Mining Correlated High-Utility Itemsets using the Cosine Measure

Students: Huynh Anh Duy, Huynh Anh Khoa

Supervisor: Assoc. Prof. Phan Duy Hung

Agenda

- Introduction
- Algorithms
- Methodology
- Experiment and analyze
- Conclusion and perspectives

Introduction

Basic concepts

Problem definition

Related works and contribution

Basic concepts

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	Item
T1	{a, b, c, d, e}
T2	{a, b, e}
T3	{c, d, e}
T4	{a, b, d, e}

Problem Definition

Discovering Frequent Patterns

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

Transaction database

Transaction	Item
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
…	…

Problem Definition

How to solve this problem?

The naïve approach:

- Scan the database to count the frequency of **each** possible itemset.

  eg: {a}, {a,b}, {a,c}, {a,d}, {a, e}, {a,b,c}, {a,b,d}, …{b}, {b,c}, … {a,b,c,d,e}
- If **n** items, then $\mathbf{2^{n}-1}$ possible itemsets.
- Thus, inefficient.

Several efficient algorithms:

- Apriori, FPGrowth, H-Mine, LCM, etc.

Problem Definition

The “Apriori” property

Property (anti-monotonicity).
Let be itemsets X and Y. If X ⊂ Y**,** then the support of Y is less than or equal to the support of X.

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

Example
The support of {a,b} is 3.
Thus, supersets of {a,b} have support ≤ 3.

Problem Definition

Limitations of frequent patterns

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

Problem Definition

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 mile 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

Problem Definition

High Utility Itemset Mining

Input

A transaction database

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

A unit profit table

Item	a	b	c	d	e	f	g
Profit	5	2	1	2	3	1	1

minutil: a minimum utility threshold set by the user (a positive integer)

Output

All high-utility itemsets (itemsets having a *utility* $\geq$ *minutil*)

For example, if *minutil = 33$,* the high-utility itemsets are:

{b,d,e} 36$ 2 transactions	{b,c,d} 34$ 2 transactions
{b,c,d,e} 40$ 2 transactions	{b,c,e} 37$ 3 transactions

Problem Definition

Utility calculation

A transaction database

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

A unit profit table

Item	a	b	c	d	e	f	g
Profit	5	2	1	2	3	1	1

The **utility** of itemset {b,d,e} is calculated as follows:

$$u(\{b,d,e\}) = \underbrace{(5x2)+(3x2)+(3x1)}_{\text{Utility in transaction } T_1} + \underbrace{(4x2)+(2x3)+(1x3)}_{\text{Utility in transaction } T_2} = 36\$$$

Problem Definition

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 iemset mining algorithms cannot applied to this problem

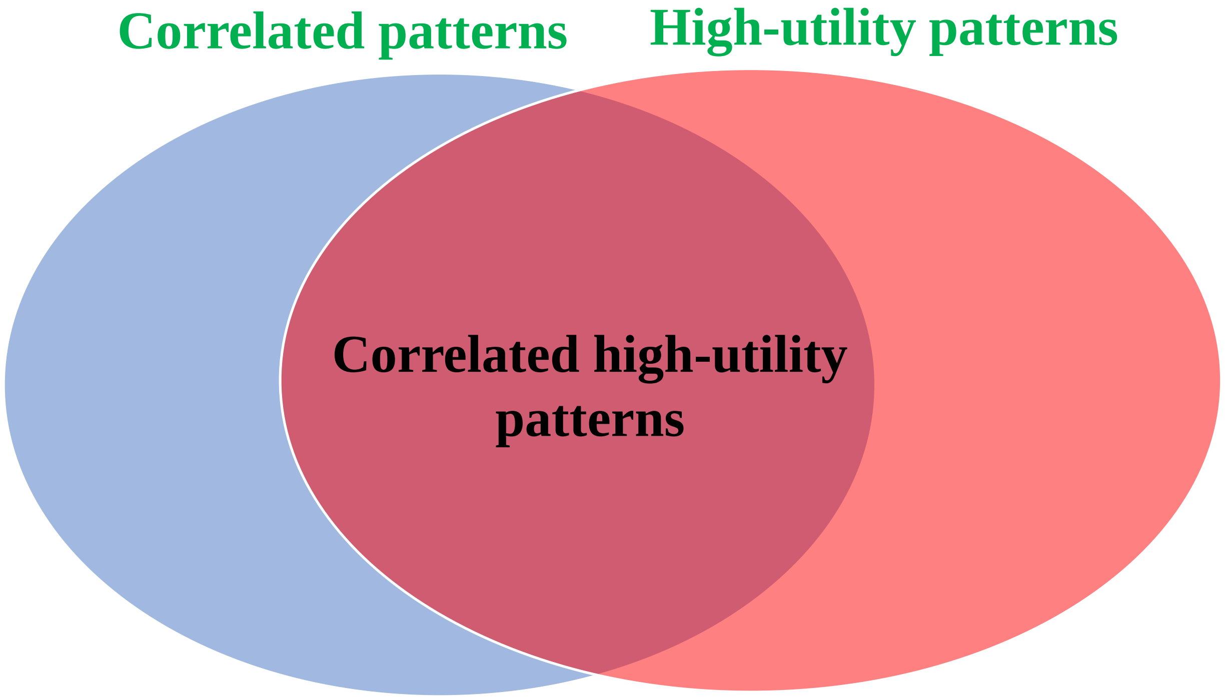
Problem Definition

Correlation problem

High-utility itemset mining

- Is useful for discovering profitable itemsets.
- But may discover many itemsets that are weakly correlated.
- E.g. bread with caviar has a high profit

We need a new type of patterns:



Related works and contribution

Solve high utility itemset mining problems

- **Algorithms**
  - Two-Phase (PAKDD 2005),
  - IHUP (TKDE 2010),
  - UP-Growth (KDD 2011),
  - HUI-Miner (CIKM 2012),
  - **FHM (ISMIS 2014)**,
  - EFIM (MICAI 2015),
  - mHUIMiner (PAKDD 2017)
- **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.

Related works and contribution

Solve correlated high utility itemset mining problems

- **Algorithms**
  - **FCHM (HAIS 2016)**
  - CoHUIM (Knowledge-Based Systems 2018)
  - CoUPM (Information Sciences 2019)
  - CoHUI-Miner (IEEE Access 2020)
- **Key idea**: The correlation measure must satisfy some properties that support the process of pruning candidates.

⇒ Propose a new version of FCHM algorithm which uses cosine measure to evaluate correlation between itemsets

Algorithms

The FHM algorithm

The FCHM algorithm

The FHM algorithm

The *TWU* upper bound

TWU of an itemset: the sum of the transaction utility for transactions containing the itemset

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

Item	a	b	c	d	e	f	g
Profit	5	2	1	2	3	1	1

Example:

TWU({a,e}) = TU(T_1) + TU(T_4) = 30\$ + 27\$ = 57\$

TWU({a,e}) = 57\$ $\geq$ u({a,e}) = 24\$ and the utility of any superset of {a,e}

The FHM algorithm

Utility-list structure

Create a vertical structure named *Utility-List* for each item

Trans.	Items
T_1	(a,1), (b,5), (c,1), (d,3), (e,1), (f,5)
T_2	(b,4), (c,3), (d,3), (e,1)
T_3	(a,1), (c,1), (d,1)
T_4	(a,2), (c,6), (e,2), (g,5)
T_5	(b,2), (c,2), (e,1), (g,2)

Item	a	b	c	d	e	f	g
Profit	5	2	1	2	3	1	1

Example: The utility-list of {d}:

Trans.	util	rutil
T_1	6	8
T_2	6	3
T_3	2	0

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

The FHM algorithm

Utility-list structure

Create a vertical structure named *Utility-List* for each item

Trans.	Items
T_1	(a,1), (b,5), (c,1), (d,3), (e,1), (f,5)
T_2	(b,4), (c,3), (d,3), (e,1)
T_3	(a,1), (c,1), (d,1)
T_4	(a,2), (c,6), (e,2), (g,5)
T_5	(b,2), (c,2), (e,1), (g,2)

Item	a	b	c	d	e	f	g
Profit	5	2	1	2	3	1	1

Example: The utility-list of {d}:

Trans.	util	rutil
T_1	6	8
T_2	6	3
T_3	2	0

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

The FHM algorithm

Utility-list structure

Create a vertical structure named *Utility-List* for each item

Trans.	Items
T_1	(a,1), (b,5), (c,1), (d,3), (e,1), (f,5)
T_2	(b,4), (c,3), (d,3), (e,1)
T_3	(a,1), (c,1), (d,1)
T_4	(a,2), (c,6), (e,2), (g,5)
T_5	(b,2), (c,2), (e,1), (g,2)

Item	a	b	c	d	e	f	g
Profit	5	2	1	2	3	1	1

Example: The utility-list of {d}:

Trans.	util	rutil
T_1	6	8
T_2	6	3
T_3	2	0

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

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

The FHM algorithm

Utility-list structure

Create a vertical structure named *Utility-List* for each item

Trans.	Items
T_1	(a,1), (b,5), (c,1), (d,3), (e,1), (f,5)
T_2	(b,4), (c,3), (d,3), (e,1)
T_3	(a,1), (c,1), (d,1)
T_4	(a,2), (c,6), (e,2), (g,5)
T_5	(b,2), (c,2), (e,1), (g,2)

Item	a	b	c	d	e	f	g
Profit	5	2	1	2	3	1	1

Example: The utility-list of {d}:

Trans.	util	rutil
T_1	6	8
T_2	6	3
T_3	2	0

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

The FHM algorithm

Utility-list structure

Create a vertical structure named *Utility-List* for each item

Trans.	Items
T_1	(a,1), (b,5), (c,1), (d,3), (e,1), (f,5)
T_2	(b,4), (c,3), (d,3), (e,1)
T_3	(a,1), (c,1), (d,1)
T_4	(a,2), (c,6), (e,2), (g,5)
T_5	(b,2), (c,2), (e,1), (g,2)

Item	a	b	c	d	e	f	g
Profit	5	2	1	2	3	1	1

Example: The utility-list of {d}:

Trans.	util	rutil
T_1	6	8
T_2	6	3
T_3	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 + 8 + 3 + 0 = 25 $

The FHM algorithm

Utility-list structure

Utility-list can be *joined* to calculate utility-list of large itemsets

Utility list of {a}

Trans.	Util	rutil
T_1	5	25
T_3	5	3
T_4	10	17

join

Utility list of {d}

Trans.	util	rutil
T_1	6	8
T_2	6	3
T_3	2	0

Utility list of {a,d}

Trans.	util	rutil
T_1	11	8
T_3	7	0

u({a}) = 20 $

u({d}) = 14 $

u({a,d}) = 18 $

The FHM algorithm

Utility-list structure

Utility-list can be *joined* to calculate utility-list of large itemsets

Utility list of {a}

Trans.	Util	rutil
T_1	5	25
T_3	5	3
T_4	10	17

+ **join**

Utility list of {d}

Trans.	util	rutil
T_1	6	8
T_2	6	3
T_3	2	0

→

Utility list of {a,d}

Trans.	util	rutil
T_1	11	8
T_3	7	0

u({a}) = 20 $

u({d}) = 14 $

u({a,d}) = 18 $

The FHM algorithm

Utility-list structure

Utility-list can be *joined* to calculate utility-list of large itemsets

Utility list of {a}

Trans.	Util	rutil
T_1	5	25
T_3	5	3
T_4	10	17

join

Utility list of {d}

Trans.	util	rutil
T_1	6	8
T_2	6	3
T_3	2	0

Utility list of {a,d}

Trans.	util	rutil
T_1	11	8
T_3	7	0

u({a}) = 20 $

u({d}) = 14 $

u({a,d}) = 18 $

The FHM algorithm

Utility-list structure

Construct utility-list of k-itemsets ($k \geq 3$)

Utility list of {a,b}

Trans.	Util	rutil
T_1	15	15

+ join

Utility list of {a,c}

Trans.	util	rutil
T_1	6	14
T_3	6	2
T_4	16	11

→

Utility list of {a,b,c}

Trans.	util	rutil
T_1	16	14

u({a,b}) = 15 $

u({a,c}) = 28 $

u({a,b,c}) = 16 $

Observation: Join operations are very costly in terms of execution time

→ We need to reduce the number of join operations

The FHM algorithm

Estimated Utility Co-occurrence pruning (EUCS)

- We pre-calculate the TWU of all pairs of items and store it in a structure named EUCS
- 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.

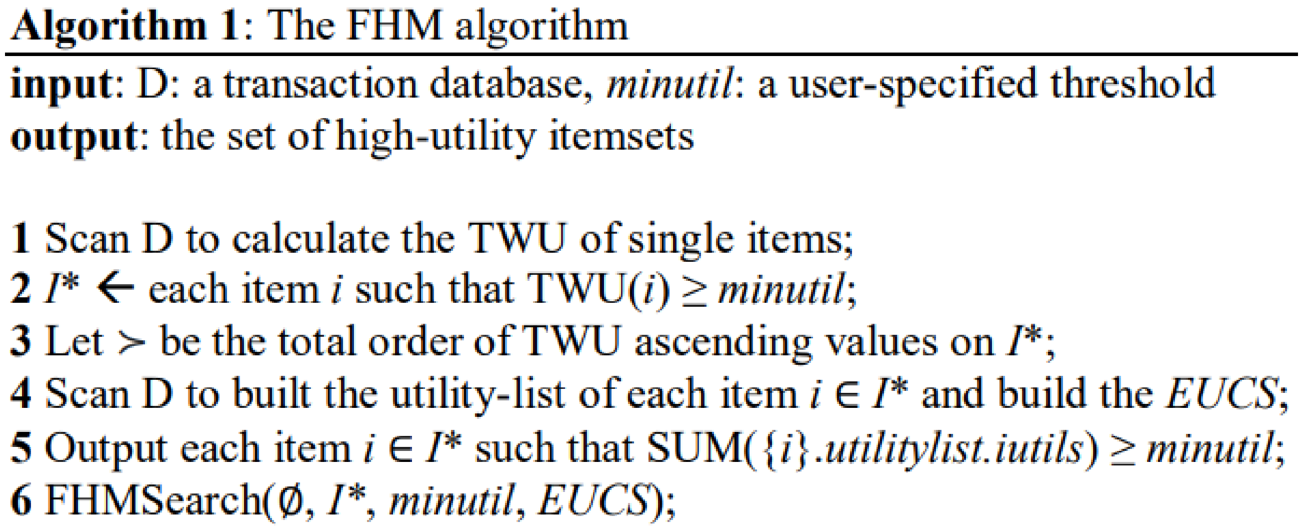
	a	b	c	d
b	25			
c	55	54		
d	33	45	53	
e	47	54	76	45

