FHM: Faster High-Utility Itemset Mining using Estimated Utility Co-occurrence Pruning

Philippe Fournier-Viger[1]

Cheng Wei Wu[2]

Souleymane Zida[1]

Vincent S. Tseng[2]

presented by Ted Gueniche[1]

[1]**University of Moncton, Canada**

[2]**National Cheng Kung University**

Introduction

- **Frequent Itemset Mining**
  - consists of discovering groups of items frequently occurring in a set of transactions.
- **Example**:

A transaction database

Transaction	item
T_1	{1, 2, 3, 4, 5}
T_2	{1, 2, 5}
T_3	{3, 4, 5}
T_4	{1, 2, 4, 5}

FIM with minsup = 50 % →

Frequent itemsets

Itemset	Support
{5}	100 %
{4, 5}	75 %
{2, 4, 5}	50 %
...	...

Limitations : assume an item can only appear once in a transaction !
assume all items have the same importance/weight (e.g. profit)
Thus, may ignore rare itemset having high profit ! (e.g. caviar, wine)

High Utility Itemset Mining

- **A generalization of FIM such that:**
  - items can appear more than once in each transaction
  - each item has a weight/profit
- **Several applications**:
  - click-stream analysis,
  - cross-marketing in retail stores,
  - bio-medical applications…

High Utility Itemset Mining

Input: transaction database with quantities

TID \ ITEM	A	B	C	D	E
T_1	0	0	18	0	1
T_2	0	6	0	1	1
T_3	2	0	1	0	1
T_4	1	0	0	1	1
T_5	0	0	4	0	2
T_6	1	1	0	0	0
T_7	0	10	0	1	1
T_8	3	0	25	3	1
T_9	1	1	0	0	0
T_{10}	0	6	2	0	2

unit profit table

ITEM	PROFIT ($)(per unit)
A	3
B	10
C	1
D	6
E	5

a threshold *minutil*

Output: *high-utility itemsets*, the itemsets having a utility no less than *minutil*

How to calculate an itemset's utility?

ITEM / TID	A	B	C	D	E
T_1	0	0	18	0	1
T_2	0	6	0	1	1
T_3	2	0	1	0	1
T_4	1	0	0	1	1
T_5	0	0	4	0	2
T_6	1	1	0	0	0
T_7	0	10	0	1	1
T_8	3	0	25	3	1
T_9	1	1	0	0	0
T_{10}	0	6	2	0	2

ITEM	PROFIT ($)(per unit)
A	3
B	10
C	1
D	6
E	5

For each transaction, where the itemset appears, we make the sum of the quantity of each item in the itemset multiplied by its unit profit.

$$u(\{B,D\}) = (6\times10 + 1\times6) + (10\times10 + 1\times6) = 172$$

A difficult task!

- In **frequent itemset mining**, the anti-monotonicity of the support is used to prune the search space.
- In **high-utility-itemset mining**, **utility** is **not anti-monotonic**.
- **Example**:

  u({D}) = 30
  u({B}) = 240

  u({B, D}) = 172
- Therefore, algorithms for FIM cannot be directly applied to HUIM.

How to solve this problem?

- Mine itemsets using two phases:
  - **Two-Phase (PAKDD, 2005), IHUP (TKDE 2010), UP-Growth (KDD, 2011)**
  - The TWU measure is introduced.
    - an upper bound on the utility of itemsets.
    - anti-monotonic
  - **Phase 1**: Discover candidate itemsets, that is having a **TWU $\geq$ minutil,**
  - **Phase 2:** For each candidate, calculate its exact utility of by scanning the database.

