TKC: Mining Top-k Cross-Level High Utility Itemsets

Mourad Nouioua
School of Humanities and Social Sciences
Harbin Inst. of Technology (Shenzhen)
Shenzhen, China
University of Bordj Bou Arreridj, Algeria
mouradnouioua@gmail.com

Ying Wang
School of Computer Sciences
Harbin Inst. of Technology (Shenzhen)
Shenzhen, China
iwangying919@163.com

Philippe Fournier-Viger*
School of Humanities and Social Sciences
Harbin Inst. of Technology (Shenzhen)
Shenzhen, China
philfv8@yahoo.com

Jerry Chun-Wei Lin
Dept. of Computing, Mathematics and Physics
Western Norway Univ. of Applied Sciences (HVL)
Bergen, Norway
jerrylin@ieee.org

Jimmy Ming-Tai Wu
School of Computer Science and Engineering
Shandong University of Science and Technology
Qingdao, China
wmt@wmt35.idv.tw

Outline

- Introduction
  - Frequent itemset mining
  - High utility itemset mining
- Limitation
- A solution: TKC
- Experimental evaluation
- Conclusion

Frequent Itemset Mining

Input:

A transaction database

TID	Items
T_1	a, c
T_2	e
T_3	a, b, c, d, e
T_4	b, c, d, e
T_5	a, c, d
T_6	a, c, e
T_7	b, c, e

a $minsup$ threshold

Output:

Frequent itemsets (with support≥$minsup$)

if $minsup = 3$, the frequent itemsets are:

{a}:4	{b,e}:3
{b}:3	{c,d}:3
{d}:3	{c,e}:4
{a,c}:4	{b,c,e}:3
{b,c}:3	

Algorithms:

- Apriori（VLDB 1994）
- Apriori-TID（VLDB 1994）
- Eclat（TKDE 2000）
- FP-Growth（ACM SIGMOD 2000）

High Utility Itemset Mining

Input:

A transaction database

TID	Items
T_1	(a,1), (c,1)
T_2	(e,1)
T_3	(a,1), (b,5), (c,1), (d,3), (e,1)
T_4	(b,4), (c,3), (d,3), (e,1)
T_5	(a,1), (c,1), (d,1)
T_6	(a,2), (c,6), (e,2)
T_7	(b,2), (c,2), (e,1)

External utility values

Item	Unit profit
a	5$
b	2$
c	1$
d	2$
e	3$

a *minutil* threshold

Output:

High-utility itemsets (with utility≥*minutil*)

if *minutil* = 30$, the *HUIs* are:

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

High Utility Itemset Mining

- **Several algorithms:**
  - Two-Phase (PAKDD 2005)
  - IHUP (TKDE, 2010)
  - UP-Growth (KDD 2011)
  - HUI-Miner (CIKM 2012)
  - FHM (ISMIS 2014)
  - EFIM (KAIS 2017)
  - mHUIMiner (PAKDD 2017)
- **Key idea:**

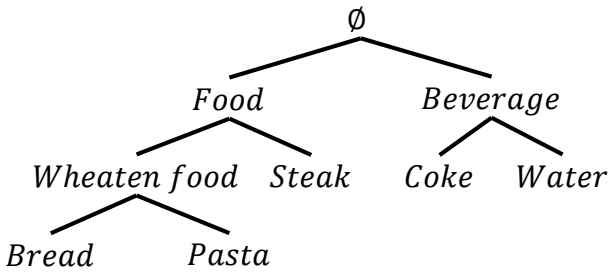
  Calculate an upper-bound on the utility of itemsets (**e.g.** the TWU) that is anti-monotonic to be able to prune the search space.

