Permutation-based Sequential Pattern Hiding

Robert Gwadera
EPFL
robert.gwadera@epfl.ch

Aris Gkoulalas-Divanis
IBM Research-Ireland
arisdiva@ie.ibm.com

Grigorios Loukides
Cardiff University
g.loukides@cs.cf.ac.uk

ICDM 2013
IEEE International Conference on Data Mining
Dallas, Texas / December 8, 2013

Overview

Motivation (1/3)

- Sequence data increasingly shared by organizations to enable mining applications
  - market-basket analysis
  - web and process modeling
  - preference-based services
- Mining may expose *sensitive* sequential patterns, which
  - allow intrusive inferences to individuals
  - provide competitive advance to organizations
- This threat causes unsolicited advertisement, customer profiling, financial and reputational loss, etc.
- It has transpired in several cases (e.g., Walmart - P&G).

Motivation (2/3)

- Insurance company shares sequence data with marketing partners.
- Data can be mined to find the number of customers who bought
  - sports car, travel, and home insurance in this order ($[f, g, b]$), or
  - travel, home, and trailer insurance in any order ($\{g, b, d\}$).
- Sensitive patterns are exposed when their support is at least 3.
  - e.g., $[a, c, e] \rightarrow$ insurance purchase for motorbike, jet-ski and then truck

a b c d e f
a b c e
a c e h b
f g c e a b
d f g h b
d h b f g
f h g b
c d f g e
d h f g b

Original data

$[a, c, e]$
$[d, f, g]$
$[d, h, b]$

Sensitive patterns

To preserve privacy and mining accuracy

- Upper-bound the support of sensitive patterns (*minSup* threshold), so that they are no longer actionable.
- Minimize *side-effects* (changes in frequent, nonsensitive patterns)

Motivation (3/3)

- All existing methods [1, 3] use *symbol deletion*.
  - minimal number of deleted symbols and/or side-effects
- Deletion is a *drastic* operation - it reduces symbol support
  - *utility loss* in sequence mining and tasks based on itemset properties.
- The state-of-the-art algorithm, *SBSH* [3], applied with $minSup = 3$:
  - 3 symbol deletions, 5 side-effects (e.g., $[a, c]$ is no longer frequent)
  - 12 out of 48 frequent itemsets are not mineable (*lost*)

a b c d e f
* b c e
a c e h b
f g c e a b
d f g h b
* h b f g
f h g b
c * f g e
d h f g b

Output of *SBSH*

$[a, c]$
$[a, e]$
$[d, b]$
$[d, g]$
$[d, h]$

Side-effects

$\{d, g\}$, $\{d, h\}$
$\{b, d, g\}$, $\{b, d, h\}$
$\{d, f, h\}$, $\{d, g, h\}$, $\{d, g, f\}$
$\{b, d, g, f\}$, $\{b, d, g, h\}$
$\{b, d, f, h\}$, $\{d, g, f, h\}$
$\{b, d, g, f, h\}$

Lost itemsets

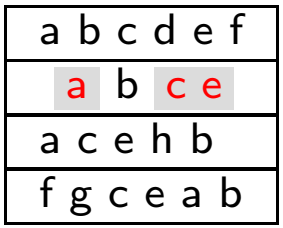
Contributions (1/3)

The first permutation-based approach to hiding sensitive patterns.

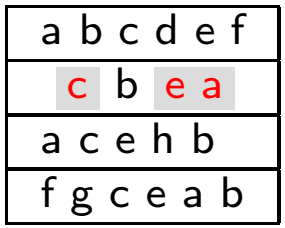
- Permutation induces a different, non-sensitive, symbol ordering and preserves symbol support, but
  - Huge number of permutations (factorial to sensitive pattern length).
    $\rightarrow$ Which ones to use?
  - A subset of the sequences need to be *sanitized*.
    $\rightarrow$ Which ones to sanitize?
  - *Side-effects* and *Distortion* (change to sequences) must be avoided.
    $\rightarrow$ Challenging problem due to symbol overlap.

