Efficient Algorithm for the Problem of High-Utility Itemset Mining

Philippe Fournier-Viger

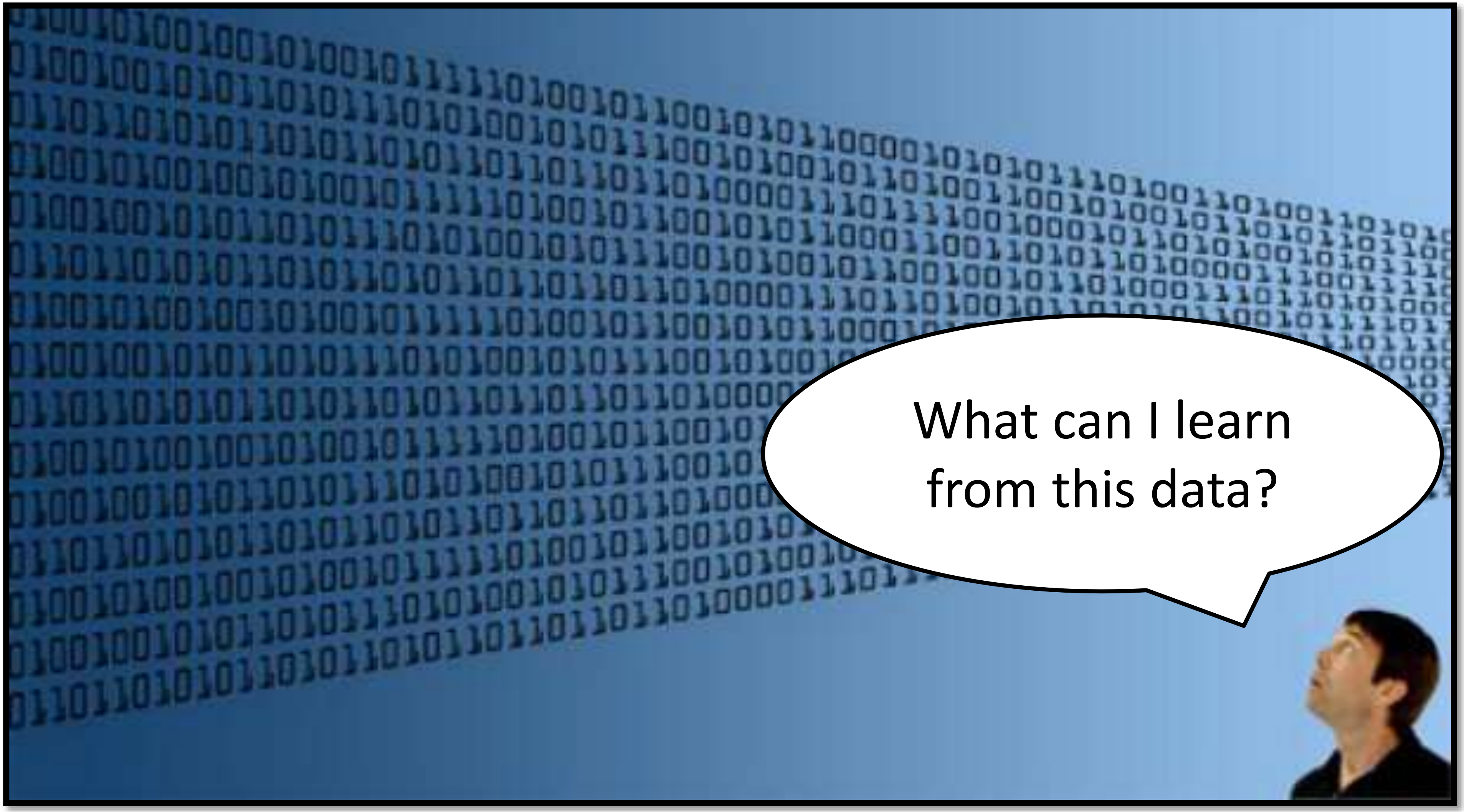
Assistant professor

Department of Computer Science,

University of Moncton, Canada

Topic of this talk

- The problem of High utility itemset mining
- Three new algorithms
  - FHM
  - FHN
  - FOSHU



This talk is about **data mining**, and more specifically, the subfield of "***pattern mining***" (discovering interesting *patterns* in database).

The goal of pattern mining

- Given a **database**, we want to discover **patterns** that are: *novel, unexpected* and *useful*.
- **Example:** discovering new relationships between genes and diseases that lead to the development of a new medicine.

But in what kind of data?

Various types:

- relational databases,
- graphs,
- text,
- spatial data,
- sequences, time series, etc.

We are interested in
transaction databases
to analyse **shopping behavior**.

What is a transaction database?

- Let be a set of items **{a, b, c, d, e, …}** sold in a store.

- A ***transaction*** is a set of items bought by a customer.
- **Example:**

Transaction	items
T_1	{a, b, c, d, e}
T_2	{a, b, e}
T_3	{c, d, e}
T_4	{a, b, d, e}

four transactions

Discovering Frequent Patterns

- The **task of *frequent pattern mining*** was proposed by Agrawal (1993).
- **Input**: a transaction database and a parameter ***minsup*** ≥ 1.
- **Output**: the ***frequent itemsets*** (all sets of items appearing in at least ***minsup*** transactions).

An example

transaction database

Transaction	items
T_1	{a, b, c, d, e}
T_2	{a, b, e}
T_3	{c, d, e}
T_4	{a, b, d, e}

***minsup* = 2**

frequent itemsets

Itemset	Support
{e}	**4**
{d, e}	**3**
{b, d, e}	**2**
{a}	**3**
…	**…**

How to solve this problem?

The naïve approach:

- scan the database to count the frequency of **each** possible itemset.
  e.g.: {a}, {a,b}, {a,c}, {a,d}, {a,e}, {a,b,c}, {a,b,d} …
  {b}, {b, c}, … {a,b,c,d,e}
- If $\boldsymbol{n}$ items, then $\mathbf{2^n - 1}$ possible itemsets.
- Thus, inefficient.

- **Several efficient algorithms:**
  - Apriorii, FPGrowth, H-Mine, LCM, PrePost, etc.

The "Apriori" property

Property (anti-monotonicity).

Let be itemsets **X** and **Y**. If **X** $\subset$ **Y**, then the support of Y is less than or equal to the support of **X**.

Example

Transaction	items
T_1	{a, b, c, d, e}
T_2	{a, b, e}
T_3	{c, d, e}
T_4	{a, b, d, e}

The support of **{a,b}** is **3**.

Thus, supersets of **{a,b}** have a support ≤ **3.**

Limitations of frequent patterns

- Frequent pattern mining has **many** applications**.**
- However, it has important **limitations**
  - many frequent patterns are not interesting,
  - quantities of items in transactions must be 0 or 1
  - all items are considered as equally important (having the same weight)

High Utility Itemset Mining

- **A generalization of frequent pattern mining:**
  - items can appear more than once in a transaction (**e.g.** a customer may buy 3 bottles of milk)
  - items have a unit profit (**e.g.** a bottle of milk generates 1 $ of profit)
  - the goal is to find **patterns that generate a high profit**
- **Example**:
  - {caviar, wine} is a pattern that generates a high profit, although it is rare

High Utility Itemset Mining

Input: transaction database with quantities

Trans.	**items**
T_0	a(1), b(5), c(1), d(3), (e,1)
T_1	b(4), c(3), d(3), e(1)
T_2	a(1), c(1), d(1)
T_3	a(2), c(6), e(2)
T_4	b(2), c(2), e(1)

unit profit table

item	**unit profit**
a	5 $
b	2 $
c	1 $
d	2 $
e	3 $

and a threshold ***minutil***

Output: high-utility itemsets

(itemsets having a **utility** $\geq$ ***minutil***)

A full example

transaction database with quantities

Trans.	items
T_0	a(1), b(5), c(1), d(3), (e,1)
T_1	b(4), c(3), d(3), e(1)
T_2	a(1), c(1), d(1)
T_3	a(2), c(6), e(2)
T_4	b(2), c(2), e(1)

unit profit table

item	unit profit
a	5 $
b	2 $
c	1 $
d	2 $
e	3 $

Suppose that
minutil **= 25 $** →

High utility itemsets

{a,c} : 28$
{a,b,c,d,e}: 25 $
{b,c,d}: 34 $
{b,c,e} : 37 $
{b,d,e} : 36 $
{c, e}: 27$
{a,c,e}: 31 $
{b,c} : 28 $
{b,c,d,e}: 40 $
{b,d} : 30 $
{b,e} : 31 $

How to calculate *utility*?

Trans.	items
T_0	a(1), b(5), c(1), d(3), (e,1)
T_1	b(4), c(3), d(3), e(1)
T_2	a(1), c(1), d(1)
T_3	a(2), c(6), e(2)
T_4	b(2), c(2), e(1)

item	unit profit
a	5 \$
b	2 \$
c	1 \$
d	2 \$
e	3 \$

The utility of an itemset is the sum of the utility of items (profit $\times$ quantity) in that itemset for transactions where the itemset appears.

u({a,e})

How to calculate *utility*?

Trans.	items
T_0	a(1), b(5), c(1), d(3), (e,1)
T_1	b(4), c(3), d(3), e(1)
T_2	a(1), c(1), d(1)
T_3	a(2), c(6), e(2)
T_4	b(2), c(2), e(1)

item	unit profit
a	5 $
b	2 $
c	1 $
d	2 $
e	3 $

The utility of an itemset is the sum of the utility of items (profit × quantity) in that itemset for transactions where the itemset appears.

$$u(\{a,e\}) = (1\times 5\$ + 1\times 3\$)$$

How to calculate *utility*?

Trans.	items
T_0	a(1), b(5), c(1), d(3), (e,1)
T_1	b(4), c(3), d(3), e(1)
T_2	a(1), c(1), d(1)
T_3	a(2), c(6), e(2)
T_4	b(2), c(2), e(1)

item	unit profit
a	5 \$
b	2 \$
c	1 \$
d	2 \$
e	3 \$

The utility of an itemset is the sum of the utility of items (profit × quantity) in that itemset for transactions where the itemset appears.

$$u(\{a,e\}) = (1\times 5\$ + 1\times 3\$) + (2\times 5\$ + 2\times 3\$) = 24\ \$$$

A difficult task!

Why?

- because *utility* is **not** *anti-monotonic* (i.e. does not respect the *Apriori property*)
- **Example**:

  u({a}) = 20 $
  u({a,e}) = 24 $

  u({a,b,c}) = 16 $
- Thus, frequent itemset mining algorithms cannot be applied to this problem.

How to solve this problem?

- **Several algorithms:**
  - **Two-Phase (PAKDD 2005),**
  - **IHUP (TKDE, 2010),**
  - **UP-Growth (KDD 2011),**
  - **HUI-Miner (CIKM 2012),**
  - **FHM (ISMIS 2014)**
- **Key idea**: calculate an upper-bound on the utility of itemsets (**e.g.** the **TWU**) that respects the *Apriori* property to be able to prune the search space.

Transaction Utility

Transaction utility **of a transaction:**

the sum of the utility of all items in that transaction

Trans.	items
T_0	a(1), b(5), c(1), d(3), (e,1)
T_1	b(4), c(3), d(3), e(1)
T_2	a(1), c(1), d(1)
T_3	a(2), c(6), e(2)
T_4	b(2), c(2), e(1)

item	unit profit
a	5 $
b	2 $
c	1 $
d	2 $
e	3 $

Example:

$TU(T_0) = (1 \times 5)+(5 \times 2)+(1 \times 1)+(3 \times 2)+(1 \times 3) = 25\ \$$

Transaction Utility

Transaction utility **of a transaction:**

the sum of the utility of all items in that transaction

Trans.	items
T_0	a(1), b(5), c(1), d(3), (e,1)
T_1	b(4), c(3), d(3), e(1)
T_2	a(1), c(1), d(1)
T_3	a(2), c(6), e(2)
T_4	b(2), c(2), e(1)

item	unit profit
a	5 $
b	2 $
c	1 $
d	2 $
e	3 $

Example:

$TU(T_3) = (2\times 5)+(6 \times 1)+(2 \times 3) = 22\ \$$

The *TWU* upper bound

***TWU* of an itemset:**

the sum of the transaction utility for transactions containing the itemset.