$$u(\{b,c,d\}) = u(\{b,c,d\}, T_3) + u(\{b,c,d\}, T_4)$$

$$TWU(\{b,c,d\}) = u(T_3) + u(T_4)$$

$$Twu(\{b,c,d\}) \geq u(\{b,c,d\})$$

Limitation



- **High utility itemset mining**
  - is useful for discovering profitable itemsets in a whole database
  - it ignores that items are organized according to **taxonomies**.
- **Consider taxonomy information**
  - Cross-level frequent itemset mining: Cumulate, Prutax, SET.
  - Multi-level frequent itemset mining: Han & Fu, Ong et al., Pramudiono.
  - Cross-level frequent itemset mining using multiple thresholds: Lui & Chung.
  - Multi-level frequent itemset mining using multiple thresholds: Rajkumar et al.
  - **Multi-level high utility itemset mining: MLHUI-Miner.**

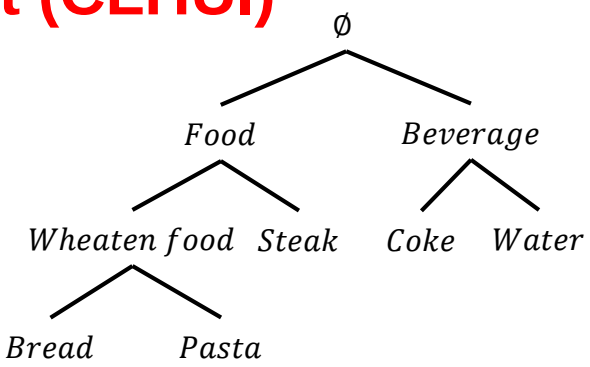
Limitation

- **MLHUI-Miner**
  - cannot find cross-level patterns.
  - mines each taxonomy levels independently.
  - does not use relationships between taxonomy levels.

We have previously proposed a new algorithm (CLH-Miner) to discover a new pattern type:

Cross-Level High Utility Itemset (CLHUI)

e.g. {Bread, Steak, Beverage} is a CLHUI, while not being a MLHUI or a HUI.



Limitation

- **CLH-Miner**
  - The difficulty of selecting the appropriate value of *minutil* that allows to find the desired number of patterns.
    - If *minutil* is too low, a huge number of patterns is found and the algorithm may have a very long execution time.
    - If *minutil* is too high, only few patterns are found and the algorithm may miss some important patterns.

Could we make a more efficient algorithm that does not require setting *minutil*?

A solution: TKC

- **TKC**
  - The problem is redefined as **Top-k cross-level high utility itemset mining**.
  - A novel algorithm named **TKC** (**T**op-**K** **C**ross-level high utility itemset miner) is proposed to:
    - Find **k cross-level HUIs with highest utilities without using minutil.**

      Input: k is the number of patterns to be generated.

      Output: The k cross-level HUIs with highest utilities.

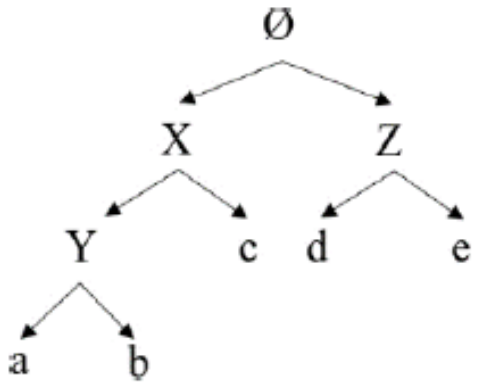
A solution: TKC

- TKC adopts:
  - A vertical database representation using **tax-utility-lists.**
  - Two kind of extensions to explore the search space.
  - Three optimizations (pruning strategies) to increase efficiency of the mining process.
- TKC does not use a fixed *minutil* to prune the search space.
- The key idea is once a new cross-level high utility itemset is found during the mining search, minutil is raised to the utility of the k-th pattern.
- The faster, minutil is raised, the faster the search space is reduced.

How to calculate the utility?