Contributions (2/3)

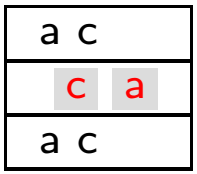
- We investigate how to generate permutations that avoid side-effects
  - Conditions for identifying *candidates* for **lost** and **ghost**
  - Proof that the problem is NP-complete



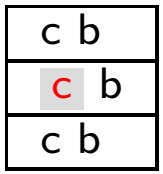
Original data



Sanitized data



$[a, c]$ is lost



$[c, b]$ is ghost

- We develop *PDPG* for generating permutations that avoid side-effects and have minimum distortion.
  - Heuristic to reduce side-effects, if they cannot be avoided
  - Distortion measured by *Cayley distance* (based on # of symbol swaps)
  - Permutations with *implausible* symbol orderings are not generated

Contributions (3/3)

- We propose *PH* for hiding sensitive patterns.
  - selects records whose sanitization would cause "few" side-effects
  - sanitizes each selected record, by replacing sensitive patterns with the permutations generated by PDPG.

a b c d e f
* b c e
a c e h b
f g c e a b
d f g h b
* h b f g
f h g b
c * f g e
d h f g b

Output of *SBSH*

a b c d e f
a b e c
a e c h b
f g c e a b
d g f b h
d b h g f
f h g b
c d g f e
d h f g b

Output of *PH*

- No side-effects → accurate sequential pattern mining
- No deleted symbols → accurate frequent itemset mining
- PDPG minimizes Cayley distance $[a, e, c]$ vs. $[e, a, c]$

- Experiments showing that *PH* allows accurate sequential pattern and frequent itemset mining, and estimation of item/itemset support.

Background (1/4)

Frequent sequential pattern mining framework

- $\mathcal{A} = \{a_1, a_2, \ldots, a_{|\mathcal{A}|}\}$: symbol alphabet
- $\mathbf{S} = \{t^{(1)}, t^{(2)}, \ldots, t^{(|\mathbf{S}|)}\}$: collection of sequences (*transactions*)

- $s = [\mathrm{s}_1, \mathrm{s}_2, \ldots, \mathrm{s}_{|s|}]$ is *subsequence* of $s' = [\mathrm{s}'_1, \mathrm{s}'_2, \ldots, \mathrm{s}'_{|s'|}]$ (i.e., $s \sqsubseteq s'$), if there exist integers $1 \leq i_1 \leq i_2 \ldots \leq i_{|s|}$ s.t. $\mathrm{s}_1 = \mathrm{s}'_{i_1}, \mathrm{s}_2 = \mathrm{s}'_{i_2}, \ldots, \mathrm{s}_{|s|} = \mathrm{s}'_{i_{|s|}}$.
- s' is a *supersequence* of s and s is *contained* in s'.

- $sup_{\mathbf{S}}(s)$: *support* of s in $\mathbf{S}$ (number of transactions)
- A sequence s is a *frequent sequential pattern*, if $sup_{\mathbf{S}}(s) \geq minSup$, and *infrequent* otherwise.
- The problem of sequential pattern mining [2] is to find the set $\mathcal{F}_{\mathbf{S},minSup}$ of all frequent sequential patterns in $\mathbf{S}$, given *minSup*.

Background (2/4)

Permutations and Cycle notation

- $\pi(\mathcal{E})$: permutation of a set of elements $\mathcal{E}$ (1-1 mapping from $\mathcal{E}$ to itself)
- $\mathcal{E}$ refers to positions of symbols in sequences (not to the actual symbols)
- *Cycle notation* of a permutation $\pi(s)$ obtained by grouping the elements in s and $\pi(s)$ that swap

$\pi(s)$	cycle notation
$[0, 1, 2]$	$[(0), (1), (2)]$
$[1, 2, 0]$	$[(0, 1, 2)]$

- A *cycle* $(x_1, \ldots, x_n)$ is a permutation that maps x_1 to x_2, x_2 to x_3, ..., x_n to x_1, while fixing any other element in $\mathcal{E}$. This cycle is a product of $n-1$ *transpositions* (i.e., cycles of length 2) [4], i.e.,

$$(x_1, x_2, \ldots, x_n) = (x_1, x_2)(x_1, x_3)\ldots(x_1, x_n)$$

Background (3/4)

- Example
  - The permutation $[1, 2, 0]$ corresponds to $[(0, 1, 2)] = (0, 1)(0, 2)$.
  - $[0, 1, 2]$ and (0,1) gives $[1, 0, 2]$, and $[1, 0, 2]$ and $(0, 2)$ gives $[1, 2, 0]$

Cayley distance

The *Cayley distance* $\mathcal{C}(\pi, \mathcal{I})$, between a permutation π and the identity permutation [4]:

$$\mathcal{C}(\pi, \mathcal{I}) = n - c(\pi)$$

where $c(\pi)$ is the number of cycles in π and n is the number of symbols in π.