Trans.	items
T_0	a(1), b(5), c(1), d(3), (e,1)
T_1	b(4), c(3), d(3), e(1)
T_2	a(1), c(1), d(1)
T_3	a(2), c(6), e(2)
T_4	b(2), c(2), e(1)

item	unit profit
a	5 $
b	2 $
c	1 $
d	2 $
e	3 $

Example:

TWU({a,e}) = TU(T_0) + TU(T_3) = 25 $ + 22 $ = 47 $

The *TWU* upper bound

Property: The ***TWU*** of an itemset is an upper bound on its *utility*, and all its supersets.

Trans.	items
T_0	a(1), b(5), c(1), d(3), (e,1)
T_1	b(4), c(3), d(3), e(1)
T_2	a(1), c(1), d(1)
T_3	a(2), c(6), e(2)
T_4	b(2), c(2), e(1)

item	unit profit
a	5 $
b	2 $
c	1 $
d	2 $
e	3 $

Example:

TWU({a,e}) = 47 $ $\geq$ u({a,e}) = 24$ and the utility of any superset of {a,e}

TWU based algorithms

- Algorithms such as **Two-Phase** (PAKDD 2005) and **UPGrowth** (KDD 2010) work as follows:
  - **Phase 1:** find each itemset **X** such that **TWU(X)** $\geq$ ***minutil*** using the ***TWU*** upper bound to prune the search space.
  - **Phase 2:** Scan the database again to calculate the exact utility of remaining itemsets. **Output** the high-utility itemsets.

But, a problem

- High-utility itemset mining is still a very expensive task!

- **Our solution:** a new algorithm named **FHM**, which extends **HUI-Miner** (CIKM 2012)

The FHM algorithm

Fournier-Viger, P., Wu, C.-W., Zida, S., Tseng, V. S. (2014) FHM: Faster High-Utility Itemset Mining using Estimated Utility Co-occurrence Pruning. Proc. 21st International Symposium on Methodologies for Intelligent Systems (ISMIS 2014), Springer, LNAI, pp. 83-92.

HUI-Miner(CIKM,2012)

- **HUI-Miner** is one of the fastest algorithm for high utility mining
- It performs a **depth-first search** by appending items to itemsets.

$\emptyset$

HUI-Miner(CIKM,2012)

It performs a **depth-first search** by appending items to itemsets.

Ø

{a}.

{a, b}.

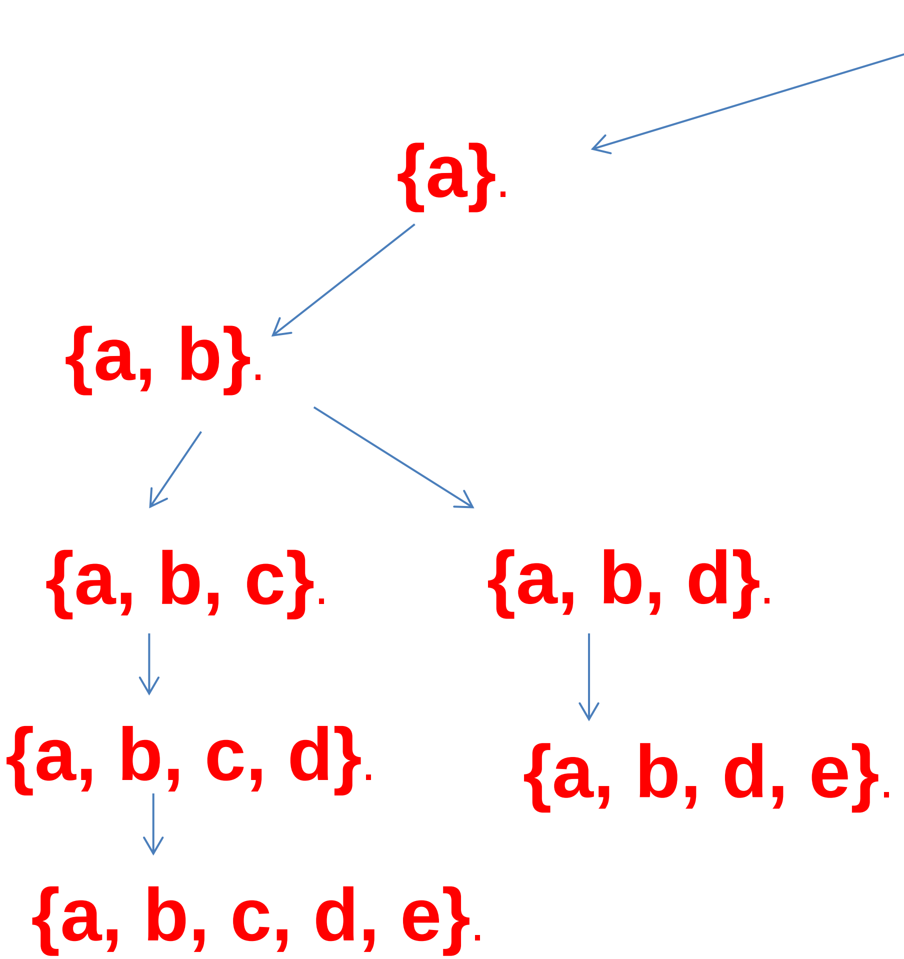
{a, b, c}.

{a, b, c, d}.

{a, b, c, d, e}.

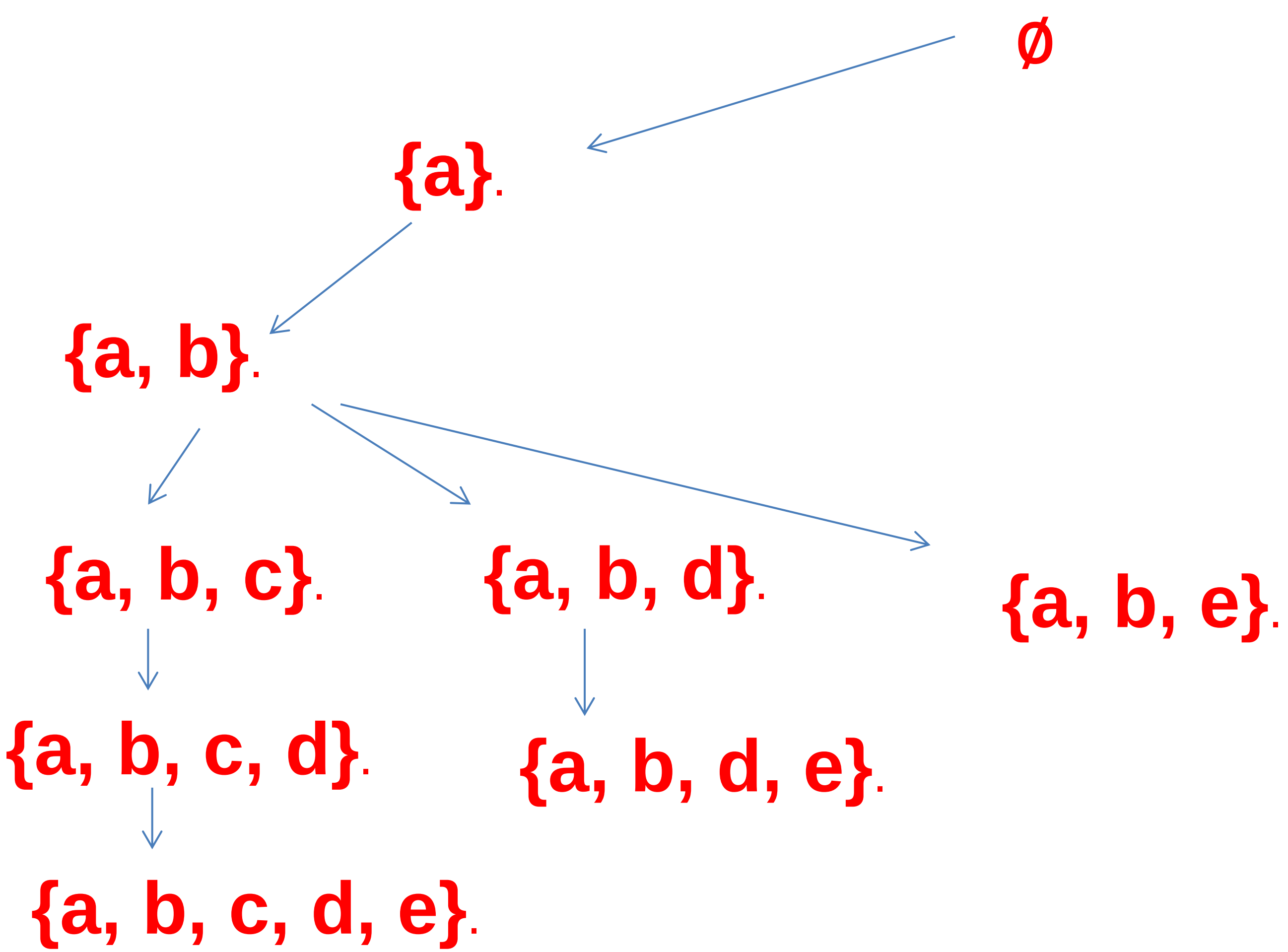
HUI-Miner(CIKM,2012)

It performs a **depth-first search** by appending items to itemsets.



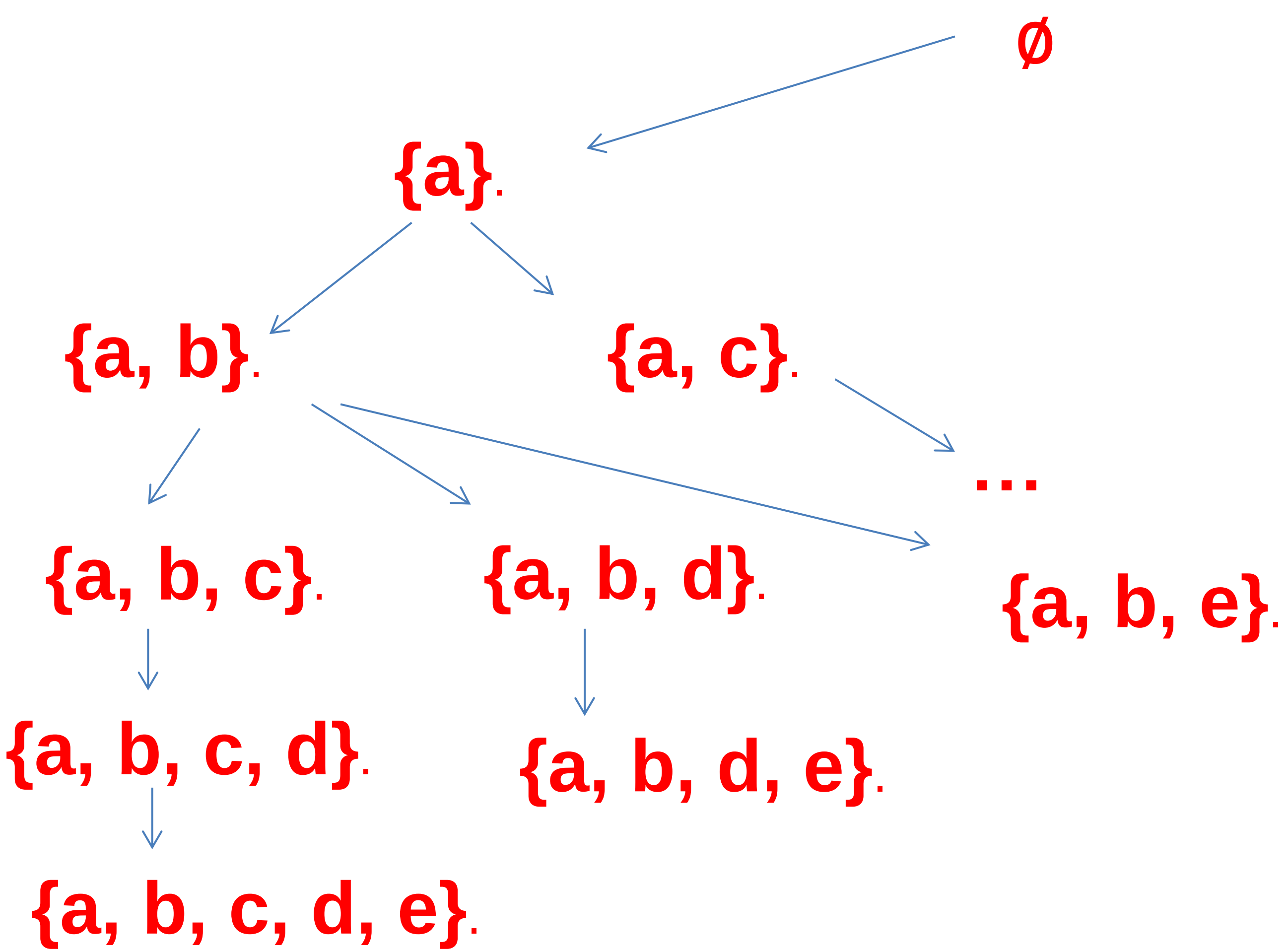
HUI-Miner(CIKM,2012)