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

The FHM algorithm

General idea

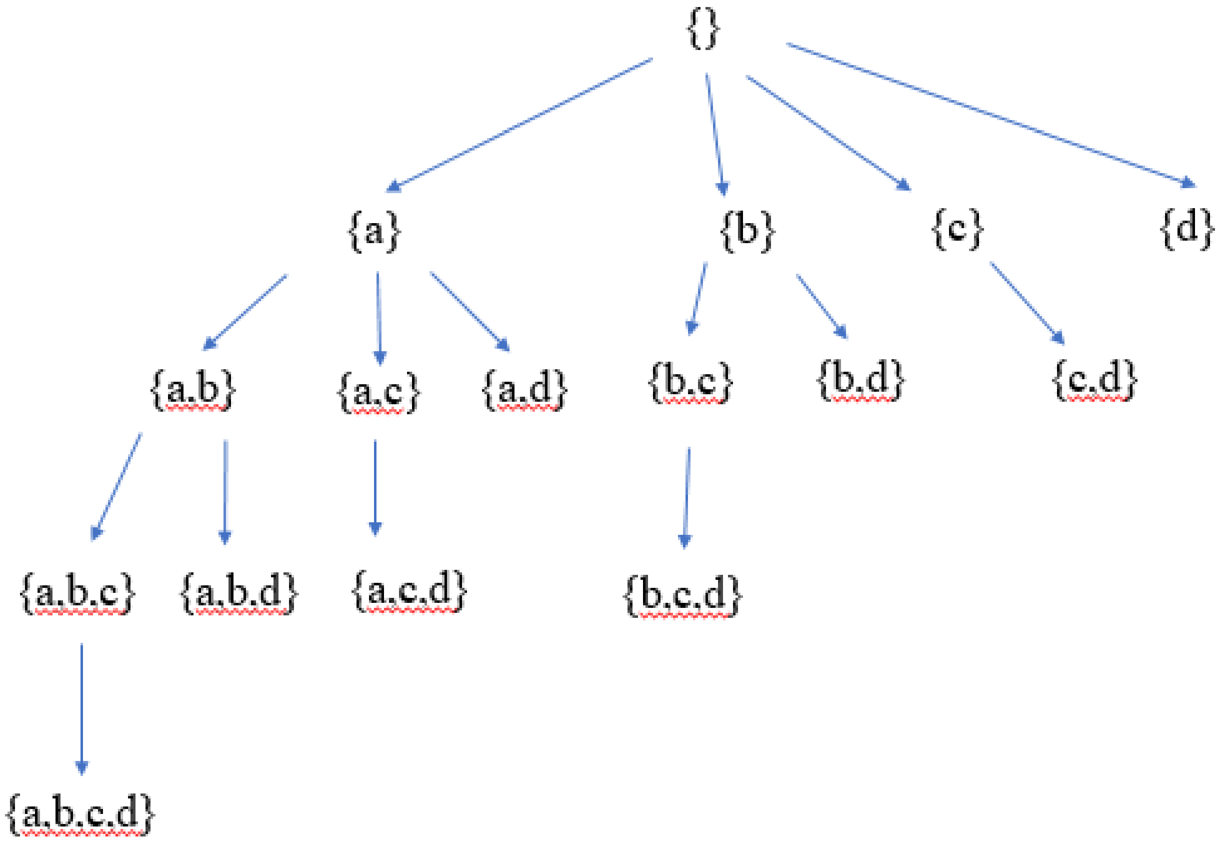
- An algorithm for mining high utility itemsets
- It performs a depth-first search



Algorithm 1: The FHM algorithm

input: D: a transaction database, *minutil*: a user-specified threshold
output: the set of high-utility itemsets

```
Scan D to calculate the TWU of single items;
$I^* \leftarrow$ each item $i$ such that $TWU(i) \geq minutil$;
Let $\succ$ be the total order of TWU ascending values on $I^*$;
Scan D to built the utility-list of each item $i \in I^*$ and build the $EUCS$;
Output each item $i \in I^*$ such that $SUM(\{i\}.utilitylist.iutils) \geq minutil$;
FHMSearch($\emptyset$, $I^*$, $minutil$, $EUCS$);
```



- It prune the search space using the utility measures

The FCHM algorithm

How to detect if items are correlated?

Several approaches:

- Using ***statistical tests*** to find productive itemsets (Webb et al., 2010)
- The ***affinity*** measure (Ahmed et al.2011)
- The ***bond*** measure (Bouasker et al.2015)
- The ***all-confidence*** measure (Omiecinski et al.2003)

The FCHM algorithm

The *bond* of an itemset

- The **conjunctive support** of an itemset X in a database is the number of transactions that **contains X**.
- The **disjunctive support** of an itemset X in a database is the number of transactions that **contains any item from X**.
- The *bond* of an item X is defined as:

$$bond(X) = \frac{conj\_\sup(X)}{disj\_\sup(X)}$$

Property (Anti-monotonicity of the bond measure). Let X and Y be two item-sets such that $X \subseteq Y$. It followes that bond(X) ≥ bond(Y)

The FCHM algorithm

The ***all-confidence*** of an itemset

The all-confidence of an item X is defined as:

$$all - confidence(X) = \frac{supp(X)}{max_{x \in X}\left(supp(x)\right)}$$

Where $max_{x \in X}\left(supp(x)\right)$ is the support of the item with the highest support in X

Property (Anti-monotonicity of the all-confidence measure). Let X and Y be two item-sets such that X ⊆ Y. It followes that all-confidence(X) ≥ all-confidence(Y)

The FCHM algorithm

Problem definition of FCHM

- Discovering all correlated high utility itemsets, that is itemsets:

  - Having a **utility** no less than a threshold **min_util**

  - Having a **bond** no less than a threshold **min_bond** or having an **all-confidence** no less than a threshold **min_all-confidence**

A transaction database

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

Item	a	b	c	d	e	f	g
Profit	5	2	1	2	3	1	1

For example, if **minutil = 30** and **minbond = 0.5**, correlated high utility itemsets are:

{b,d} util = 30 bond = 2/4 = 0.5

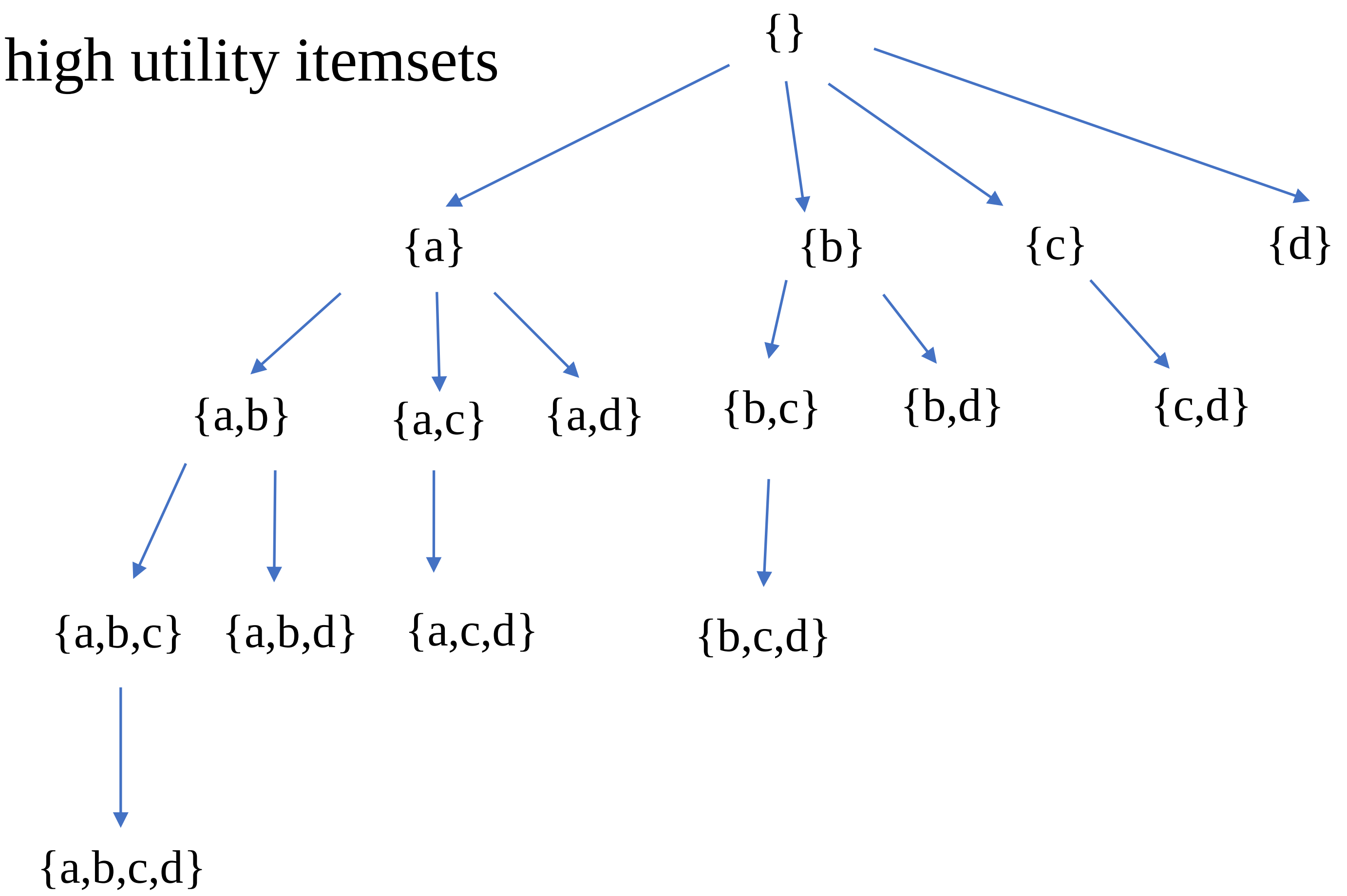
{b,e} util = 31 bond = 3/4 = 0.75

{b,c,e} util = 37 bond = 3/5 = 0.6

The FCHM algorithm

General idea

- An algorithm for mining correlated high utility itemsets
- It performs a depth-first search