- Example
  - $[1, 2, 0]$ has 1 cycle and length 3, so $C([1, 2, 0]) = 3 - 1 = 2$

Background (4/4)

Sequential pattern hiding

Given the original dataset $\mathbf{S}$, a support threshold *minSup* and a set of sensitive sequential patterns $\mathcal{S}$ (selected by data owners), construct a version $\mathbf{S}'$ of $\mathbf{S}$ such that:

(I) $sup_{\mathbf{S}'}(s) < minSup$, for each $s \in \mathcal{S}$,

(II) $\mathcal{F}_{\mathbf{S}'} = \mathcal{F}_{\mathbf{S}} - \mathcal{S}^*$, where $\mathcal{S}^* = \{s^* \in \mathcal{F}_{\mathbf{S}'} | s \sqsubseteq s^*, s \in \mathcal{S}\}$, and

(III) $\mathbf{S}'$ is constructed with minimum distortion.

- The problem is NP-hard (the proof follows from [1]).
- Avoiding side-effects more important than reducing distortion [3], as data are shared for mining.
- Extensions for multiple support thresholds are straightforward [1].

Hiding methodology (1/8)

Symbol-based identification of candidates for lost and ghost

S1 When s and s' share symbols, the support of s' may increase or decrease.

S2 When (a)-(c) hold simultaneously, the support of s' may decrease:

(a) s and s' have a common subsequence s''

(b) s and s' co-occur in t, by sharing s''

(c) the permutation of s causes at least one symbol in s'' to change its relative ordering w.r.t. the other symbols in s''.

Swapping symbols in	s	s'	s''	t	$\pi(s)$	t after $\pi(s)$
s''	$[a, b, c]$	$[a, d, c]$	$[a, c]$	[a , $b, d,$ c]	$[c, b, a]$	[c , $b, d,$ a]
$s \setminus s''$, s''	$[a, b, c]$	$[d, b, c]$	$[b, c]$	[a , $d, b,$ c]	$[c, b, a]$	[c , $d, b,$ a]

S3 When (a)-(c) hold simultaneously, the support of s' may increase:

(a) s and s' have a common set of symbols s^*, but s' is not a subsequence of s, and s is not a subsequence of s'

(b) s co-occurs in t with a permutation $\pi(s')$ of s', by sharing s^*

(c) the permutation of s reverses $\pi(s')$ and recovers s'.

Hiding methodology (2/8)

Support-based identification of candidates for lost and ghost

C1 A frequent, nonsensitive pattern s' may become *infrequent* (lost), as a result of permuting s in $\Delta_{supp}(s|\pi(s))$ transactions in $\mathbf{S}$, if

- it satisfies S1 and S2, and
- its support, before permuting s, satisfies

$$sup_{\mathbf{S}}(s') < minSup + \Delta_{supp}(s|\pi(s)).$$

C2 An infrequent, nonsensitive pattern s' may become *frequent* (ghost), as a result of permuting s in $\Delta_{supp}(s|\pi(s))$ transactions in $\mathbf{S}$, if

- it satisfies S1 and S3, and
- its support, before permuting s, satisfies

$$sup_{\mathbf{S}}(s') \geq minSup - \Delta_{supp}(s|\pi(s)).$$

- C1 and C2 are used to identify candidates efficiently
- Heuristic order and simple pattern matching algorithm for checking S3

Hiding methodology (3/8)

Pattern-constrained permutation

A permutation $\pi(s)$ s.t. all candidate patterns for lost and ghost do not lose and gain occurrence in t, respectively, when $\pi(s)$ replaces s.

t	s	$d^{(-t)}$	$d^{(+t)}$	$\pi(s)$	t'
$[0,1,2,3,4]$	$[1,2,3]$	$[0,1,2]$	$[2,1,4]$	$[3,1,2]$	$[0,3,1,2,4]$

- Finding a pattern-constrained permutation $\pi(s)$, given candidates patterns, for lost and ghost, is NP-complete (reduction to SAT)

Partial permutation $\pi(s|m)$

A permutation of a prefix of positions of s, from 0 to $m-1$, for $m \leq |s|$.