It performs a **depth-first search** by appending items to itemsets.



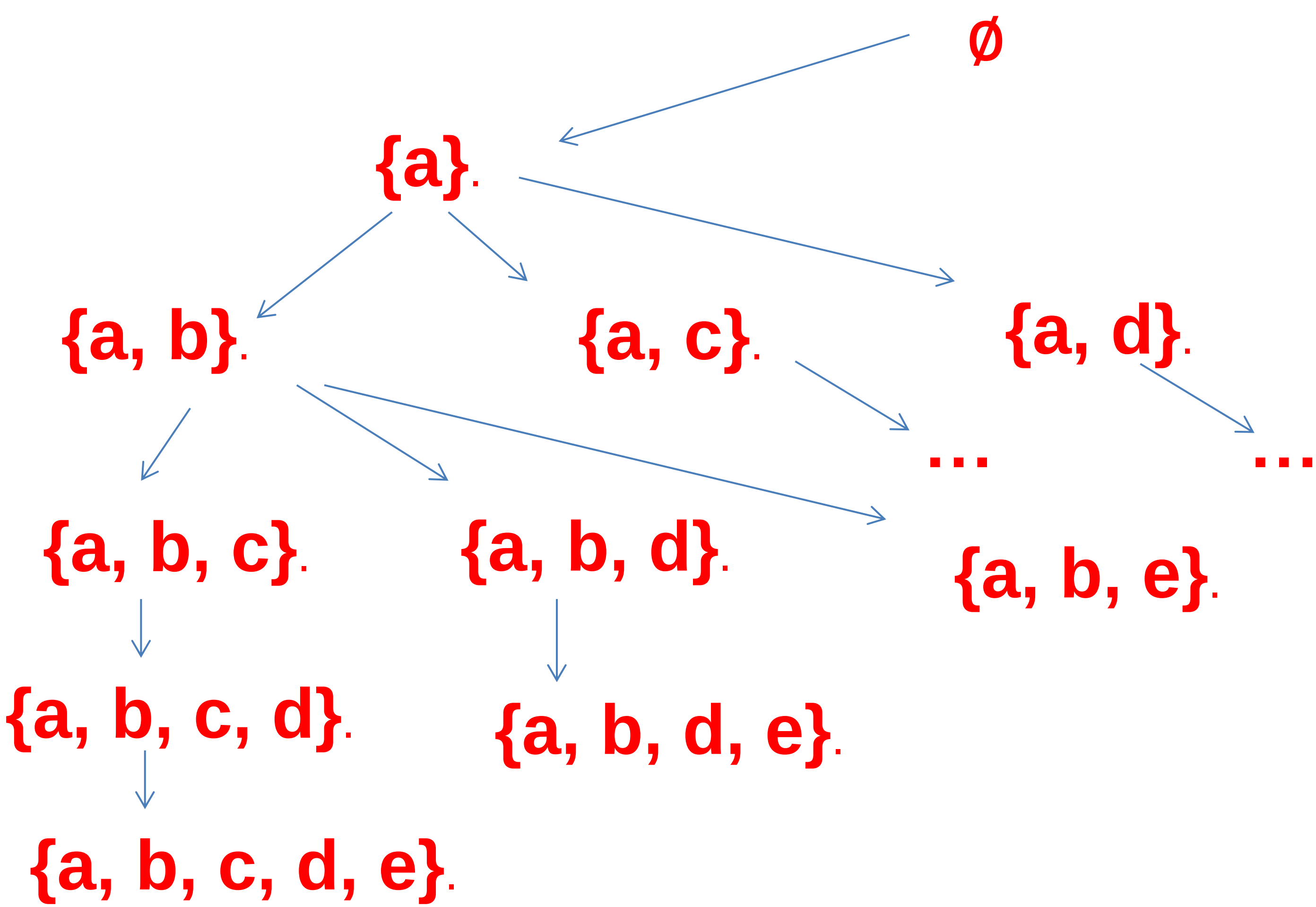
HUI-Miner(CIKM,2012)

It performs a **depth-first search** by appending items to itemsets.



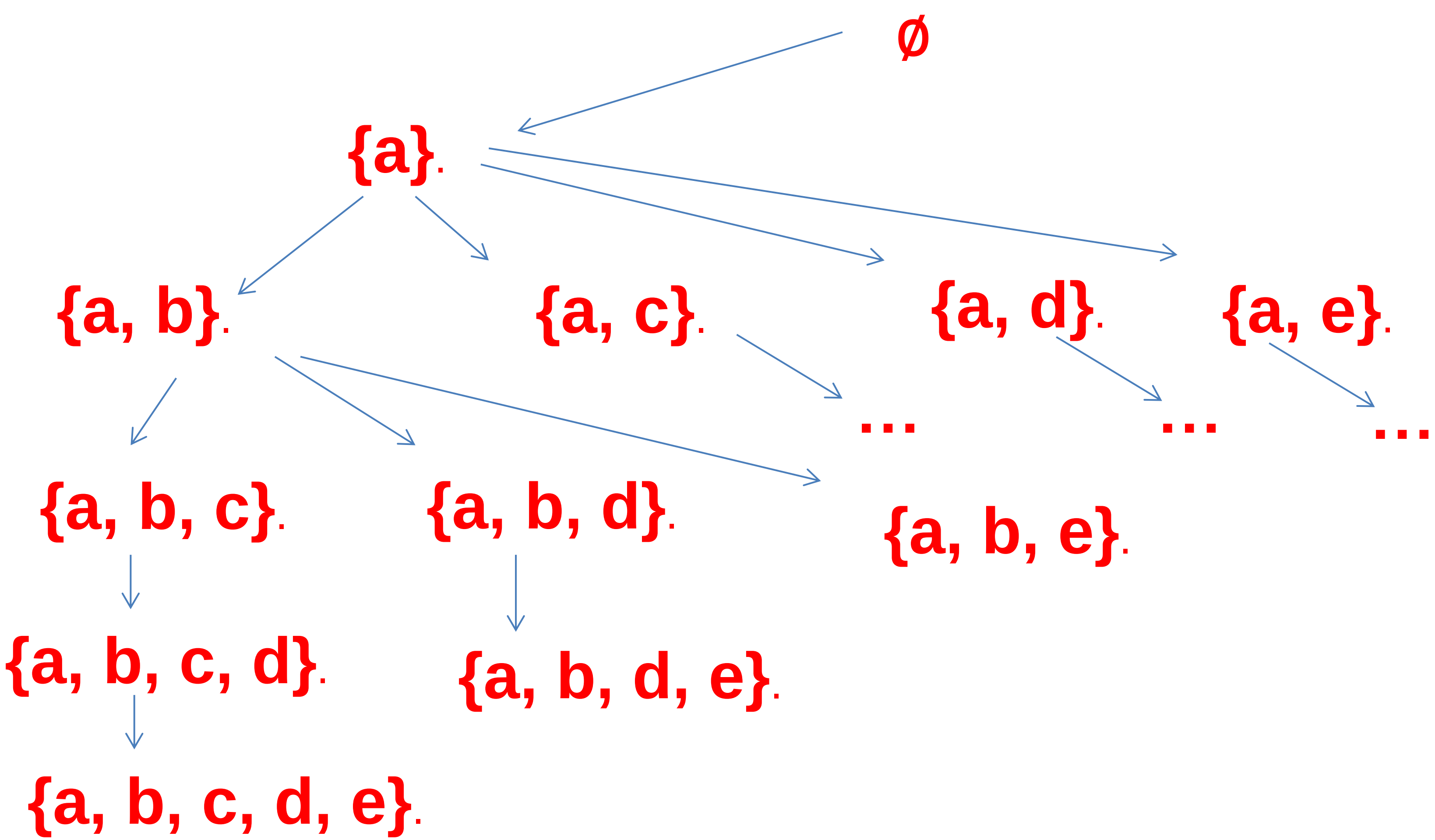
HUI-Miner(CIKM,2012)

It performs a **depth-first search** by appending items to itemsets.



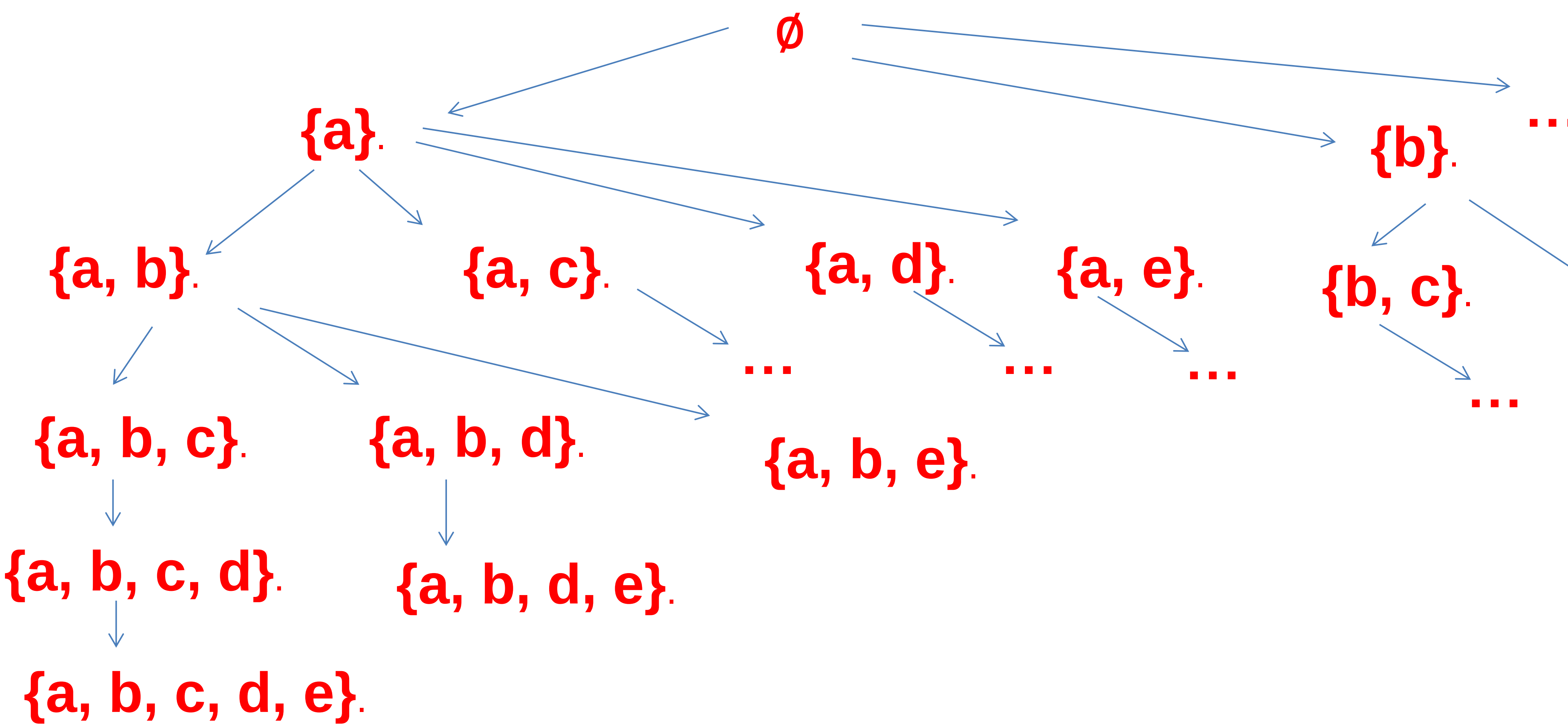
HUI-Miner(CIKM,2012)

It performs a **depth-first search** by appending items to itemsets.