- It prune the search space using the correlation measures (bond or all-confidence) and utility measures
- **Key challenge**: how to calculate the bond and all-confidence of an itemset

The FCHM algorithm

Calculation of Bond measure

Each itemset **X** is annotated with a **disjunctive bit vector** that stores the union of all items in X, denoted as *bv(X)*

e.g. the disj. bitvector of {a} is T_1, T_3, T_4 → *bv(a)* = 10110

the disj. bitvector of {b} is T_1, T_2, T_5 → *bv(b)* = 11001

the disj. bitvector of {a,b} is *bv(a)* OR *bv(b)* → 10110 OR 11001→ 11111

The bond of X can be calculated as $\frac{|ul(X)|}{|bv(X)|}$ where:

- $|ul(X)|$ is the number of elements in the utility list of X
- $|bv(X)|$ is the number of elements in the disjunctive bit vector

Calculation of All-confidence measure

- The support of X can be obtained by the size of its utility-list
- The support of single items can be obtained from their respective utility-list

The FCHM algorithm

Additional optimization for $\boldsymbol{FCHM_{all-confidence}}$

- Directly Outputting Single items (DOS)
- Pruning supersets of Non correlated itemsets (PSN)
- Pruning with Upper-Bound (PUB) version 1.

Additional optimization for $\boldsymbol{FCHM_{bond}}$

- Directly Outputting Single items (DOS)
- Pruning supersets of Non correlated itemsets (PSN)
- Pruning with Upper-Bound (PUB) version 2
- Abandoning Utility-list construction early (AUL)
- LA-Prune
- Pruning Utility-list by upper-bound (PUL)

Methodology

The Cosine measure

Proposes approach

The Cosine measure

- Cosine measure for two items:

$$cosine(A_1, A_2) = \frac{P(A_1 \cup A_2)}{\sqrt{P(A_1) \times P(A_2)}} = \frac{\sup(A_1 \cup A_2)}{\sqrt{\sup(A_1) \times \sup(A_2)}}$$

- Cosine measure for more than two items:

$$cosine(A_1, A_2, \dots, A_n) = \frac{P(A_1 \cup A_2 \cup \cdots \cup A_n)}{\sqrt{P(A_1) \times P(A_2) \times \cdots \times P(A_n)}} = \frac{\sup(A_1 \cup A_2 \cup \cdots \cup A_n)}{\sqrt{\sup(A_1) \times \sup(A_2) \times \cdots \times \sup(A_n)}}$$

➡

- Null-invariant measure
- Anti-monotonicity property

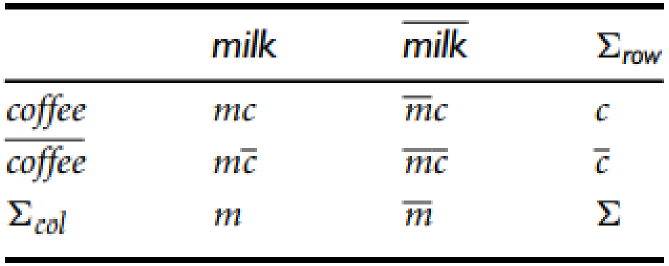
➡ Proposes $FCHM_{cosine}$ algorithm

The Cosine measure

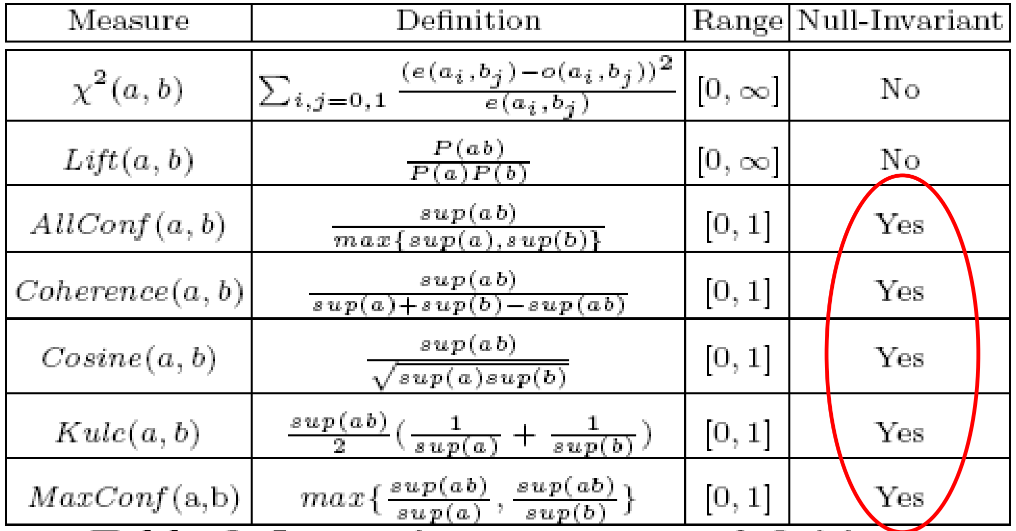
Null-invariant property

- A null-transaction is a transaction that does not contain any of the itemsets being examined
- Null-(transaction) invariance is crucial for correlation analysis

Table 6.8 2 × 2 Contingency Table for Two Items



	milk	$\overline{milk}$	Σ_{row}
coffee	mc	$\overline{m}c$	c
$\overline{coffee}$	$m\overline{c}$	$\overline{m}\overline{c}$	$\overline{c}$
Σ_{col}	m	$\overline{m}$	Σ

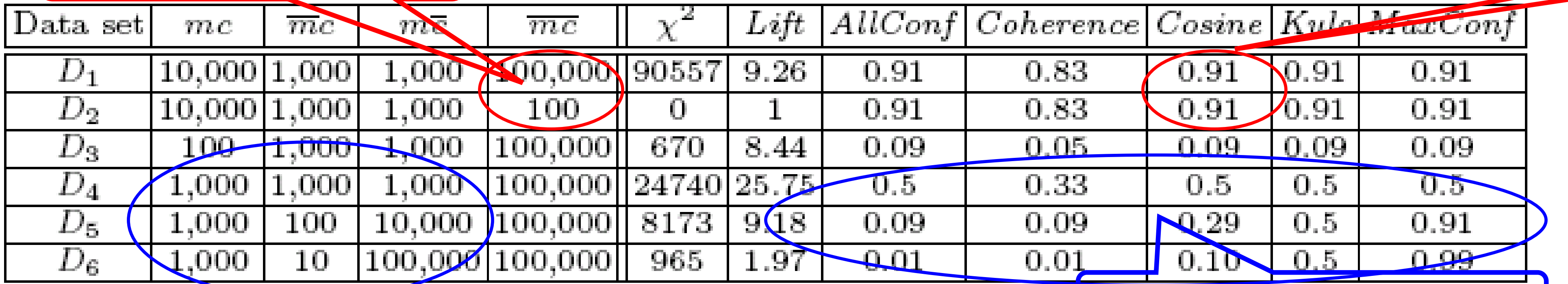


Measure	Definition	Range	Null-Invariant
$\chi^2(a,b)$	$\sum_{i,j=0,1} \frac{(e(a_i,b_j)-o(a_i,b_j))^2}{e(a_i,b_j)}$	$[0,\infty]$	No
$Lift(a,b)$	$\frac{P(ab)}{P(a)P(b)}$	$[0,\infty]$	No
$AllConf(a,b)$	$\frac{sup(ab)}{max\{sup(a),sup(b)\}}$	$[0,1]$	Yes
$Coherence(a,b)$	$\frac{sup(ab)}{sup(a)+sup(b)-sup(ab)}$	$[0,1]$	Yes
$Cosine(a,b)$	$\frac{sup(ab)}{\sqrt{sup(a)sup(b)}}$	$[0,1]$	Yes
$Kulc(a,b)$	$\frac{sup(ab)}{2}(\frac{1}{sup(a)}+\frac{1}{sup(b)})$	$[0,1]$	Yes
$MaxConf$(a,b)	$max\{\frac{sup(ab)}{sup(a)},\frac{sup(ab)}{sup(b)}\}$	$[0,1]$	Yes

Table 3. Interestingness measure definitions.