Recently…

HUI-Miner (CIKM, 2012) – a single phase algorithm

- Create a vertical structure named **Utility-List** for **each item.**
- To find larger itemsets, perform a depth-first search by appending items one at a time.
- The exact utility of an itemset is obtained by joining utility-lists of smaller itemsets (no need to scan database).
- **Pruning** using remaining utility in utility lists
- **HUI-Miner outperforms all previous algorithms.**

Utility list of {a}

TID	util	rutil
T1	5	3
T2	10	17
T3	5	25

utility = 20

\+ join

Utility list of {e}

TID	util	rutil
T2	6	5
T3	3	5
T4	3	0

utility = 12

→

Utility list of {a, e}

TID	util	rutil
T2	16	5
T3	8	5

utility = 24

Problems of HUI-Miner

- **Observation**: Calculating the utility of an itemset joining utility list is very costly.
- We should try to avoid performing joins if possible for low-utility itemsets.
- How?

Utility list of {a}

TID	util	rutil
T1	5	3
T2	10	17
T3	5	25

utility = 20

join

Utility list of {e}

TID	util	rutil
T2	6	5
T3	3	5
T4	3	0

utility = 12

Utility list of {a, e}

TID	util	rutil
T2	16	5
T3	8	5

utility = 24

The FHM algorithm

Main characteristics:

- Extends HUI-Miner.
- Depth-first search.
- Relies on utility-lists to calculate the exact utility of itemsets.
- **Estimated-Utility Co-occurrence pruning:**
  - we pre-calculate the TWU measures of 2-itemsets.
  - If an itemset contains a 2-itemset such that its TWU < minutil, then it is low utility as well as all its supersets, and the join is not performed.

How to calculate TWU? (1)

- The **transaction utility** of **a transaction** is the sum of the utility of items in that transaction
- **Example**:

TID	Transaction Utility	TID	Transaction Utility
T_1	23	T_6	13
T_2	71	T_7	111
T_3	12	T_8	57
T_4	14	T_9	13
T_5	14	T_{10}	72

How to calculate TWU (2)

The **transaction weighted utility (TWU) of an itemset** is the sum of the transaction utilities of transactions containing it.

- TWU({A}) = $tu(T3) + tu(T4) + tu(T6) + tu(T8) + tu(T9)$ = 12 + 14 + 13 + 57 + 13 = 109
- TWU({A, D}) =$tu(T4) + tu(T8)$ = 14 + 57 = 71.

Estimated Utility Co-Occurrence Structure (EUCS)

- Stores the TWU of all 2-itemsets.
- Built during the initial database scans.
- Represented as a triangular matrix or hashmap of hashmaps
- **Example**:

Item	a	b	c	d	e	f
b	30					
c	65	61				
d	38	50	58			
e	57	61	77	50		
f	30	30	30	30	30	
g	27	38	38	0	38	0

Note: this example is using another input database

Pseudocode (1)

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* ← each item i such that TWU(i) < minutil;
Let ≻ be the total order of TWU ascending values on I*;
Scan D to built the utility-list of each item i ∈ I* and build the EUCS
structure;
Search (∅, I*, minutil, EUCS);
```

Algorithm 2: The *Search* procedure

```
input : P: an itemset, ExtensionsOfP: a set of extensions of P, the minutil
        threshold, the EUCS structure
output: the set of high-utility itemsets

foreach itemset Px ∈ ExtensionsOfP do
    if SUM(Px.utilitylist.iutils) ≥ minutil then
        output Px;
    end
    if SUM(Px.utilitylist.iutils)+SUM(Px.utilitylist.rutils) ≥ minutil then
        ExtensionsOfPx ← ∅;
        foreach itemset Py ∈ ExtensionsOfP such that y ≻ x do
            if ∃(x, y, c) ∈ EUCS such that c ≥ minutil) then
                Pxy ← Px ∪ Py;
                Pxy.utilitylist ← Construct (P, Px, Py);
                ExtensionsOfPx ← ExtensionsOfPx ∪ Pxy;
            end
        end
        Search (Px, ExtensionsOfPx, minutil);
    end
end
```

Pseudocode (2)

Algorithm 3: The Construct procedure

input : P: an itemset, Px: the extension of P with an item x, Py: the extension of P with an item y

output: the utility-list of Pxy