HUI-Miner(CIKM,2012)

It performs a **depth-first search** by appending items to itemsets.



HUI-Miner(2012)

Creates a vertical structure named ***Utility-List*** for **each item.**

Trans.	items
T_0	a(1), b(5), c(1), **d**(3), (e,1)
T_1	b(4), c(3), **d**(3), e(1)
T_2	a(1), c(1), **d**(1)
T_3	a(2), c(6), e(2)
T_4	b(2), c(2), e(1)

item	unit profit
a	5 $
b	2 $
c	1 $
d	2 $
e	3 $

Example: The utility-list of **{d}**:

Trans.	util	rutil
T_0	6	3
T_1	6	3
T_2	2	0

HUI-Miner(2012)

Creates a vertical structure named ***Utility-List*** for **each item.**

Trans.	items
T_0	a(1), b(5), c(1), **d**(3), (e,1)
T_1	b(4), c(3), **d**(3), e(1)
T_2	a(1), c(1), **d**(1)
T_3	a(2), c(6), e(2)
T_4	b(2), c(2), e(1)

item	unit profit
a	5 $
b	2 $
c	1 $
d	2 $
e	3 $

Example: The utility-list of **{d}**:

Trans.	util	rutil
T_0	6	3
T_1	6	3
T_2	2	0

The first column is the **list of transactions** containing the itemset

HUI-Miner(2012)

Creates a vertical structure named ***Utility-List*** for **each item.**

Trans.	items
T_0	a(1), b(5), c(1), **d**(3), (e,1)
T_1	b(4), c(3), **d**(3), e(1)
T_2	a(1), c(1), **d**(1)
T_3	a(2), c(6), e(2)
T_4	b(2), c(2), e(1)

item	unit profit
a	5 $
b	2 $
c	1 $
d	2 $
e	3 $

Example: The utility-list of **{d}**:

Trans.	util	rutil
T_0	6	3
T_1	6	3
T_2	2	0

The second column is the **utility** of **the itemset** in these transactions

HUI-Miner(2012)

Creates a vertical structure named ***Utility-List*** for **each item.**

Trans.	items
T_0	a(1), b(5), c(1), **d**(3), (e,1)
T_1	b(4), c(3), **d**(3), e(1)
T_2	a(1), c(1), **d**(1)
T_3	a(2), c(6), e(2)
T_4	b(2), c(2), e(1)

item	unit profit
a	5 $
b	2 $
c	1 $
d	2 $
e	3 $

Example: The utility-list of **{d}**:

Trans.	util	rutil
$\mathbf{T_0}$	6	3
$\mathbf{T_1}$	6	3
$\mathbf{T_2}$	2	0

Property 1. The sum of the second column gives the utility of **the itemset.**

u({d}) = 6+6+2 = 14 $

HUI-Miner(2012)

Creates a vertical structure named ***Utility-List*** for **each item.**

Trans.	items
T_0	a(1), b(5), c(1), **d**(3), e,1)
T_1	b(4), c(3), **d**(3), e(1)
T_2	a(1), c(1), **d**(1)
T_3	a(2), c(6), e(2)
T_4	b(2), c(2), e(1)

item	unit profit
a	5 $
b	2 $
c	1 $
d	2 $
e	3 $

Example: The utility-list of **{d}**:

Trans.	util	rutil
T_0	6	3
T_1	6	3
T_2	2	0

The third column is the ***remaining utility***, that is utility of items appearing after the itemset in the transactions.

HUI-Miner(2012)

Creates a vertical structure named ***Utility-List*** for **each item.**

Trans.	items
T_0	a(1), b(5), c(1), **d**(3), e,1)
T_1	b(4), c(3), **d**(3), e(1)
T_2	a(1), c(1), **d**(1)
T_3	a(2), c(6), e(2)
T_4	b(2), c(2), e(1)

item	unit profit
a	5 $
b	2 $
c	1 $
d	2 $
e	3 $

Example: The utility-list of **{d}**:

Trans.	util	rutil
T_0	6	3
T_1	6	3
T_2	2	0

Property 2: The sum of all numbers is an upper bound on the utility of **the itemset** and its extensions.

6+6+2+3+3+0 = 20 $

HUI-Miner(2012)

Utility-lists can be *joined* to calculate utility-lists of larger itemsets

Utility list of {a}

Trans.	util	rutil
T_0	5	20
T_2	5	3
T_3	10	12

+ join

Utility list of {d}

Trans.	util	rutil
T_0	6	3
T_1	3	5
T_2	2	0

→

Utility list of {a, d}

Trans.	util	rutil
T_0	11	3
T_2	7	0

u({a}) = 20 $

u({e}) = 11 $

u({a,d}) = 18 $

HUI-Miner(2012)

Utility-lists can be *joined* to calculate utility-lists of larger itemsets

Utility list of {a}

Trans.	util	rutil
T_0	5	20
T_2	5	3
T_3	10	12

join

Utility list of {d}

Trans.	util	rutil
T_0	6	3
T_1	3	5
T_2	2	0

Utility list of {a, d}

Trans.	util	rutil
T_0	11	3
T_2	7	0

u({a}) = 20 $

u({e}) = 11 $

u({a,d}) = 18 $

HUI-Miner(2012)

Utility-lists can be *joined* to calculate utility-lists of larger itemsets

Utility list of {a}

Trans.	util	rutil
T_0	5	20
T_2	5	3
T_3	10	12

+ join

Utility list of {d}

Trans.	util	rutil
T_0	6	3
T_1	3	5
T_2	2	0

→

Utility list of {a, d}

Trans.	util	rutil
T_0	11	3
T_2	7	0

u({a}) = 20 $

u({e}) = 11 $

u({a,d}) = 18 $

HUI-Miner(2012)

Utility-lists can be *joined* to calculate utility-lists of larger itemsets

Utility list of {a}

Trans.	util	rutil
T_0	5	20
T_2	5	3
T_3	10	12

+ join

Utility list of {d}

Trans.	util	rutil
T_0	6	3
T_1	3	5
T_2	2	0

→

Utility list of {a, d}

Trans.	util	rutil
T_0	11	3
T_2	7	0

u({a}) = 20 $

u({e}) = 11 $

u({a,d}) = 18 $

HUI-Miner(2012)

- **HUI-Miner** (2012) can be up to two orders of magnitude faster than previous algorithms (e.g. **UPGrowth**).
- But High-Utility itemset mining remains very costly in terms of execution time and memory.
- We still need faster algorithms.

Our solution

FHM – A faster algorithm (ISMIS **2014**)

Observation:

- the main performance bottleneck of HUI-Miner is the **join** operations.
- **Join operations** are very costly in terms of execution time

Can we reduce the number of join operations?

Our solution

FHM – A faster algorithm (ISMIS **2014**)

- We propose a mechanism named ***Estimated-Utility Co-occurrence pruning.***
- First, we pre-calculate the TWU of all pairs of items and store it in a structure named EUCS.

	a	b	c	d
b	25			
c	55	54		
d	33	45	53	
e	47	54	76	45

EUCS can be implemented as
(1) a **triangular matrix** or
(2) a hashmap of hashmaps

Our solution

FHM – A faster algorithm (ISMIS **2014**)

- Then, during the search, consider that we need to calculate the utility list of an itemset **X.**
- If **X** contains a pair of items **i** and **j** such that TWU({i,j}) < minutil, then **X** is low utility as well as all its extensions.
- In this case, we can avoid performing the join.

	a	b	c	d
b	25			
c	55	54		
d	33	45	53	
e	47	54	76	45

Experimental Evaluation

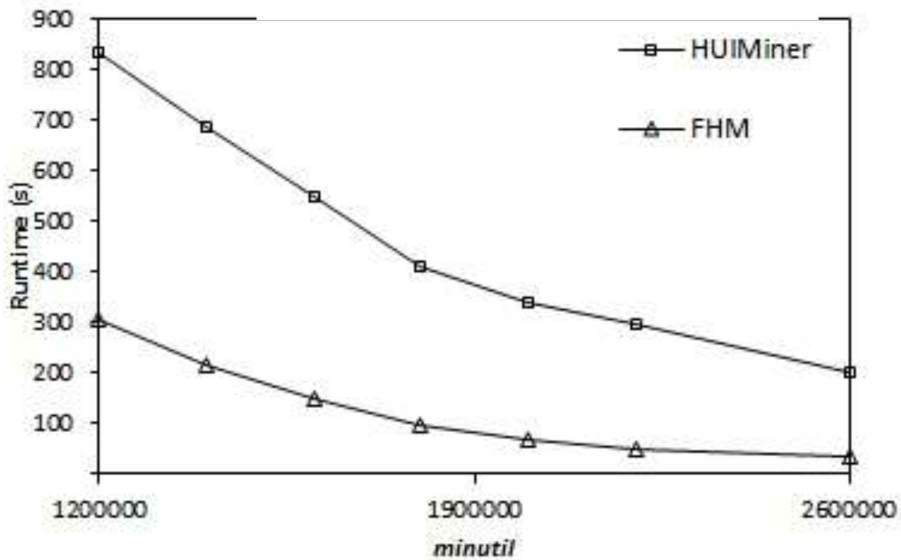
Datasets

Dataset	transaction count	distinct item count	avg. trans. length
Chainstore	1,112,949	46,086	7.26
BMS	59,601	497	4.85
Kosarak	990,000	41,270	8.09
Retail	88,162	16,470	10.30
Chess	3,396	75	37