Null-invariant

Null-transactions w.r.t. m and c



Data set	mc	$\overline{m}c$	$m\overline{c}$	$\overline{mc}$	χ^2	$Lift$	$AllConf$	$Coherence$	$Cosine$	$Kulc$	$MaxConf$
D_1	10,000	1,000	1,000	100,000	90557	9.26	0.91	0.83	0.91	0.91	0.91
D_2	10,000	1,000	1,000	100	0	1	0.91	0.83	0.91	0.91	0.91
D_3	100	1,000	1,000	100,000	670	8.44	0.09	0.05	0.09	0.09	0.09
D_4	1,000	1,000	1,000	100,000	24740	25.75	0.5	0.33	0.5	0.5	0.5
D_5	1,000	100	10,000	100,000	8173	9.18	0.09	0.09	0.29	0.5	0.91
D_6	1,000	10	100,000	100,000	965	1.97	0.01	0.01	0.10	0.5	0.99

Table 2. Example data sets.

Subtle: They disagree

Proposed approach

Proof for anti-monotonicity property

$$cosine(A_1, A_2, \dots, A_n) = \frac{\sup(A_1 \cup A_2 \cup \cdots \cup A_n)}{\sqrt{\sup(A_1) \times \sup(A_2) \times \cdots \times \sup(A_n)}}$$

$$cosine(A_1, A_2, \dots, A_n, A_{n+1}) = \frac{\sup(A_1 \cup A_2 \cup \cdots \cup A_n \cup A_{n+1})}{\sqrt{\sup(A_1) \times \sup(A_2) \times \cdots \times \sup(A_n) \times \sup(A_{n+1})}}$$

Since $\sup(A_1 \cup A_2 \cup \cdots \cup A_n) \geq \sup(A_1 \cup A_2 \cup \cdots \cup A_n \cup A_{n+1})$ and

$\sqrt{\sup(A_1) \times \sup(A_2) \times \cdots \times \sup(A_n)} \leq \sqrt{\sup(A_1) \times \sup(A_2) \times \cdots \times \sup(A_n) \times \sup(A_{n+1})}$

➡ $cosine(A_1, A_2, \dots, A_n) \geq cosine(A_1, A_2, \dots, A_n, A_{n+1})$

➡ if the itemset does not satisfy minimum cosine α, it is no need to traverse its superset

Proposed approach

Calculation of cosine measure

- Product of support value of all 1-items is calculated during the construction of the utility list in FCHM algorithm:

  $product(Pxy) = product(Px) \times product(Py)$ if prefix P is null

  else $product(Pxy) = \frac{product(Px) \times product(Py)}{product(P)}$
- Support value of itemset X can be derived from utility list.

Additional optimization

- Directly Outputting Single items (DOS)
- Pruning Supersets of Non correlated itemsets (PSN)

Experiment and Analyze

Data

Effectiveness Analysis

Efficiency Analysis

Memory Analysis

Data

Dataset	No. of distinct items	No. of transactions	Average transaction length	Type
Foodmart	21,566	1,599	4.4	Sparse with short transactions
Mushroom	88,162	16,470	23	Dense
Retail	88,162	16,470	10.3	Sparse with many items

Effectiveness Analysis

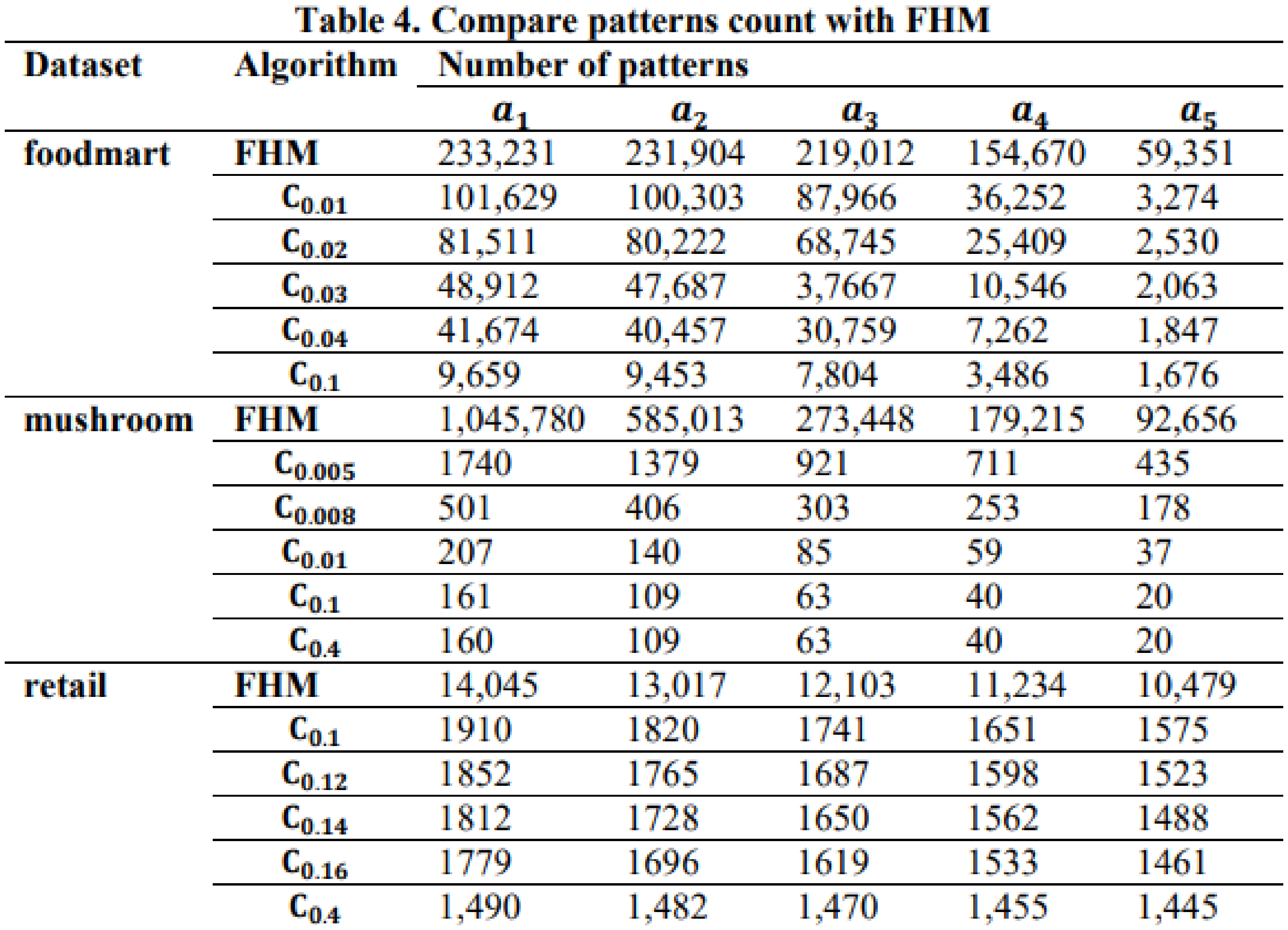


Table 4. Compare patterns count with FHM

Dataset	Algorithm	Number of patterns				
		a_1	a_2	a_3	a_4	a_5
foodmart	FHM	233,231	231,904	219,012	154,670	59,351
	$C_{0.01}$	101,629	100,303	87,966	36,252	3,274
	$C_{0.02}$	81,511	80,222	68,745	25,409	2,530
	$C_{0.03}$	48,912	47,687	3,7667	10,546	2,063
	$C_{0.04}$	41,674	40,457	30,759	7,262	1,847
	$C_{0.1}$	9,659	9,453	7,804	3,486	1,676
mushroom	FHM	1,045,780	585,013	273,448	179,215	92,656
	$C_{0.005}$	1740	1379	921	711	435
	$C_{0.008}$	501	406	303	253	178
	$C_{0.01}$	207	140	85	59	37
	$C_{0.1}$	161	109	63	40	20
	$C_{0.4}$	160	109	63	40	20
retail	FHM	14,045	13,017	12,103	11,234	10,479
	$C_{0.1}$	1910	1820	1741	1651	1575
	$C_{0.12}$	1852	1765	1687	1598	1523
	$C_{0.14}$	1812	1728	1650	1562	1488
	$C_{0.16}$	1779	1696	1619	1533	1461
	$C_{0.4}$	1,490	1,482	1,470	1,455	1,445