Hiding methodology (4/8)

Satisfiability of $\pi(s|m)$ w.r.t. a candidate pattern for

- *lost*, if applying swaps to the remaining symbols of s can lead to preserving the candidate
- *ghost*, if there is at least a swap of symbols of s that can lead to avoiding the candidate

- Conditions based on Cayley distance in the paper

t	s	$d^{(-t)}$	$d^{(+t)}$	$\pi(s\|2)$	t'
$[0,1,2,3,4]$	$[1,2,3]$	$[0,1,2]$	$[2,1,4]$	$[2,1]$	$[0,2,1,3,4]$

- $d^{(+t)}$ occurs after applying $\pi(s|2)$.
- If we swap the remaining symbol 3 in s with 2, $s = [3,1,2]$ and $t' = [0,3,1,2,4]$, which preserves $d^{(-t)}$ and avoids $d^{(+t)}$.
- Thus, $\pi(s|2)$ is satisfiable w.r.t. $d^{-(t)}$ and $d^{+(t)}(s)$.

Hiding methodology (5/8)

PDPG (*Pattern-constrained Distance-based Permutation Generation*)

- Aims at generating a random, partial permutation $\pi(s|m)$ that is *pattern-constrained* and has minimum Cayley distance (max. # cycles k)
- Performs a random search over all possible solutions
- It starts from $\pi(1|m=1) = [(0)]$ and extends it to $\pi(s|m=|s|)$, iteratively
- The permutation to extend in each iteration is selected randomly, from both types of permutations w.r.t. their cycle structure, but must be satisfiable
- The selection is made from the type that is more likely to produce a satisfiable permutation $p(s|m+1)$

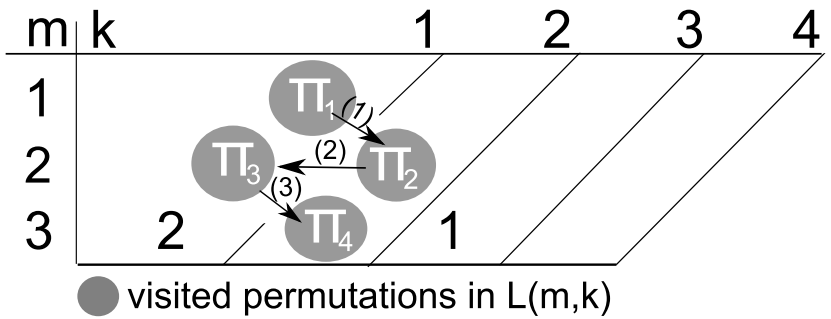
- The worst case time complexity of *PDPG* is $O(|s|^3 M)$ (vs. $O(|s|^2)$ for the unconstrained case).

Hiding methodology (6/8)

- *PDPG* through an example

t	s	$d^{(-t)}$	$d^{(+t)}$
$[0,1,2,3,4]$	$[1,2,3]$	$[2,3,4]$	$[3,2,4]$

m	k	$\pi(s\|m)$	constraints			
			$d^{-(t)}(s) = [2,3,4]$		$d^{+(t)}(s) = [3,2,4]$	
			Satisf.	$\mathcal{C}(d^{-(t)}(s), \mathcal{I})$	Satisf.	$\mathcal{C}(d^{+(t)}(s), \mathcal{I})$
1	1	$\pi_1 = [(0)]$	true	0	true	1
2	2	$\pi_2 = [(0),(1)]$	true	0	true	1
2	1	$\pi_3 = [(0,1)]$	true	0	true	1
3	2	$\pi_4 = [(0,1),2]$	true	0	true	1



Hiding methodology (7/8)