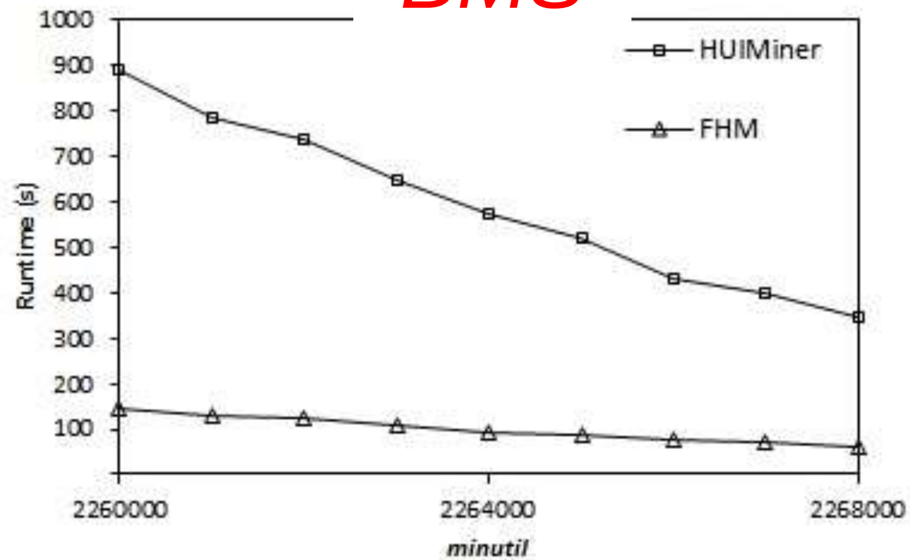
- Chainstore has real unit profit/quantity values
- Other datasets: unit profit between 1 and 1000 and quantities between 1 and 5 (normal distribution)
- FHM vs HUI-Miner
- Java, Windows 7, 5 GB of RAM

Execution times

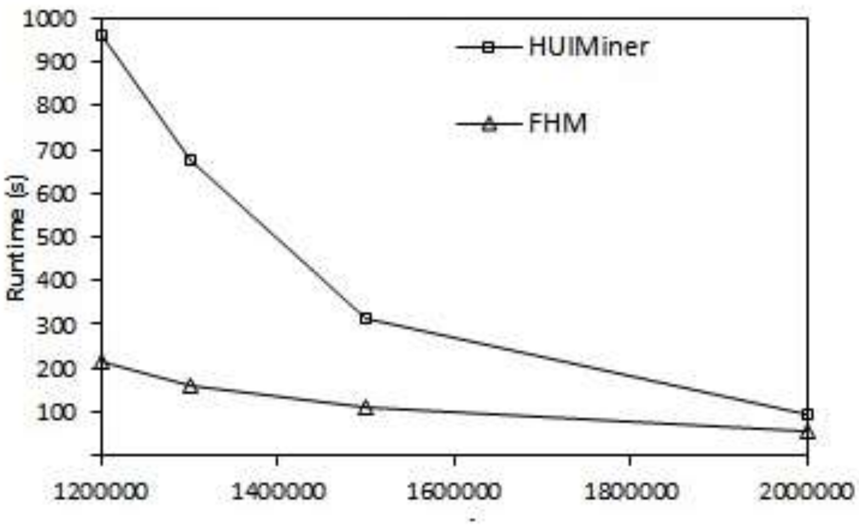
Chainstore



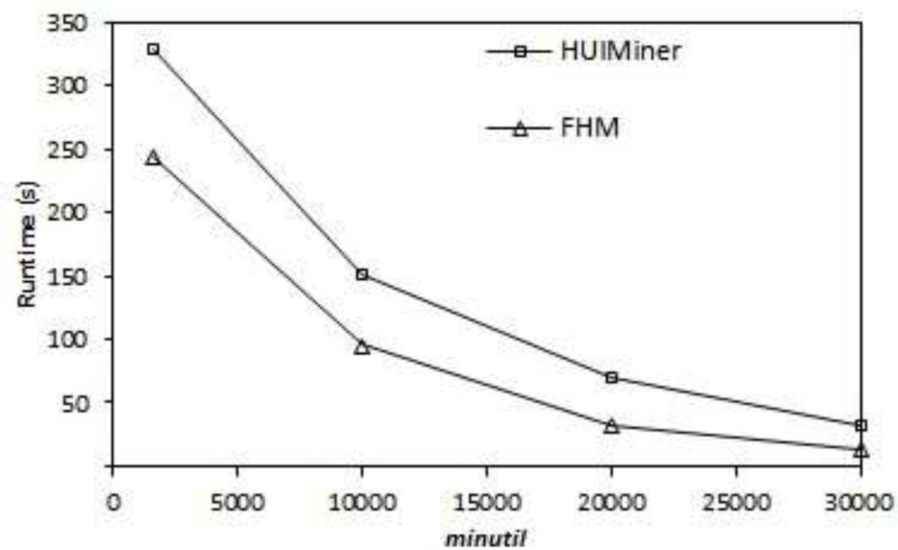
BMS



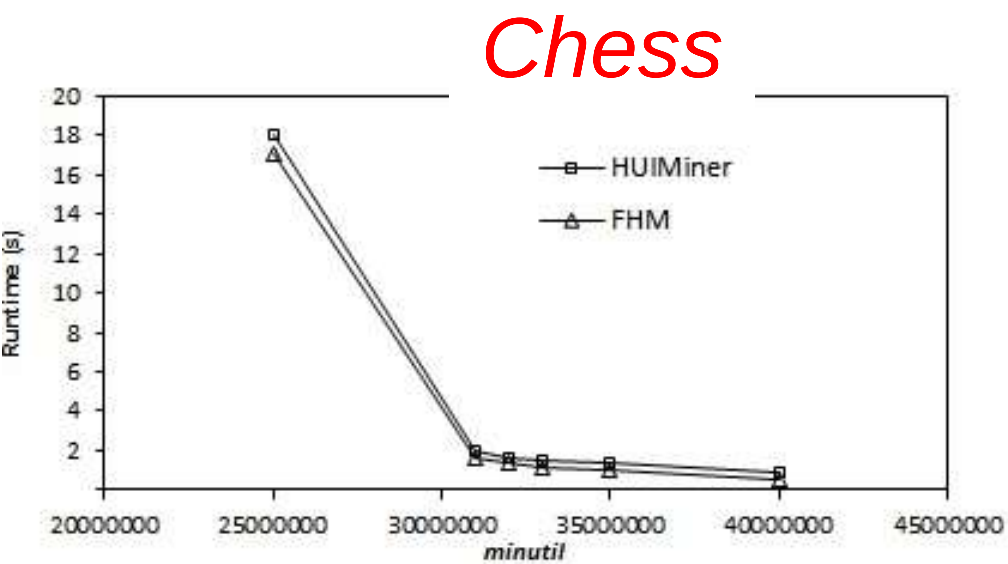
Kosarak

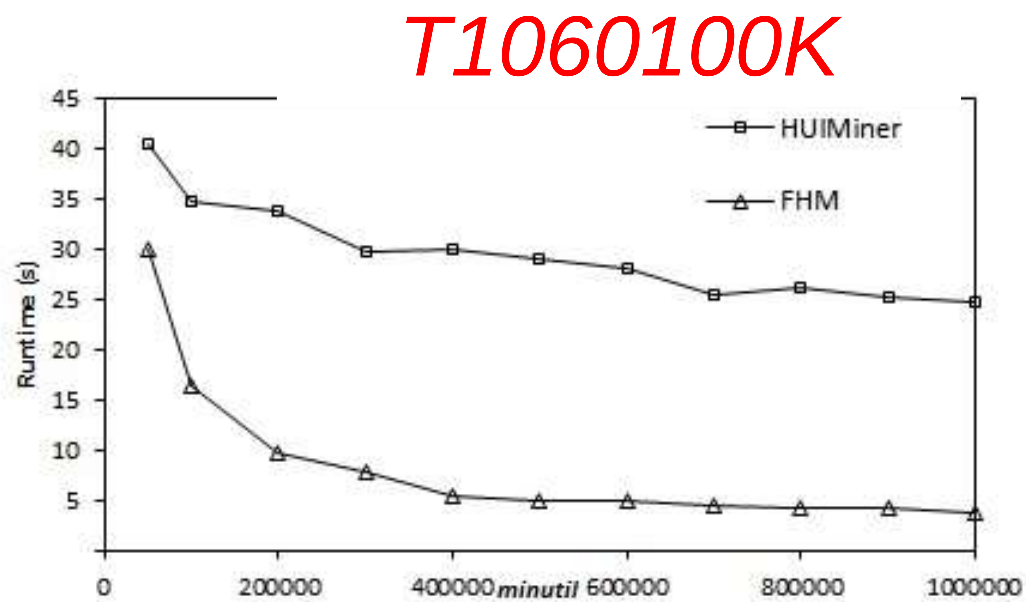


Retail



Execution times (cont'd)





Overall:

- **FHM** has the **best performance** on all datasets
- **FHM** is **up to 6 times faster** than HUI-Miner
- Performance is similar to HUI-Miner for extremely dense datasets (e.g. **Chess**) because each items co-occurs with each other in almost all transactions.

Pruning effectiveness

- How many joint operations are avoided by FHM?
- A large amount. For example:
  - Chainstore : 18 %
  - BMS : 91 %
  - Kosarak : 87 %
  - Retail : 87 %

Memory overhead

- The memory footprint of the EUCS structure is small.
- For example:
  - Chainstore: 10.3 MB
  - BMS: 4.18 MB
  - Kosarak: 1.19 MB
  - Retail: 410 MB

The FHN algorithm

Fournier-Viger, P. (2014). FHN: Efficient Mining of High-Utility Itemsets with Negative Unit Profits. Proc. 10th International Conference on Advanced Data Mining and Applications (ADMA 2014), Springer LNCS 8933, pp. 16-29.

Another important problem

In high utility mining:

- Items **are not allowed to have *negative unit profit*.**
- **But in real-life transaction databases, items are often sold at a loss**.

What happens if we apply the algorithms on such database?

The *TWU* is no longer an upper bound.

Trans.	items
T_0	a(1), b(5), **c(1**), d(3), (e,1)
T_1	b(4), c(3), d(3), e(1)
T_2	a(1), c(1), d(1)
T_3	a(2), **c(6),** e(2)
T_4	b(2), c(2), e(1)

item	unit profit
a	5 $
b	2 $
c	**-5 $**
d	2 $
e	3 $

u({a,d}) = 24 $ TWU({a,d}) = 19 – 14 = 5 $

Thus, u({a,d,e}) =14 $ may <u>not</u> be found!

HUINIV-Mine (2009)

- **HUINIV-Mine solves this problem.**
- **How ?** it *excludes items having a negative profit* from the *TWU* calculation
- Thus, the *TWU* becomes again an upper bound on utility.
- **However, HUINIV-Mine is not efficient**
  - Based on Apriori, it keeps huge amount of candidates in memory,
  - the TWU upper bound is too loose,
  - scanning database in Phase 2 is very slow

Our solution
FHN (ADMA **2014**)

- **FHN**: **F**ast **H**igh utility mining with **Negative unit profit**
- Extends the **FHM** algorithm for high utility itemset mining.
- Integrate new ideas to handle negative items more efficiently.

The challenge

- FHM becomes an incomplete algorithm when negative unit profit are introduced.
  (it may not find some high utility itemsets)

- **Reason**: the *remaining utility* in utility-list may become negative because of negative items.

For example:

Trans.	items
T_0	**a(1),** b(5), **c(1**), d(3), (e,1)
T_1	b(4), c(3), d(3), e(1)
T_2	**a(1),** c(1), d(1)
T_3	**a(2)**, **c(6),** e(2)
T_4	b(2), c(2), e(1)

item	unit profit
a	5 $
b	- 2 $
c	**-100$**
d	2 $
e	3 $

Utility list of {a}

Trans.	util	rutil
T_0	5	**-101**
T_2	5	**-98**
T_3	10	**-594**

5 + 5 - 10 – 81 – 98 – 594 = – **763**

Thus, no extensions of **{a}** such as **{a,d}** will be explored!

Our solution in FHN

New idea 1: not include negative items in the calculation of the *remaining utility* in utility lists.

Utility list of {a}

Trans.	util	rutil
T_0	5	**-101**
T_2	5	**-98**
T_3	10	**-594**

becomes →

Utility list of {a}

Trans.	util	rutil
T_0	5	**19**
T_2	5	**2**
T_3	10	**6**

Our solution in **FHN**

<u>New idea 2</u>: separate the positive and negative utility in two columns.