Reduce a large number of weakly correlated patterns compared to FHM algorithm

Effectiveness Analysis

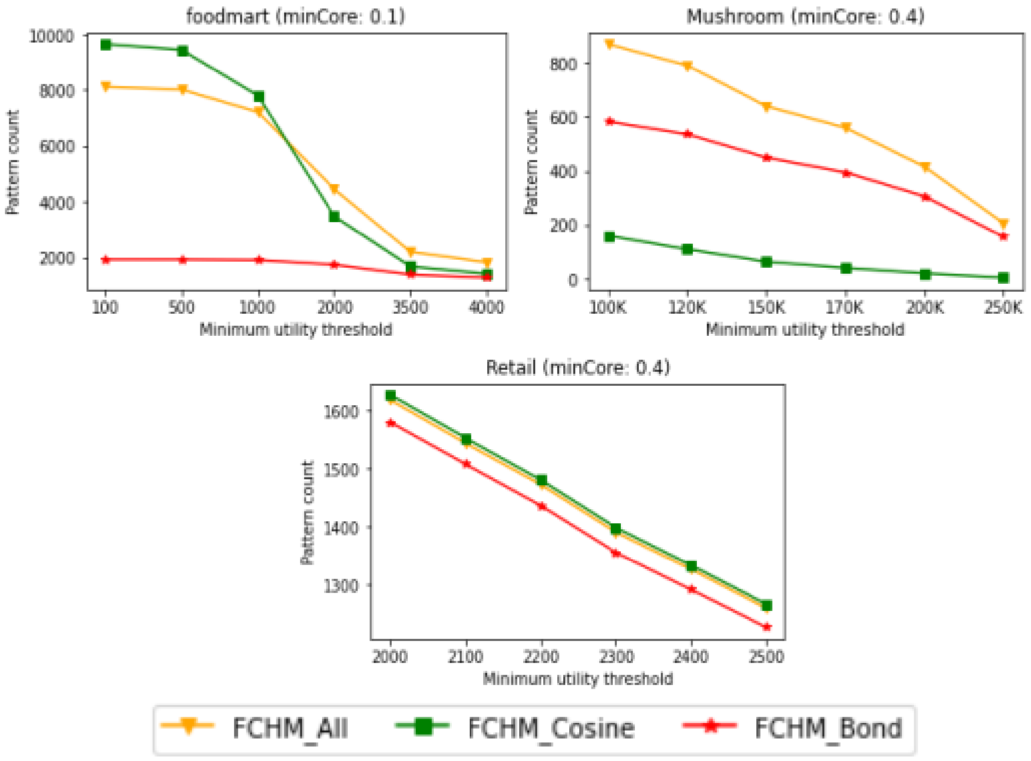


Fig. 1. Compare pattern count with other versions (varying *minUtil*, fixing *minCore*)

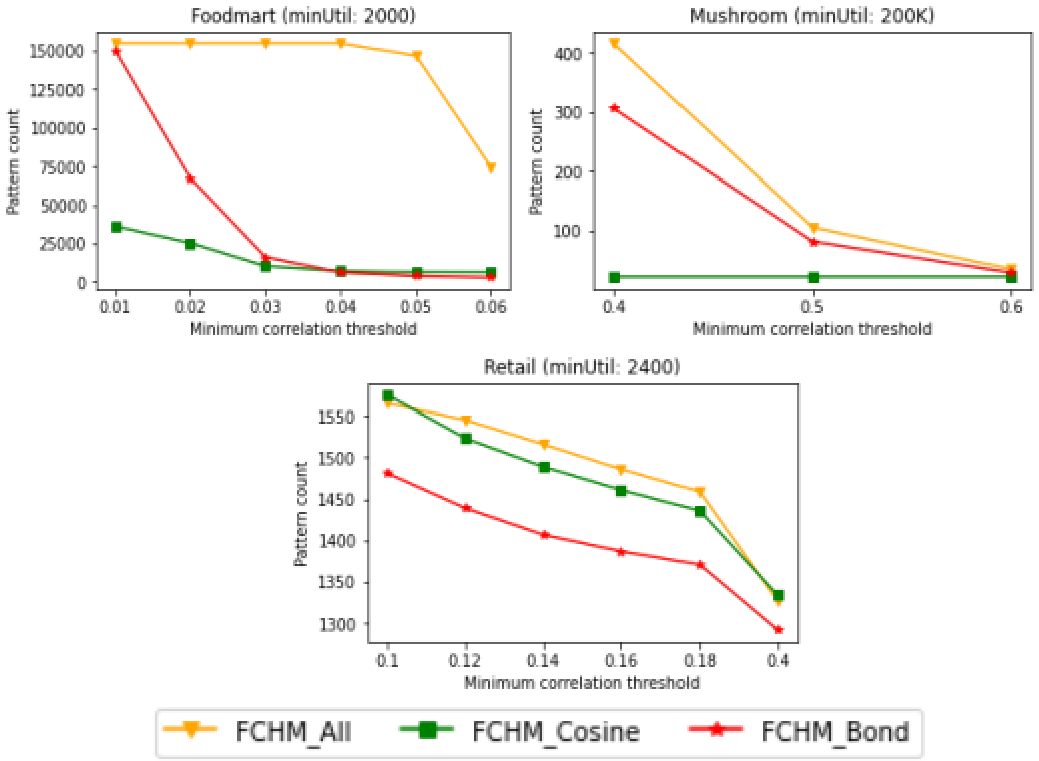


Fig. 2. Compare pattern count with other versions (varying *minCore*, fixing *minUtil*)

The constraint set by the proposed algorithm can be considered tighter than previous versions in some cases

Efficiency Analysis

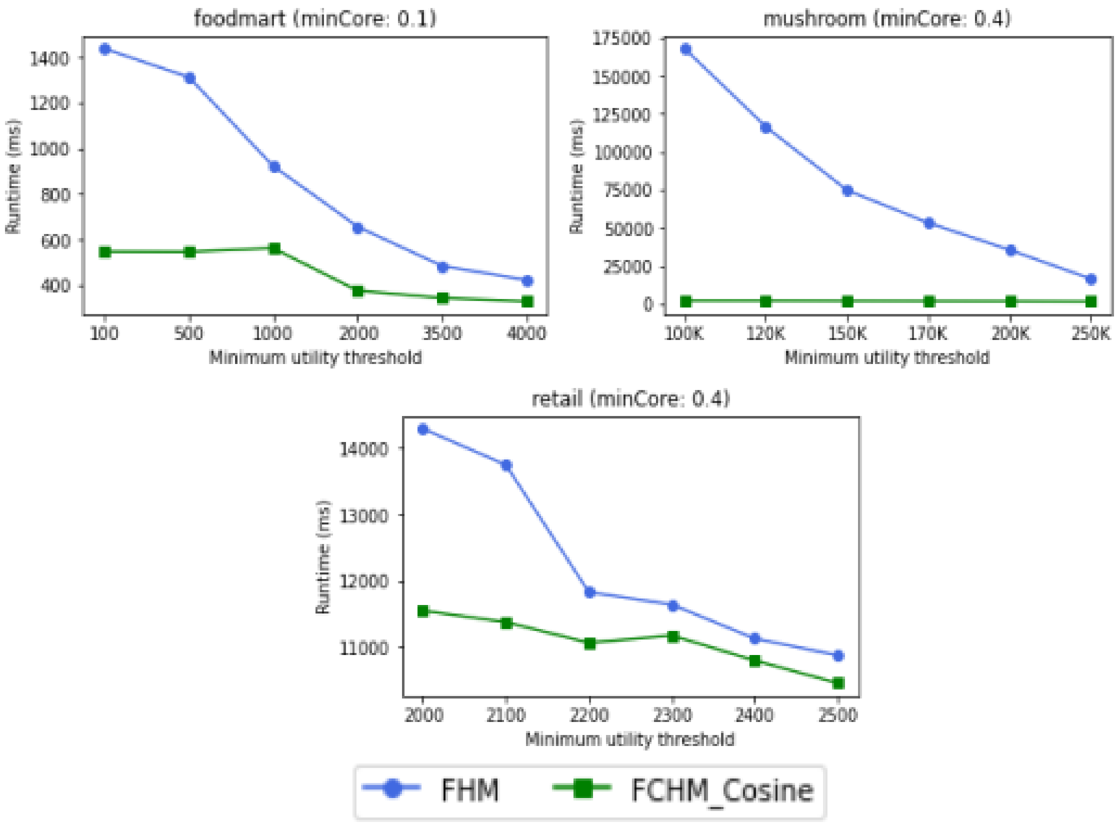


Fig. 3. Compare runtime with FHM (varying *minUtil*, fixing *minCore*)

