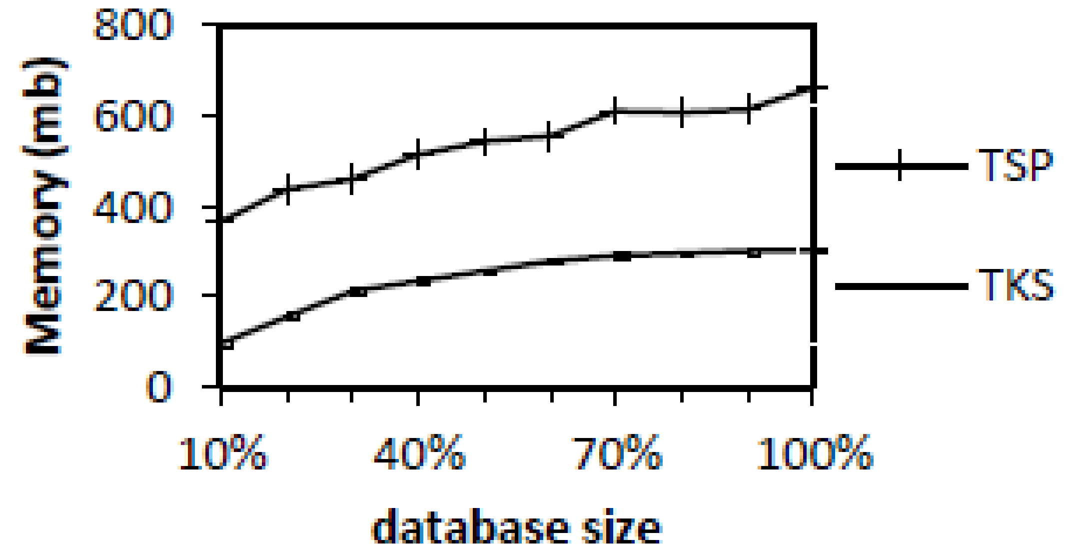


Experiment 3 – database size

- TKS and TSP
- *k = 1000*,
- database size = 10%, 20% …100 %.

Leviathan



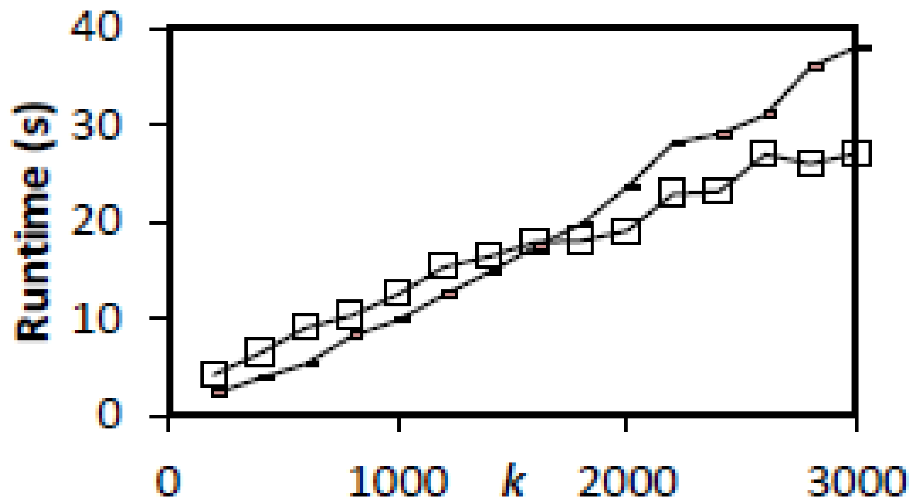


Both algorithm have great scalability.

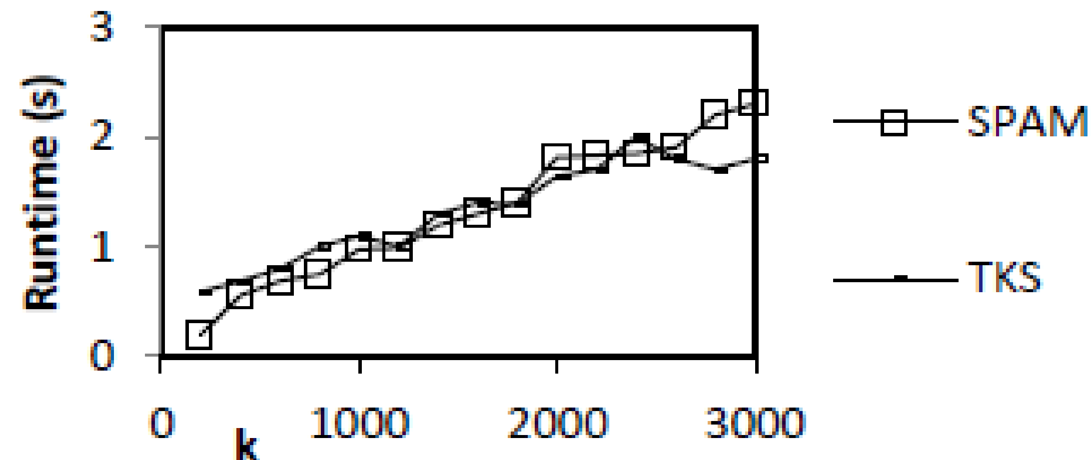
Experiment 4 – Comparison with SPAM

- We compared TKS with SPAM for the optimal minimum support to generate *k* patterns.
- In practice, very hard to choose optimal threshold for users.

Leviathan



Snake



Execution time close to SPAM and similar scalability, although top-k seq. pattern mining is harder!

Conclusion

- TKS
  - ➢ a new vertical algorithm for top-k sequential pattern mining,
  - ➢ spam-based + effective optimizations to prune the search space
  - ➢ outperforms the state-of-the-art algorithm by an order of magnitude in execution time and memory, and has better scalability
  - ➢ low performance overhead compared to SPAM
- Source code and datasets available as part of the **SPMF data mining library (**GPL 3).

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/