Utility list of {a,b}

Trans.	util	rutil
T_0	**-5**	-81

becomes

Utility list of {a,b}

Trans.	+util	-util	rutil
T_0	**5**	**-10**	19

Our solution in FHN

<u>New idea 3</u>: we fix the pruning property.

Utility list of {a,b}

Trans.	util	rutil
T_0	**-5**	-81

becomes

Utility list of {a,b}

Trans.	+util	-util	rutil
T_0	**5**	**-10**	19

Pruning property 3 : The sum of the **"+util"** and "rutil" column is an upper bound on the utility of **the itemset** and its extensions.

Our solution in **FHN**

Lastly:

- In the EUCS structure, we do not include negative items in the TWU calculation.
- Use the EUCP strategy only for items with positive unit profit.

Experimental Evaluation

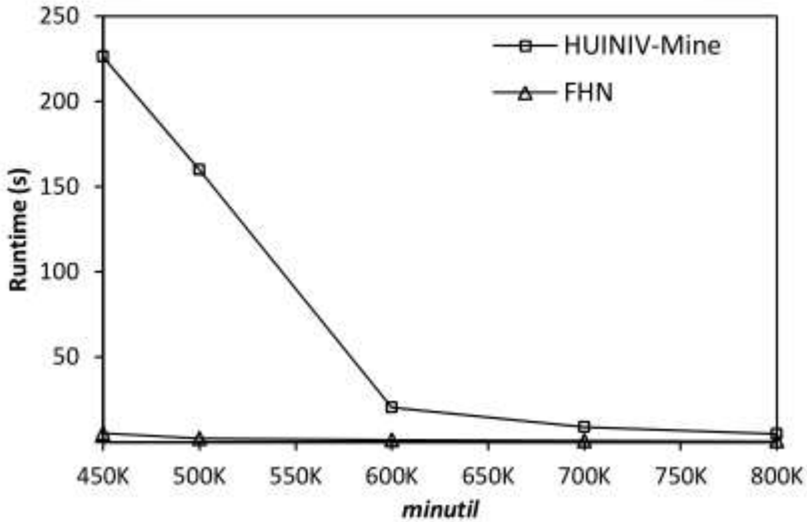
Six datasets

Dataset	trans. count	distinct item count	avg. trans. length
Mushroom	88,162	16,470	23
Accidents	340,183	468	33.8
Kosarak	990,000	41,270	8.09
Retail	88,162	16,470	10.30
Chess	3,396	75	37
Psumb	49,046	7,116	74

- Unit profit in [-1000, 1000] (normal distribution)
- Quantities in [1, 5] (normal distribution)
- FHN vs HUINIV-Mine
- Java, Windows 7, 5 GB of RAM

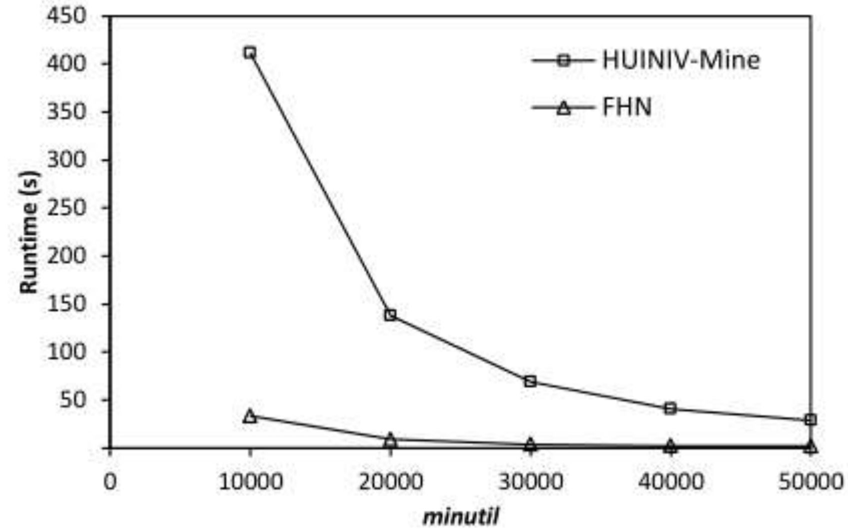
Execution time

Mushroom



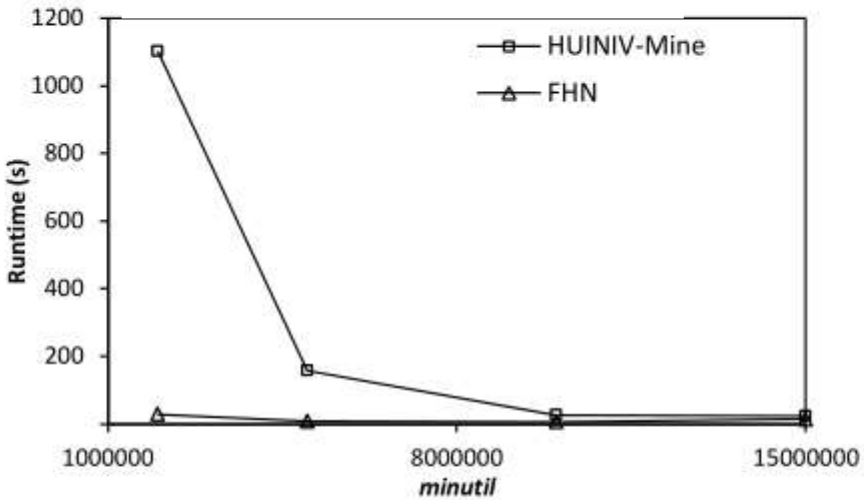
up to **42 times** faster

Retail



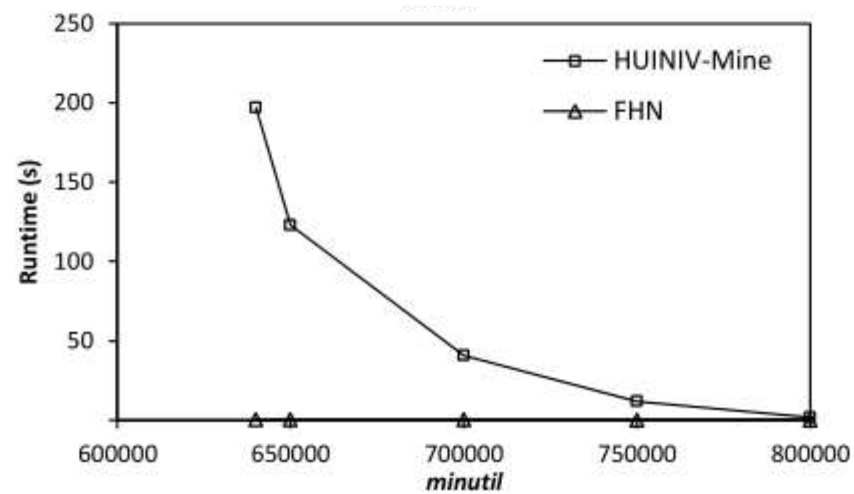
up to **18 times** faster

Kosarak



up to **38 times** faster

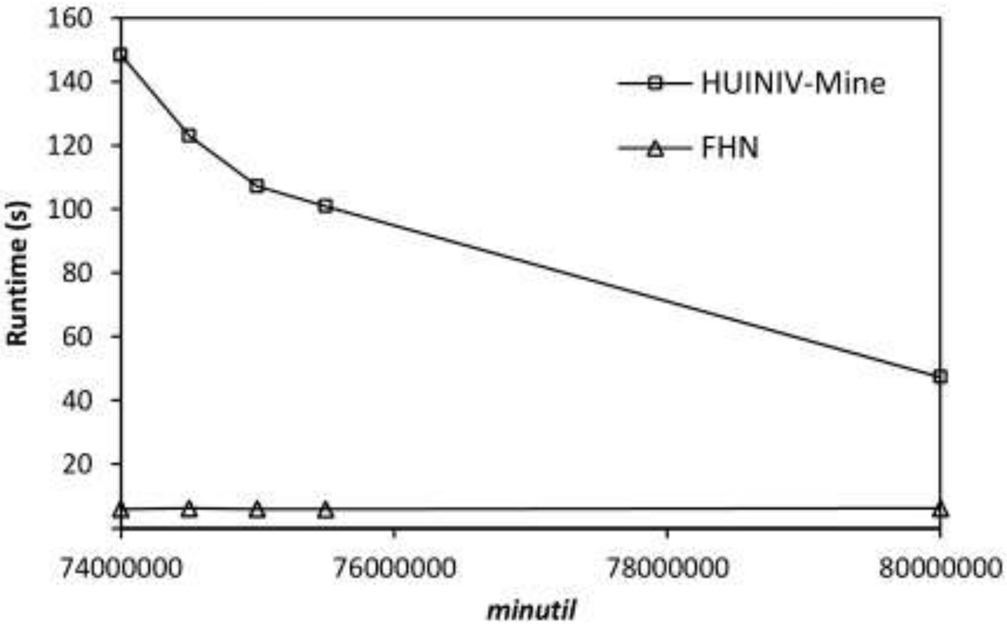
Chess



up to **500 times faster**

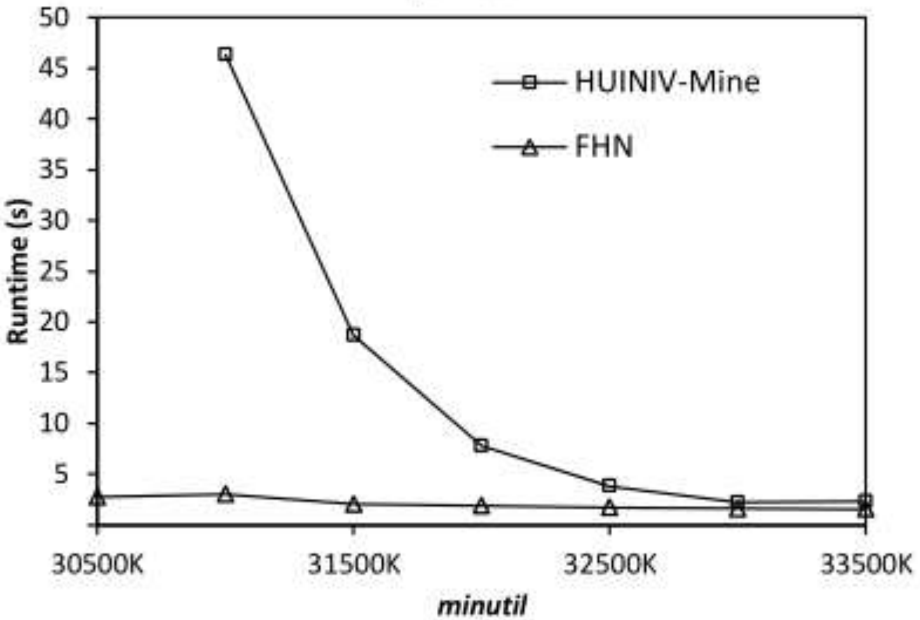
Execution time (cont'd)

Accidents



up to **15 times** faster

Psumb



up to **25 times** faster

Memory Usage

Dataset	HUINIV-Mine	FHN
Kosarak	> 5 GB	20 MB
Chess	> 5 GB	1179 MB
Psumb	> 5 GB	100 MB
Accidents	> 5 GB	350 MB
Mushroom	4.97 GB	250 MB
Retail	--	five times less

FHN uses **up to 250 times less memory!**

Why FHN performs better?

- **FHN** prunes the search space using EUCP and the remaining utility, while **HUINIV-Mine** only uses TWU.
- **FHN** uses a depth-first search and mine HUIs using a single phase, while **HUINIV-Mine** generate candidates and uses two phases

The FOSHU algorithm

Fournier-Viger, P., Zida, S. (2015). FOSHU: Faster On-Shelf High Utility Itemset Mining– with or without negative unit profit. Proc. 30th Symposium on Applied Computing (ACM SAC 2015). ACM Press, pp. 857-864

Another important problem

High utility mining:

- **Does not consider the shelf time of items.**
- In real-life, some items are only sold during specific **time periods** (e.g. summer).