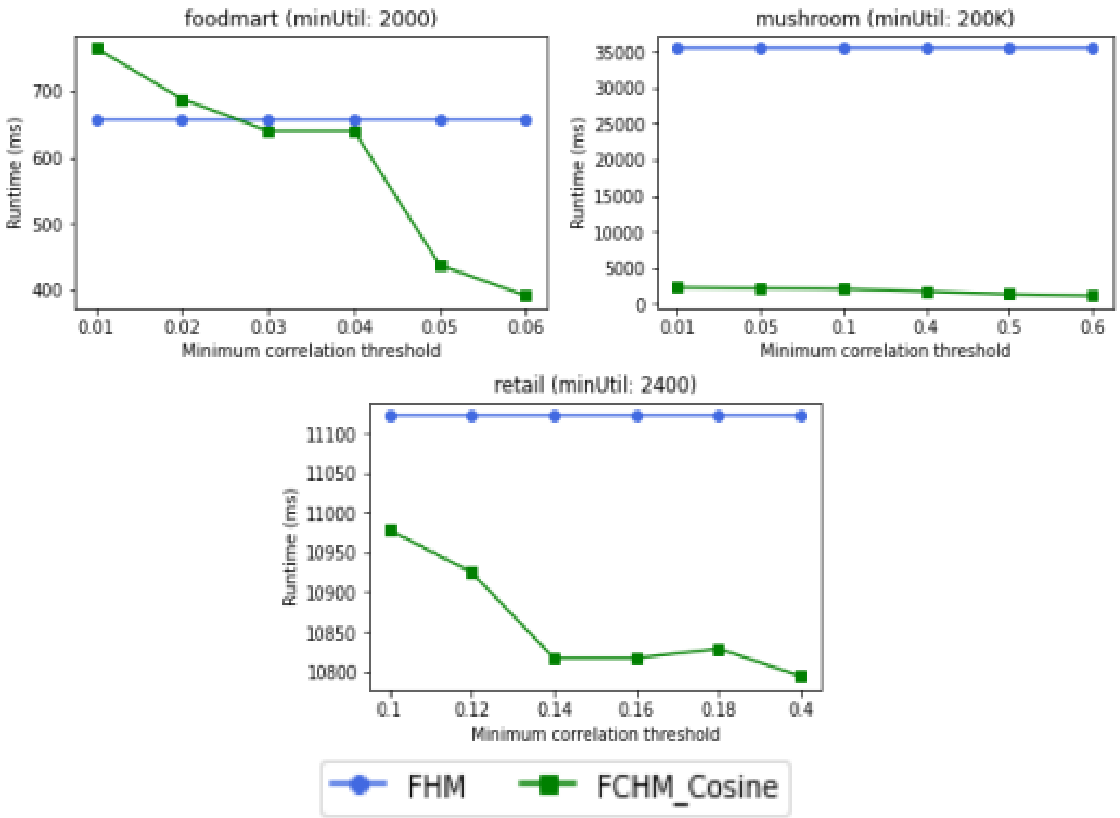


Fig. 4. Compare runtime with FHM (varying *minCore*, fixing *minUtil*)

➡ The runtime of $FCHM_{cosine}$ is much improved compared to FHM

Efficiency Analysis

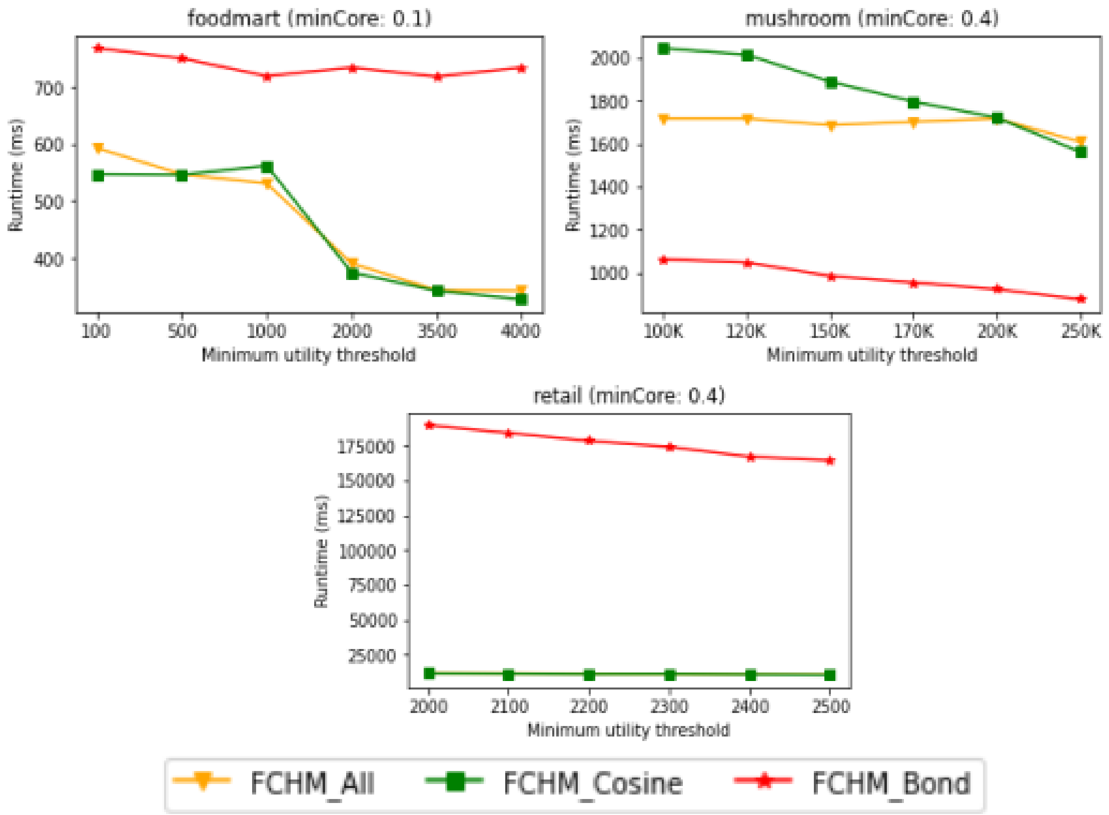


Fig. 5. Compare runtime with other versions (varying *minUtil*, fixing *minCore*)