- *RR-PDPG*, a heuristic used when *PDPG* cannot find a pattern-constrained permutation
- It aims at maximizing # of satisfied candidates, based on that
  - short candidates for ghost are the easiest to satisfy
  - long candidates for lost are the hardest

RR-PDPG

1. Rank the patterns in $\mathcal{D}^{-(t)}(s)$, in increasing order of length and the patterns in $\mathcal{D}^{+(t)}(s)$, in decreasing order of length.
2. Recursively halve the candidates for ghost, and select the first half, until all selected candidates are satisfied.
3. Recursively halve the candidates for lost and use *PDPG* to find a permutation that satisfies:
   - all selected candidates for lost, and
   - all the previously selected candidates for ghost

- *RR-PDPG* takes $O(|s|^3 M log(M))$ time, in the worst case

Hiding methodology (8/8)

- There are implausible symbol orderings (*forbidden patterns*) in certain application domains (e.g., process mining)
  - e.g., customer accepts contract terms, before they are introduced to a new service

Forbidden patterns

- Must not occur to help data utility and prevent attackers from identifying sanitized transactions
- Are given as input to *PDPG*, which generates *only* permutations that contradict them
- Are specified by data owners, based on domain knowledge, or discovered from the data

Permutation-based Hiding (1/2)

PH (Permutation-based Hiding)

(I) Identification of candidate patterns for lost and ghost

(II) Selection of transactions to sanitize

(III) Sanitization of the selected transactions

Phase (I)

foreach *sensitive pattern* **do**
- Compute candidates satisfying C1
- Compute candidates satisfying C2

Phase (II)

1. Transfer transactions that do not support any sensitive pattern
2. Rank each transaction w.r.t. the upper-bound of side-effects that would be caused by its sanitization
3. Sort transactions in decreasing rank
4. Transfer each transaction, if this does not create frequent sensitive patterns

Permutation-based Hiding (2/2)

Phase (III)

foreach *transaction* $t \in \mathbf{S}$ **do**
- Retrieve candidates for lost and ghost for t **foreach** *sensitive pattern* $s \in t$ **do**
  - Hide s using the following steps ordered w.r.t. their precedence:
    - i Apply *PDPG* and replace s with the generated permutation.
    - ii Apply *RR-PDPG* and replace s with the generated permutation, if step i fails and no patterns are forbidden.
    - iii Apply symbol deletion, if steps i and ii fail.
  - Update t and its candidate patterns
- Update candidate patterns

- *PH* needs $O(|\mathbf{S}| \cdot |\mathcal{F}_{\mathbf{S},sup}| \cdot |f|)$ time, in the worst case
- $|\mathbf{S}|$ is the size of the dataset, $|\mathcal{F}_{\mathbf{S},sup}|$ and $|f|$ are the # and avg. length of the obtained frequent patterns

Experimental evaluation (1/3)

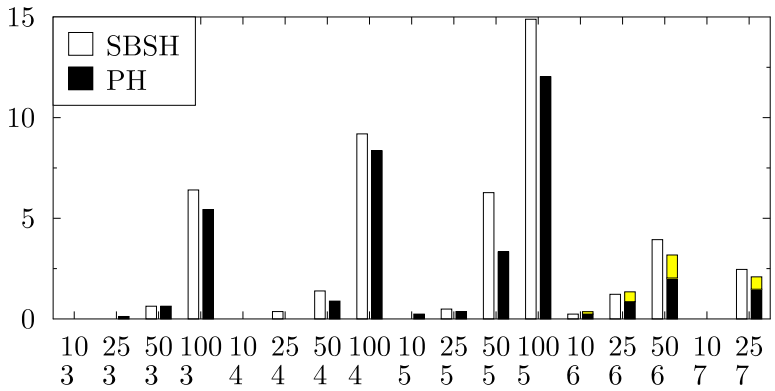
Datasets and *minSup* values

Dataset	$\|\mathbf{S}\|$	$\|\mathcal{A}\|$	**Avg.** $\|s^{(i)}\|$	*minSup*
MSN	989818	17	5.7	3711,**4949**,9898,19796
BMS	59602	497	2.5	**59**
TRUCKS	273	100	20.1	**20**,40,50,60,67,80