High utility mining
is biased toward

items with long shelf-time
(they have more opportunity to generate a higher profit)

Representing Time Periods

Time periods can be represented in a database.

TID	items	Time Period
T_1	(a,1)(c,1)(d,1)	1
T_2	(a,2)(c,6)(e,2)(g,5)	1
T_3	(a,1)(b,2)(c,1)(d,6)(e,1)(f,5)	2
T_4	(b,4)(c,3)(d,3)(e,1)	2
T_5	(b,2)(c,2)(e,1)(g,2)	3

item	unit profit
a	-5 $
b	2 $
c	1 $
d	2 $
e	3 $
f	1 $
g	1 $

E.g. 1 = spring 2 = summer 3 = autumn

Utility of a Time Period

TID	items	Time Period
T_1	(a,1)(c,1)(d,1)	1
T_2	(a,2)(c,6)(e,2)(g,5)	1
T_3	(a,1)(b,2)(c,1)(d,6)(e,1)(f,5)	2
T_4	(b,4)(c,3)(d,3)(e,1)	2
T_5	(b,2)(c,2)(e,1)(g,2)	3

item	unit profit
a	-5 $
b	2 $
c	1 $
d	2 $
e	3 $
f	1 $
g	1 $

Utility of a time period

(the total profit generated during the time period)

UT(1) = (-5 + 1 + 2)+ (-10+6+6+5) = 5 $

UT(2) = (-5+12 +1+12+3+5)+(8+3+6+3) = 48 $

UT(3) = (4+2+3+2) = 11 $

The Problem of On-Shelf High Utility Itemset Mining

Let be a user-defined threshold *minUtil* *in* [0,1]

For example: *minUtil* = 0.60

TID	items	Time Period
T_1	(a,1)(c,1)(d,1)	1
T_2	(a,2)(c,6)(e,2)(g,5)	1
T_3	(a,1)(b,2)(c,1)(d,6)(e,1)(f,5)	2
T_4	(b,4)(c,3)(d,3)(e,1)	2
T_5	(b,2)(c,2)(e,1)(g,2)	3

item	unit profit
a	-5 $
b	2 $
c	1 $
d	2 $
e	3 $
f	1 $
g	1 $

The goal: find all itemsets X such that :

$$\text{relative\_utility(X)} = \frac{(profit\ of\ X)}{(profit\ of\ time\ periods\ where\ X\ was\ sold)} \geq minutil$$

An Example

TID	items	Time Period
T_1	(a,1)(c,1)(d,1)	1
T_2	(a,2)(c,6)(e,2)(g,5)	1
T_3	(a,1)(b,2)(c,1)(d,6)(e,1)(f,5)	2
T_4	(b,4)(c,3)(d,3)(e,1)	2
T_5	(b,2)(c,2)(e,1)(g,2)	3

item	unit profit
a	-5 $
b	2 $
c	1 $
d	2 $
e	3 $
f	1 $
g	1 $

Suppose that
***minutil* = 0.6** →

On-Shelf High utility itemsets

{b, e, f} 0.81,
{b,c, e, g} 1.0,
{b,c,g} 0.72,
{c,e,g} 0.77,
{b,d} 0.67,
{b,d,e} 0.8,
{b,c,d,e} 0.89,
{b, c,d} 0.75,

A Difficult Task!

- The *relative utility* is still **not** anti-monotonic.
- **Example**:

ru({b,d}) = 30 / 48 = 0.62

ru({b,c, d}) = 34 / 48 = 0.70

ru({b,d,e,f}) = 24 / 48 = 0.50

TS-HOUN(2014)

- A **three phase** breadth-first search algorithm
  1) Finds candidate high utility-itemset in each time period by using the Apriori candidate generation procedure.
  2) Perform the union of candidates in each period.
  3) Scans database to calculate the utility of candidates. Output those with **relative utility ≥ minutil**

Our Proposal

- **FOSHU**: **F**ast **O**n-**S**helf **H**igh-**U**tility mining with **Negative unit profit**
- Extends the **FHM** (2014) search procedure for high utility itemset mining.
- Adds new ideas to efficiently handle time periods

How to handle time periods?

- **Idea:** We add a « period » column to each utility-list.

Utility list of {a}

TID	+util	-util	rutil	period
T_1	0	-5	3	1
T_2	0	10	17	1
T_3	0	-5	25	2

- **Pruning property**: if the sum of « +util » and « rutil » column is less than *minutil* in each time period, the itemset can be pruned, as well as its extensions.
- We mine all time periods at the same time.

Experimental Evaluation

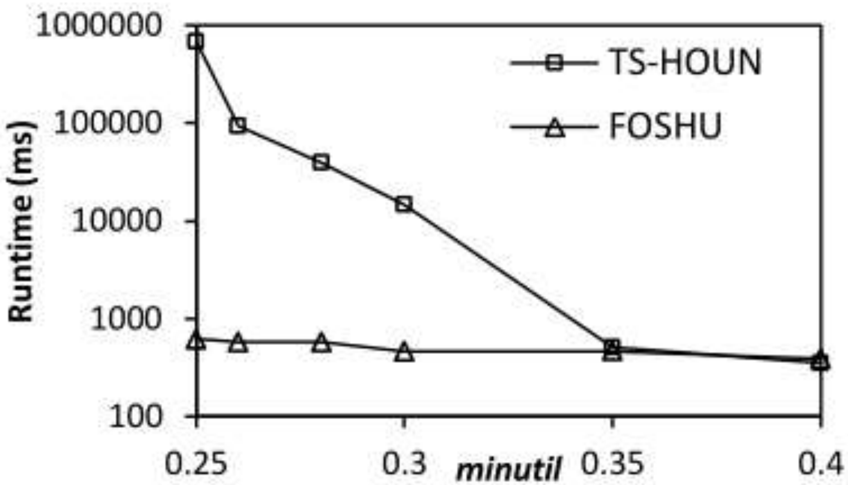
Five datasets

Dataset	transaction count	distinct item count	avg. transaction length
Mushroom	88,162	16,470	23
Accidents	340,183	468	33.8
Retail	88,162	16,470	10.30
Chess	3,396	75	37
Psumb	49,046	7,116	74

- Unit profit between -1000 and 1000 and quantities between 1 and 5 (normal distribution)
- FOSHU vs TS-HOUN
- Java, Windows 7, 5 GB of RAM

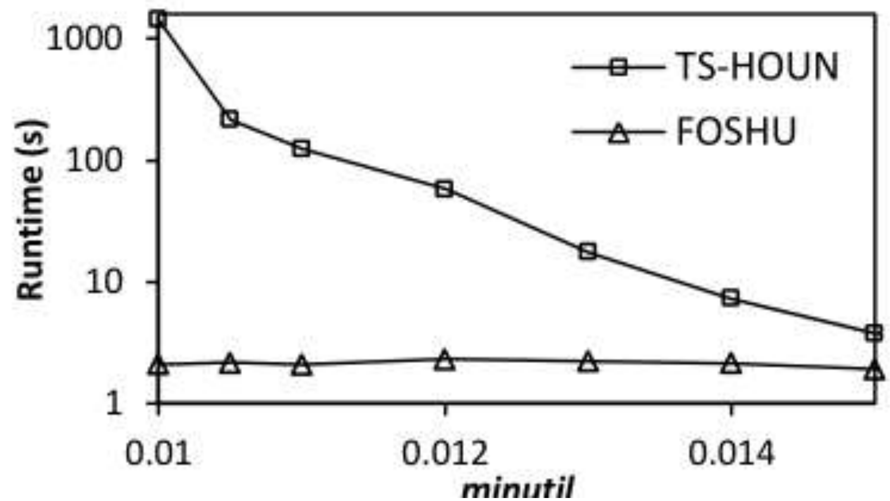
Influence of *minutil* on runtime

Mushroom



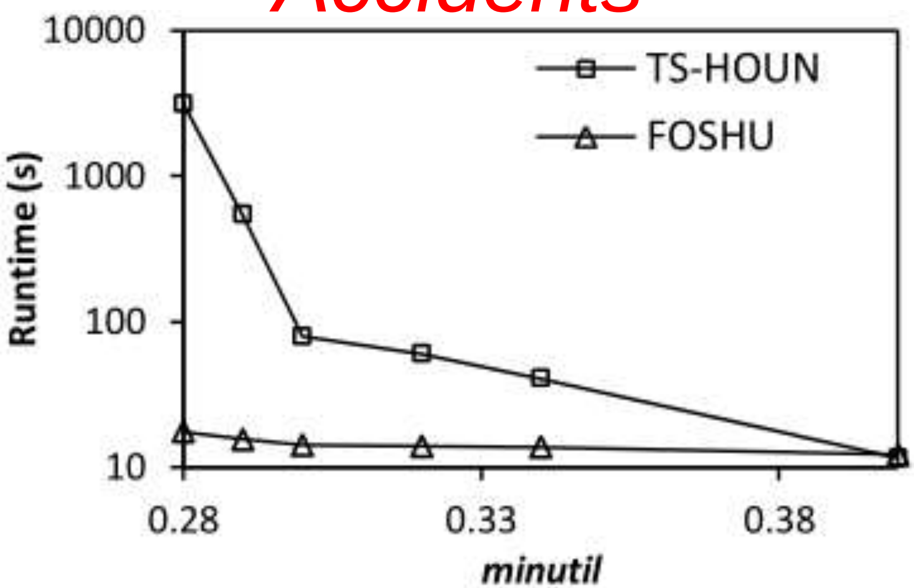
up to **1000** times faster

Retail



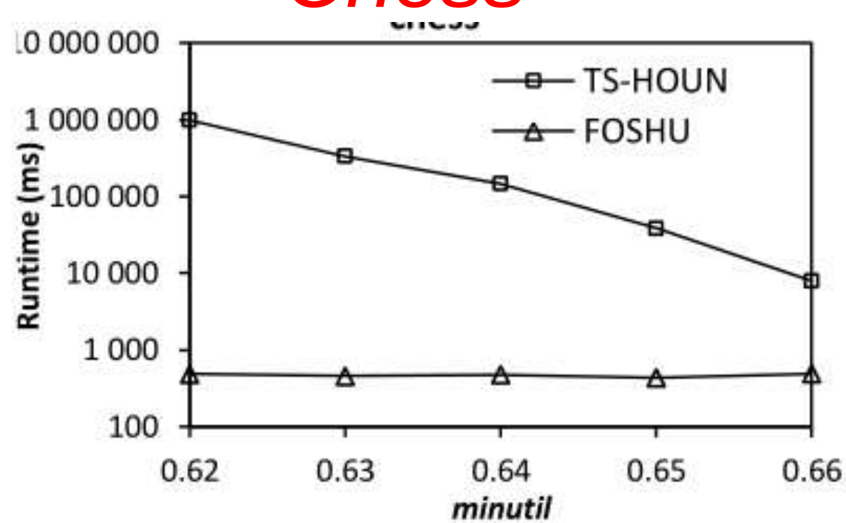
up to **683** times faster

Accidents



up to **178** times faster

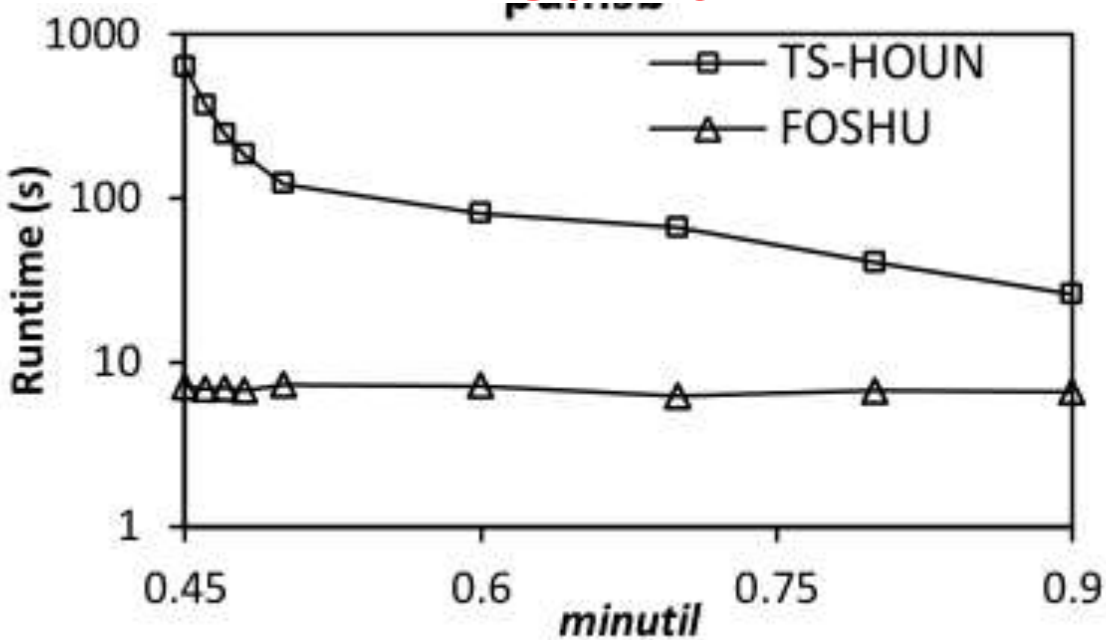
Chess



up to **2000** times faster

Influence of *minutil* on runtime (cont’d)