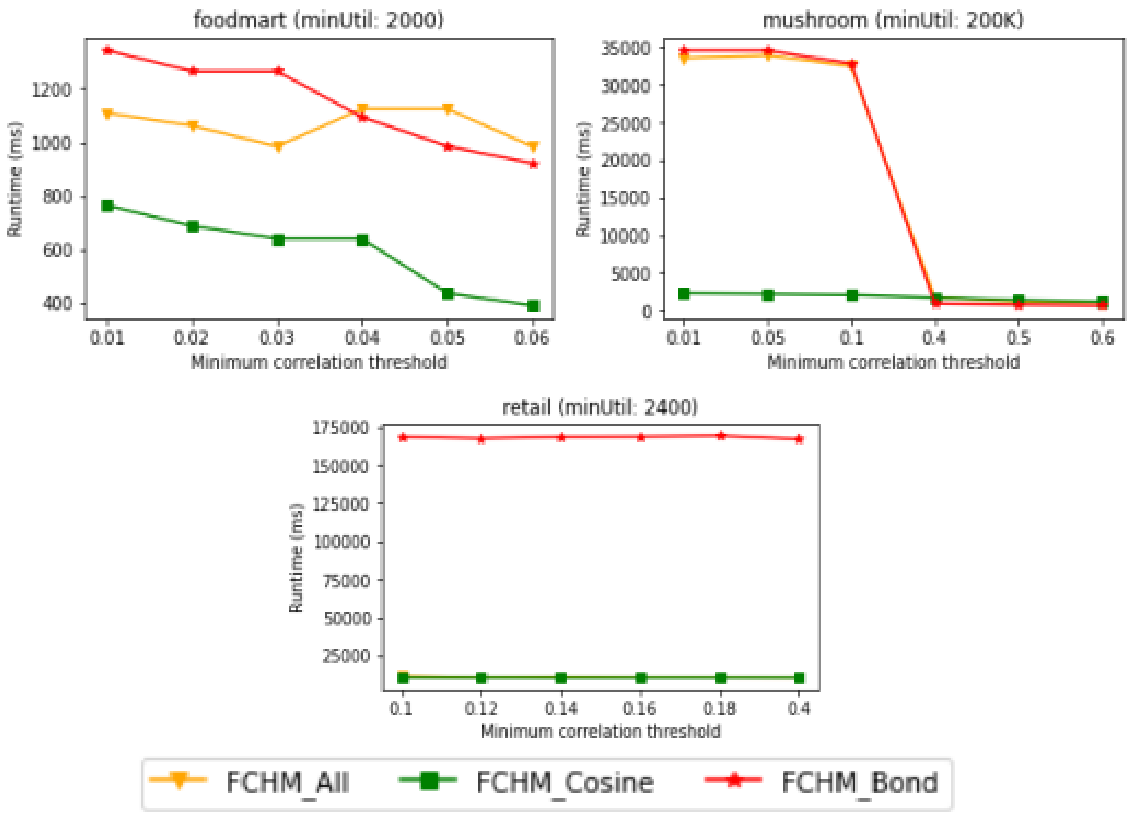


Fig. 6. Compare runtime with other versions (varying *minCore*, fixing *minUtil*)

- The runtime of $FCHM_{cosine}$ is quiet similar to $FCHM_{all-confidence}$
- The runtime of $FCHM_{cosine}$ is better than $FCHM_{bond}$ except for mushroom dataset

Memory Analysis

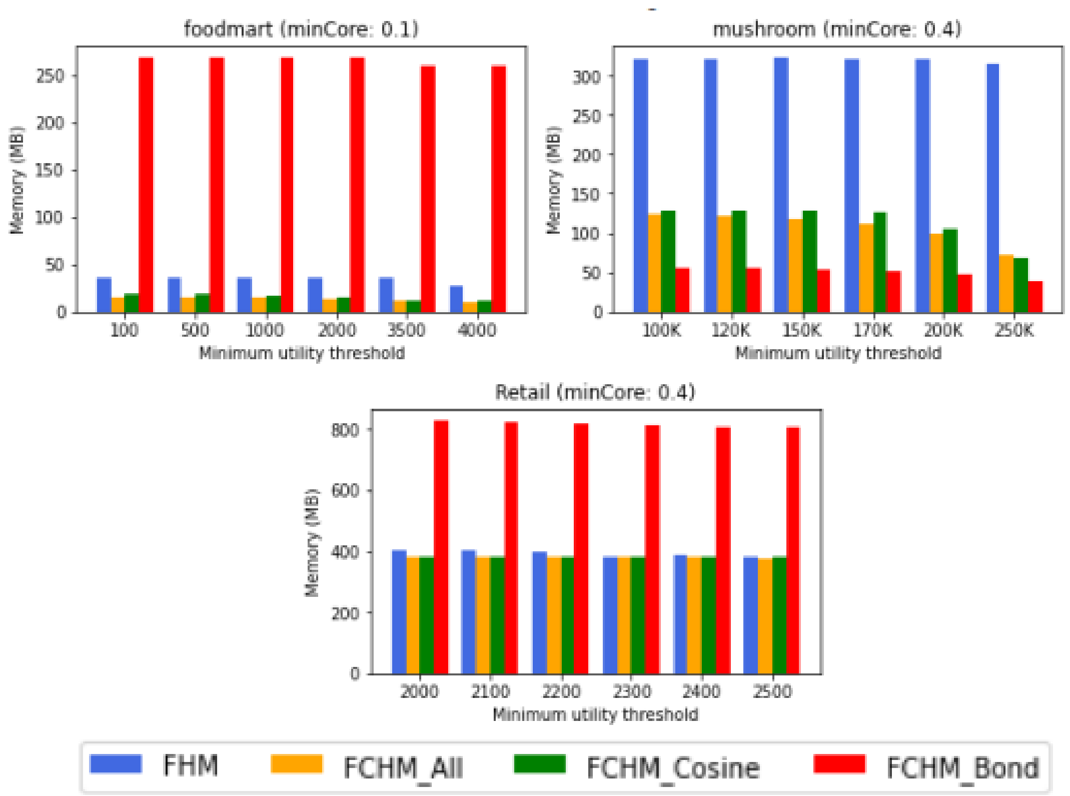


Fig. 7. Compare memory with FHM and other versions (varying *minUtil*, fixing *minCore*)

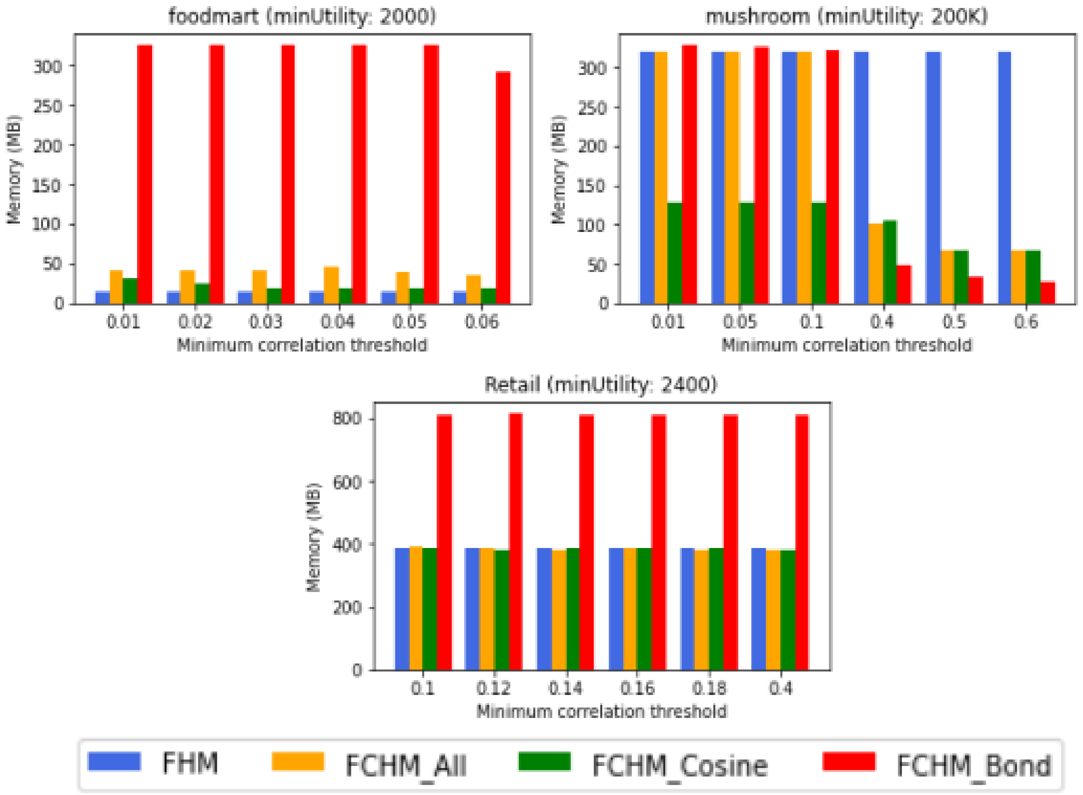


Fig. 8. Compare memory with FHM and other versions (varying *minCore*, fixing *minUtil*)

The $FCHM_{cosine}$ is always in the top two algorithms with the lowest memory consumption

Conclusion and perspectives

Conclusion

- Proposes the $FCHM_{cosine}$ algorithm, which is a new version of the FCHM algorithm
- $FCHM_{cosine}$ significantly reduces weakly correlated patterns compared with the traditional HUIM algorithm
- $FCHM_{cosine}$ has a stable runtime with memory consumption and in some cases better than the previous two versions of the FCHM algorithm

Future works

- Developing new pruning strategies which suitable for cosine measure
- Research more on other null-invariant measures

Thanks for your attention !

Q & A