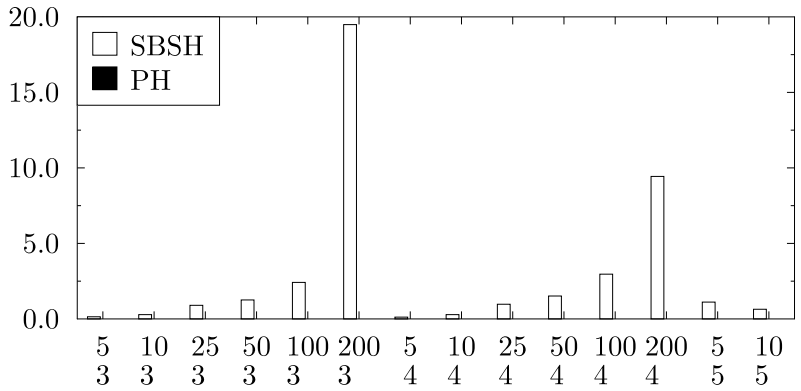
Comparison

- *SBSH*: the state-of-the-art algorithm [3]
- Data utility
  - Sequence mining
  - Frequent itemset mining
  - Difference between the distribution of the support of items (resp., frequent itemsets) in the original and the sanitized data
- Efficiency

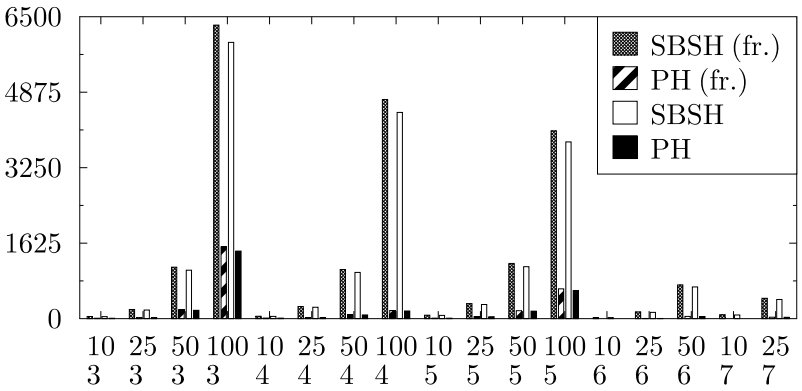
Experimental evaluation (2/3)



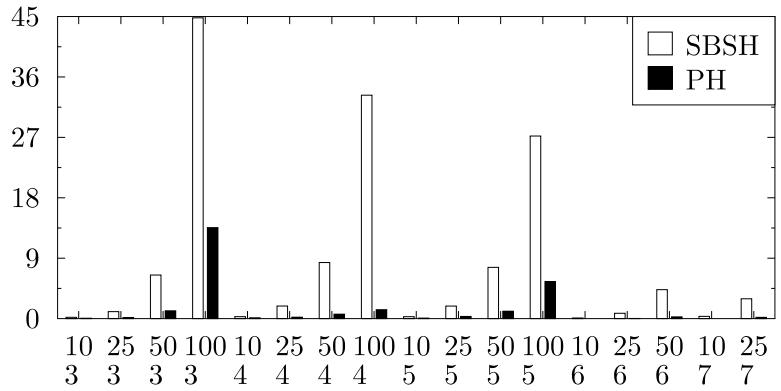
Side-effects (MSNBC)
28% fewer, $< 1.5\%$ ghosts