Psumb



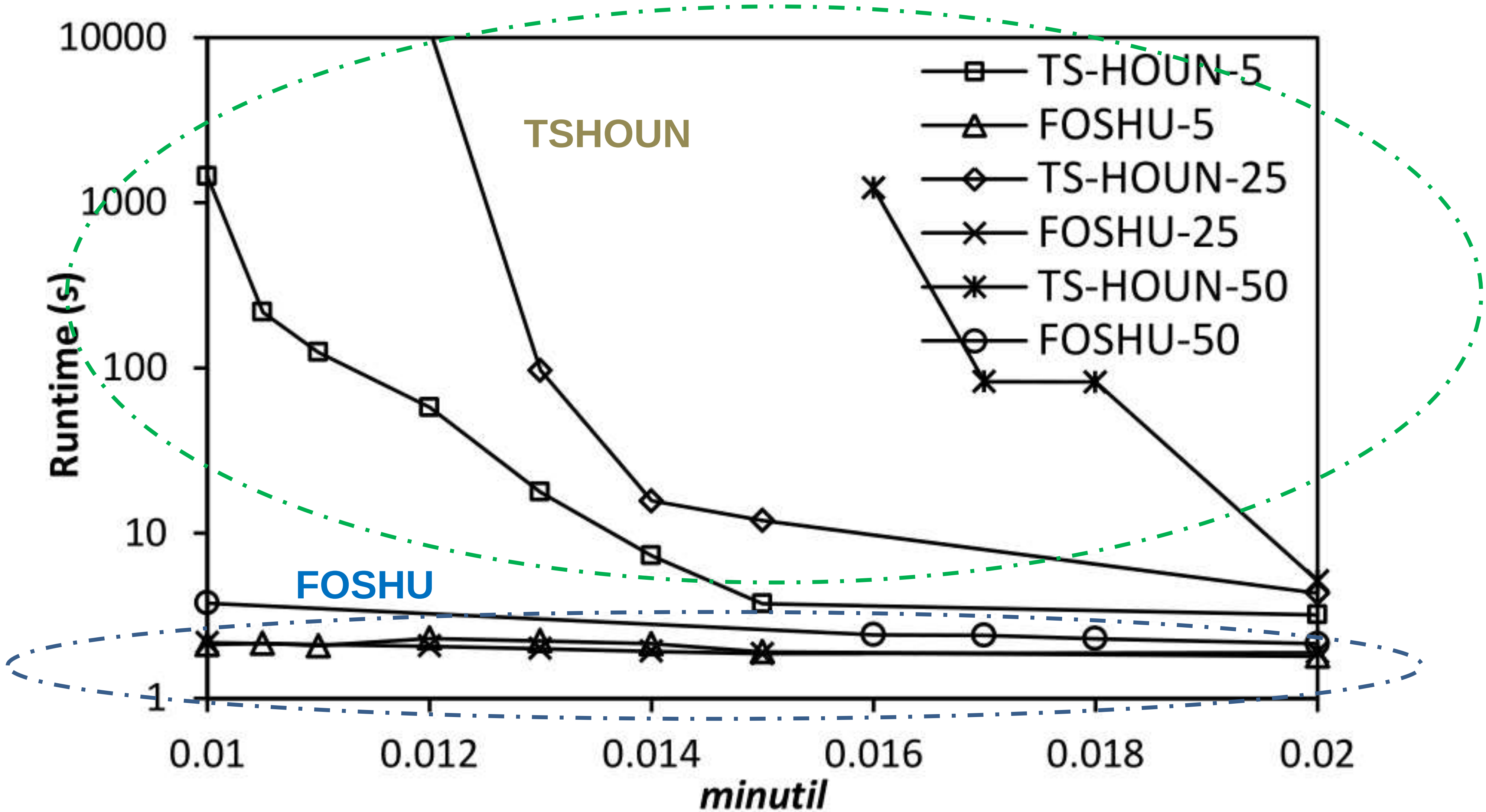
up to **89** times faster

Memory Usage (MB)

Dataset	TS-HOUN	FOSHU
Mushroom	69	**39**
Retail	571	**539**
Accidents	139	**14**
Chess	602	**498**
Pumsb	123	**98**

→ **FOSHU** uses **up to 10 times less memory**

Influence of the number of time periods



Influence of the number of transactions

Scalability of FOSHU w.r.t database size

dataset	25%	50%	75%	100%
mushroom	0.6	0.6	0.6	0.6
retail	2.1	2.1	2.1	2.1
accidents	9.3	9.5	9.8	10.2
chess	0.3	0.3	0.3	0.3
pumsb	4.1	4.1	4.2	4.2

Why FOSHU performs better?

- **FOSHU** uses TWU pruning and utility-list pruning, while **TS-HOUN** only uses TWU pruning.
- **FOSHU** uses a depth-first search and mine HUIs using a single phase, while **TS-HOUN** generate candidates and uses three phases

Other interesting problems

- **Closed high utility itemset mining**:
  - To discover a subset of all high-utility itemsets that is lossless, faster to discover, and requires less memory (ICDM 2011)
- **Generator of high utility itemsets:**
  - To discover another subset of high utility itemsets (ADMA 2014 – best paper)
- **Top-K high utility itemset mining**:
  - Discover the *K* itemsets generating the highest profit.

Other interesting problems

- **High utility sequential rule mining**
  - we are currently working on an algorithm to discover sequential rules with profit.
    (e.g. *A→B* generates a high profit and has a probability of 60% of happening)
- **… and many others!**

SPMF

An Open-Source Data Mining Library

Introduction

Algorithms

Download

Documentation

Datasets

FAQ

License

Contributors

Citations

Performance

Forum

Mailing-list

Introduction

SPMF is an **open-source data mining mining library** written in **Java**, specialized in **pattern mining**.

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

It offers implementations of **82 data mining algorithms** for:

- **sequential pattern mining,**
- **association rule mining,**
- **itemset mining,**
- **sequential rule mining,**
- **clustering**.

The source code of each algorithm can be integrated in other Java software.

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

The current version is **v0.96r2** and was released the **30th November 2014.**

http://www.phillippe-fournier-viger.com/spmf/

Conclusion

We have presented three algorithms for high utility itemset mining:

- **FHM**: to mine high utility itemsets
- **FHN**: to mine high utility itemsets in the case of negative and positive unit profit
- FOSHU: to mine high utility itemsets in the case of negative and positive unit profit, and considering shelf time

Thank you. Questions?

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

Some references

- Fournier-Viger, P. (2014). FHN: Efficient Mining of High-Utility Itemsets with Negative Unit Profits. Proc. 10th International Conference on Advanced Data Mining and Applications (ADMA 2014), Springer LNCS 8933, pp. 16-29.
- Fournier-Viger, P., Wu, C.-W., Zida, S., Tseng, V. S. (2014) FHM: Faster High-Utility Itemset Mining using Estimated Utility Co-occurrence Pruning. Proc. 21st International Symposium on Methodologies for Intelligent Systems (ISMIS 2014), Springer, LNAI, pp. 83-92.
- Fournier-Viger, P., Wu, C.W., Tseng, V.S. (2014). Novel Concise Representations of High Utility Itemsets using Generator Patterns. Proc. 10th International Conference on Advanced Data Mining and Applications (ADMA 2014), Springer LNCS 8933, pp. 30-43.
- Fournier-Viger, P., Zida, S. (2015). FOSHU: Faster On-Shelf High Utility Itemset Mining– with or without negative unit profit. Proc. 30th Symposium on Applied Computing (ACM SAC 2015). ACM Press, pp. 857-864
- Tseng, V. S., Shie, B.-E., Wu, C.-W., Yu., P. S.: Efficient Algorithms for Mining High Utility Itemsets from Transactional Databases. In: IEEE Trans. Knowl. Data Eng. 25(8), pp.~1772--1786 (2013)
- Liu, Y., Liao, W., Choudhary, A.: A two-phase algorithm for fast discovery of high utility itemsets. In: Proc. 9th Pacific-Asia Conf. on Knowledge Discovery and Data Mining, pp. 689--695 (2005)
- Wu, C.-W., Lin, Y.-F., Yu, P. S., Tseng, V. S.: Mining High Utility Episodes in Complex Event Sequences. In: Proc. KDD'2013, pp.~536--544 (2013)
- Ahmed, C. F., Tanbeer, S. K., Jeong, B.-S., Lee, Y.-K.: Efficient Tree Structures for High-utility Pattern Mining in Incremental Databases. In: IEEE Trans. Knowl. Data Eng. 21(12), pp.~1708--1721 (2009)
- Yin, J., Zheng, Z., Cao, L.: USpan: An Efficient Algorithm for Mining High Utility Sequential Patterns. In: Proc. ACM SIG KDD'12, pp.~660--668 (2012)

and more…

- Yin, J., Zheng, Z., Cao, L., Song, Y., Wei, W.: Efficiently Mining Top-K High Utility Sequential Patterns. In: Proceedings of ICDM'13, pp.~1259--1264 (2013)
- Yin, J., Zheng, Z., Cao, L.: USpan: An Efficient Algorithm for Mining High Utility Sequential Patterns. In: Proceedings of ACM SIG KDD'12, pp.~660--668 (2012)
- Wu, C.-W., Lin, Y.-F., Yu, P. S., Tseng, V. S.: Mining High Utility Episodes in Complex Event Sequences. In: Proceedings of ACM SIG KDD'13, pp.~536--544 (2013)
- Shie, B.-E., Cheng, J.-H., Chuang, K.-T., Tseng, V. S.: A One-Phase Method for Mining High Utility Mobile Sequential Patterns in Mobile Commerce Environments. In: Proceedings of IEA/AIE'12, pp.~616--626 (2012)
- Chu, C.-J., Tseng, V. S., Liang, T.: An efficient algorithm for mining high utility itemsets with negative item values in large databases. In: Applied Math. Comput., 215, pp.~767-778 (2009)
- Wu, C.-W., Fournier-Viger, P., Yu., P. S, Tseng, V. S.: Efficient Mining of a Concise and Lossless Representation of High Utility Itemsets. In: Proc. 11th IEEE International Conference on Data Mining, pp.~824--833 (2011)