```
UtilityListOfPxy ← ∅;
foreach tuple ex ∈ Px.utilitylist do
    if ∃ey ∈ Py.utilitylist and ex.tid = exy.tid then
        if P.utilitylist ≠ ∅ then
            Search element e ∈ P.utilitylist such that e.tid = ex.tid.;
            exy ← (ex.tid, ex.iutil + ey.iutil − e.iutil, ey.rutil);
        end
        else
            exy ← (ex.tid, ex.iutil + ey.iutil, ey.rutil);
        end
        UtilityListOfPxy ← UtilityListOfPxy ∪ {exy};
    end
end
return UtilityListPxy;
```

Experimental Evaluation

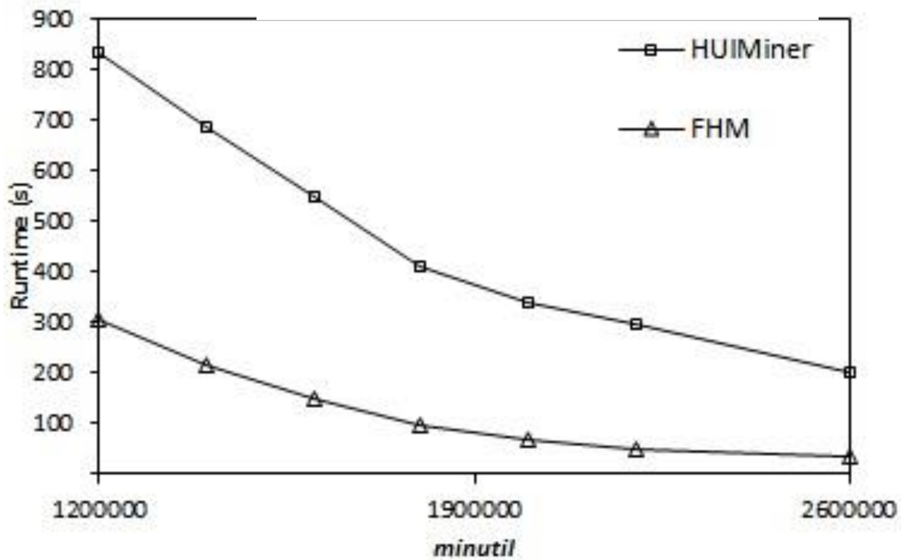
Datasets' characterictics

Dataset	transaction count	distinct item count	average transaction 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