Lost frequent itemsets (BMS)
up to 20% for *SBSH*

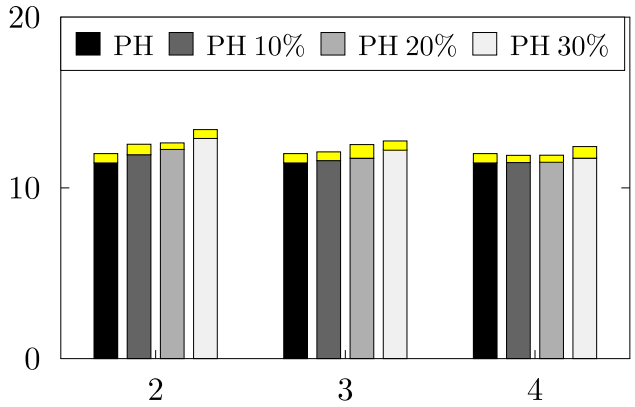


KL-divergence for items and freq. items (MSNBC), $[4, 35]$ times better

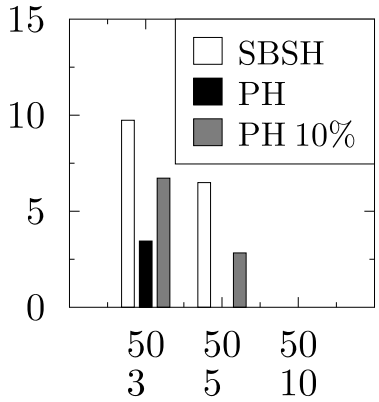


KL-divergence for frequent itemsets (MSNBC), $[2.1, 32.5]$ times better

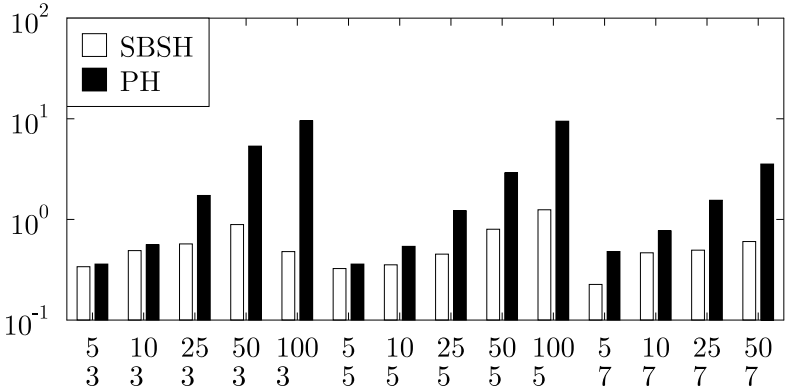
Experimental evaluation (3/3)



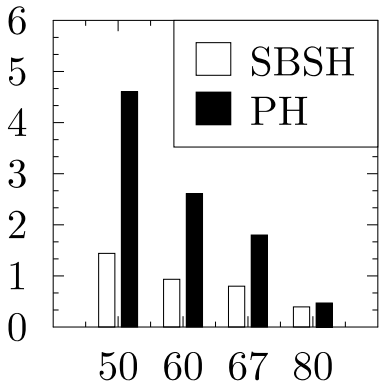
Side-effects (TRU) 50 sens. with len. 5
$10 - 30\%$ forbidden patterns with length $[2, 4]$



Lost frequent itemsets (TRU)



Runtime (TRU), 3.8 times slower on average



Runtime (TRU) vs. *minSup*
1.3 times slower on average

Preventing background knowledge attacks

- *Minimum harm approach* - most common
  - Sensitive patterns are not exposed, when their support is below *minSup*
  - These patterns model "unexpected" knowledge
- *Background knowledge attacks to*
  i. discover s when its support and that of "few" nonsensitive patterns is lower but "close" to *minSup*
  ii. exclude certain permutations of s from consideration

Alternative approaches (different utility/privacy trade-off)

- *Maximum uncertainty* - hide s using the maximum number of permutations
- *Bounded uncertainty* - hide s by replacing it with at least c different permutations, all having support at least λ

Conclusions

- First permutation-based approach to hide sequential patterns
- *PDPG* and *RR-PDPG* algorithms to generate pattern-constrained permutations
- *PH* algorithm employs *PDPG* to sanitize sequential data

Thank you!

Acknowledgements

- Reviewers, for their constructive comments
- EU project OpenIoT (ICT 287305)
- Royal Academy of Engineering

References

[1] O. Abul, F. Bonchi, and F. Giannotti.
Hiding sequential and spatiotemporal patterns.
TKDE, 22(12):1709–1723, 2010.

[2] R. Agrawal and R. Srikant.
Mining sequential patterns.
In *ICDE*, pages 3–14, 1995.

[3] A. Gkoulalas-Divanis and G. Loukides.
Revisiting sequential pattern hiding to enhance utility.
In *KDD*, pages 1316–1324, 2011.

[4] J. Marden.
Analyzing and Modeling Rank Data.
Chapman&Hall, 1995.