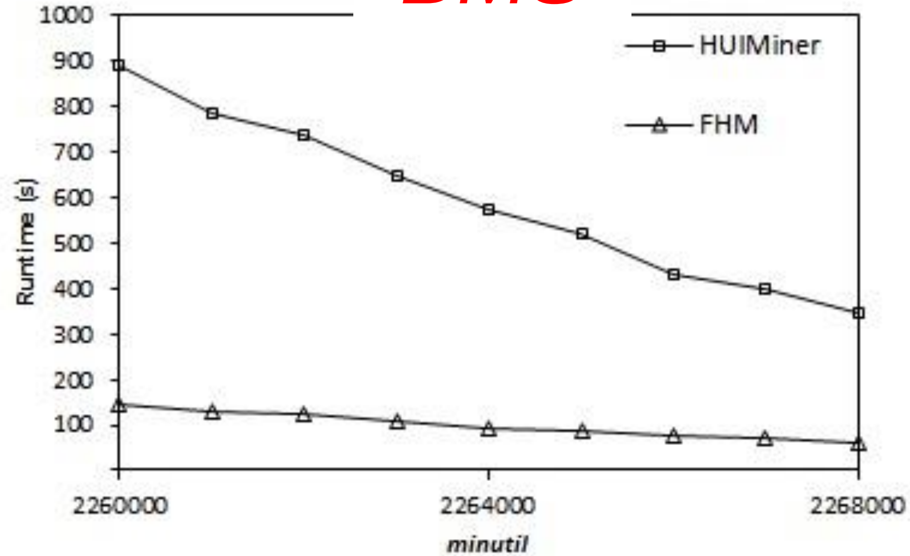
- Chainstore has real unit profit/quantity values
- Other datasets: unit profit between 1 and 1000 and quantities between 1 and 5 (normal distribution)
- FHM vs HUI-Miner
- Java, Windows 7, 5 GB of RAM

Execution times

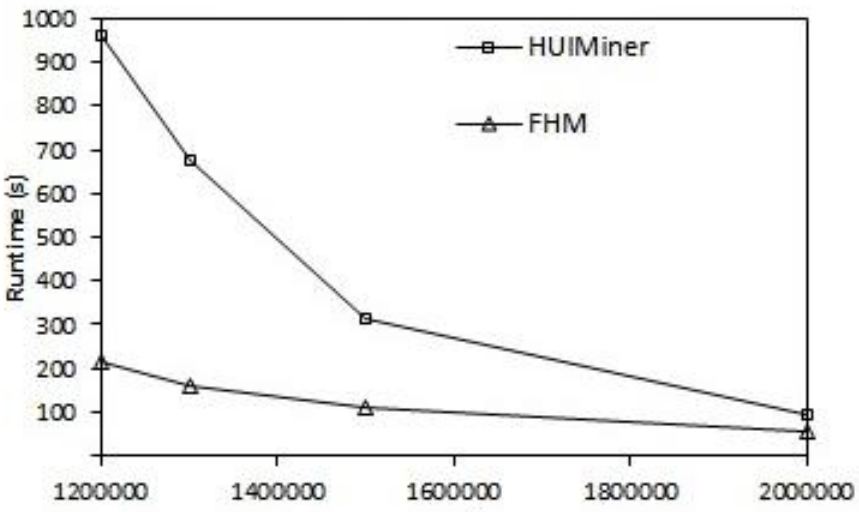
Chainstore



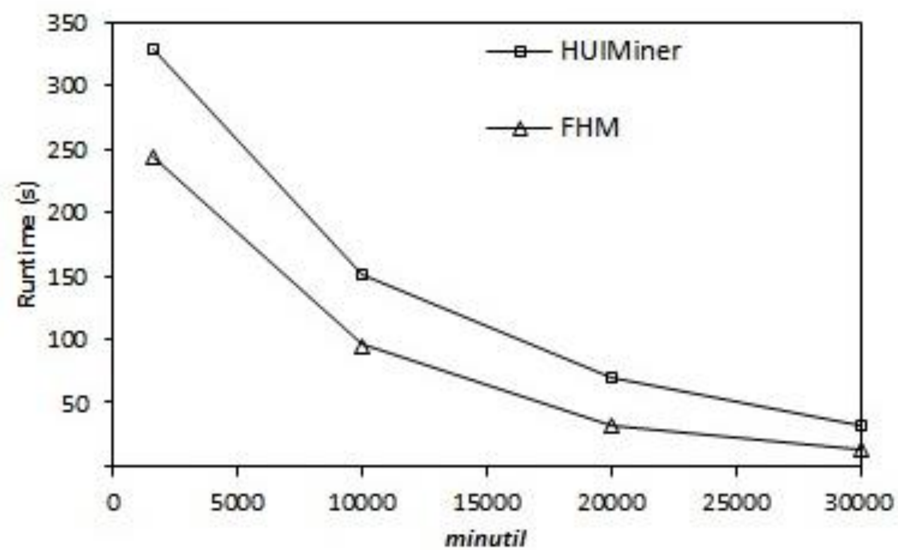
BMS



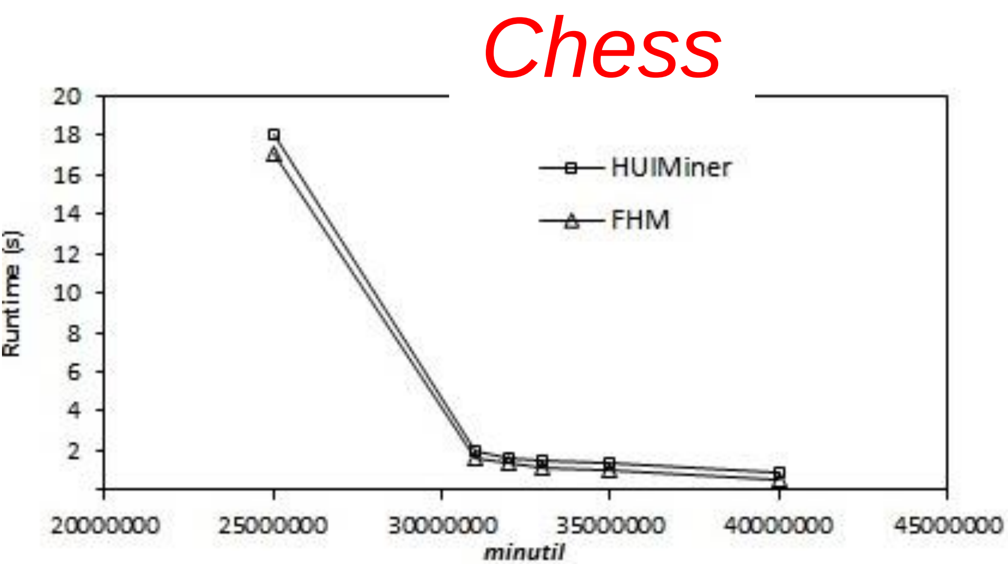
Kosarak

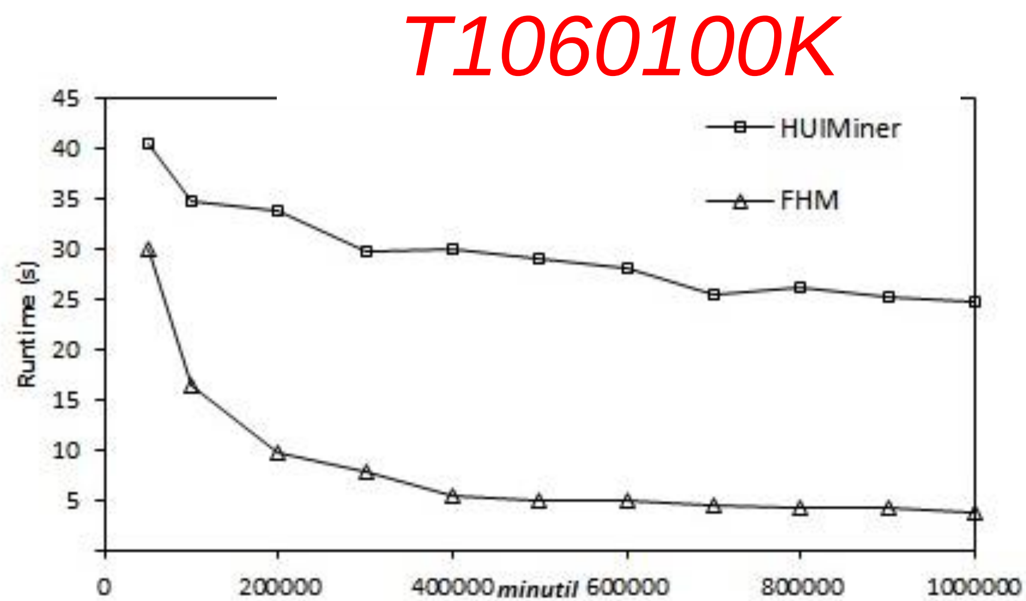


Retail



Execution times (cont'd)





Observations:

- **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

- A large amount of join operations are avoided by FHM.
- 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

Conclusion

- FHM: A novel algorithm for high-utility itemset mining
- Our proposal:
  - a novel data structure: EUCS (Estimated Utility Co-occurrence Structure)
  - a novel strategy to avoid some join operations: EUCP (Estimated Utility Coocurrence Pruning).
- Experimental results:
  - avoid up to 95 % of join operations
  - outperforms HUI-Miner by up to 6 times
- Source code and datasets available as part of the **SPMF data mining library (**GPL 3).

SPMF **Open source Java data mining software**, 66 algorithms
http://www.phillippe-fournier-viger.com/spmf/

Thank you. Questions?

Open source Java data mining software, 55 algorithms
http://www.phillippe-fournier-viger.com/spmf/

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