TID	Items
T_1	(a,1), (c,1)
T_2	(e,1)
T_3	(a,1), **(b,5), (c,1), (d,3)**, (e,1)
T_4	**(b,4), (c,3), (d,3)**, (e,1)
T_5	(a,1), (c,1), (d,1)
T_6	(a,2), (c,6), (e,2)
T_7	(b,2), (c,2), (e,1)

Item	Unit profit
a	5$
b	2$
c	1$
d	2$
e	3$



The utility of the itemset $\{b, c, d\}$ is calculated as follows:

$u(\{b, c, d\}) = (5 \times 2) + (1 \times 1) + (3 \times 2) + (4 \times 2) + (3 \times 1) + (3 \times 2) = 34$

The utility of the generalized itemset $\{Y,c\}$ is calculated as follows:

$u(\{Y, c\}) = (1 \times 5 + 1 \times 1) + (1 \times 5 + 5 \times 2 + 1 \times 1) + (4 \times 2 + 3 \times 1) + (1 \times 5 + 1 \times 1) + (2 \times 5 + 6 \times 1) + (2 \times 2 + 2 \times 1) = 61$

Problem definition

Input:

A transaction database

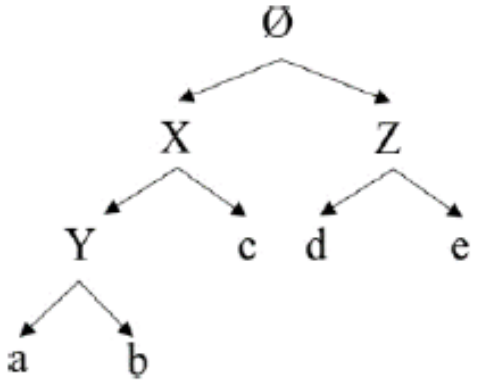
TID	Items
T_1	(a,1), (c,1)
T_2	(e,1)
T_3	(a,1), (b,5), (c,1), (d,3), (e,1)
T_4	(b,4), (c,3), (d,3), (e,1)
T_5	(a,1), (c,1), (d,1)
T_6	(a,2), (c,6), (e,2)
T_7	(b,2), (c,2), (e,1)

External utility values

Item	Unit profit
a	5$
b	2$
c	1$
d	2$
e	3$

The value of K

a taxonomy



Output:

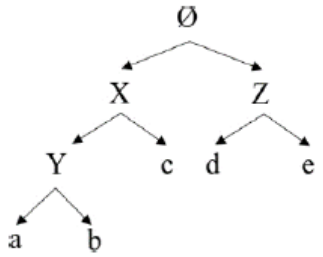
k Cross-level high-utility itemsets with highest utilities.

if $k=5\$$, the top-k *CLHUIs* are:

{Z,X}:84$	{Z,Y,c}:84$
{Z,Y}:71$	{e,X}:64$
	{e,Y,c}:64$

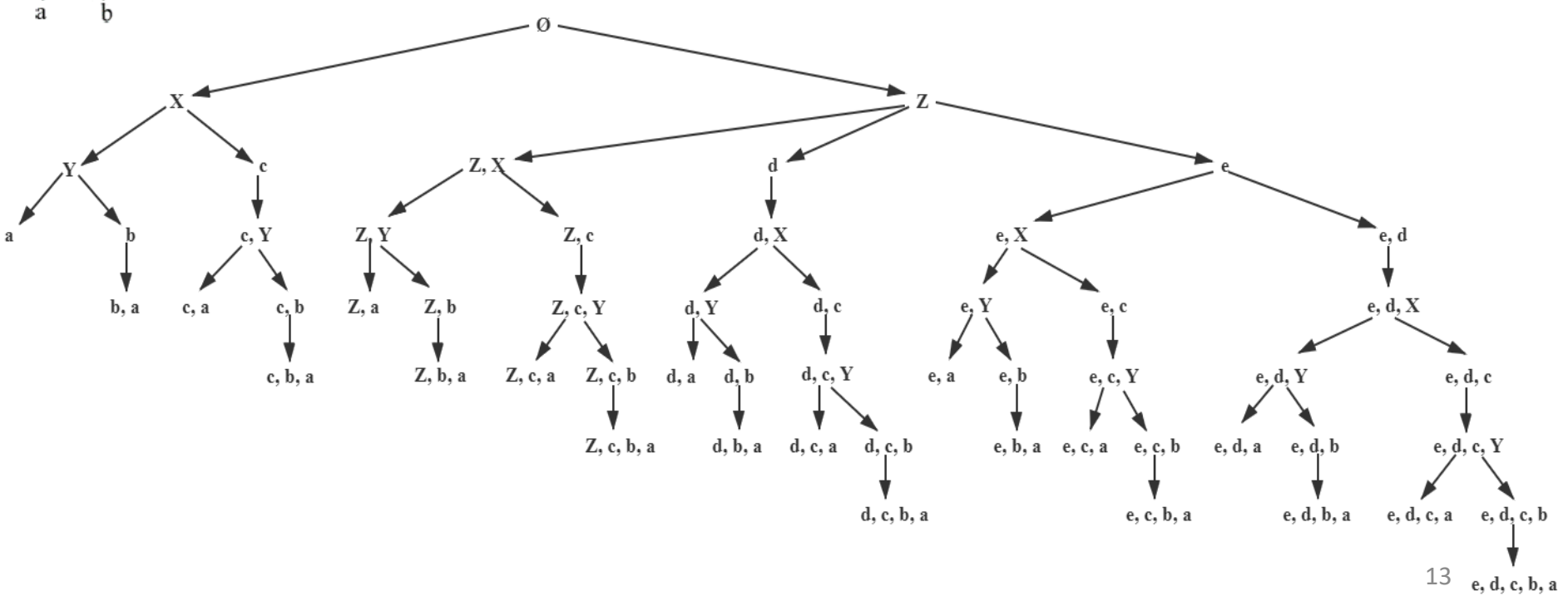
Search space exploration

Based on HUI-Miner, it extends the search space according to two kinds of **extensions**.

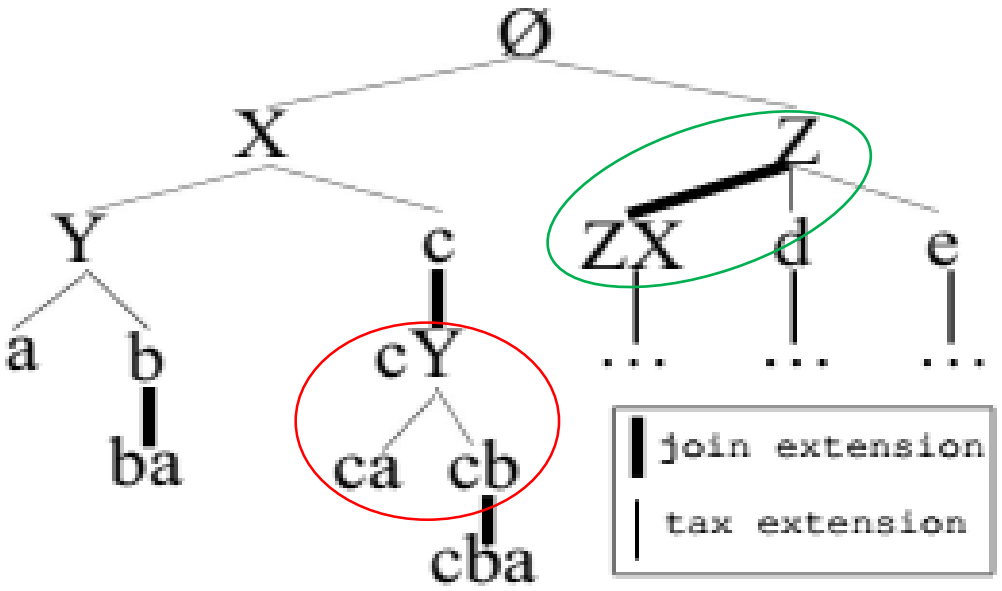


Two extensions:

- Tax-based extension
- Join-based extension



Search space exploration



e.g. {ZX} is a join-based extension of {Z}.
{ca} and {cb} are tax-based extensions of {cY}.

Construction of a tax-utility-list

Create a vertical structure named **tax-utility-list** for each itemset.

Tax-utility-list of {Z,Y}

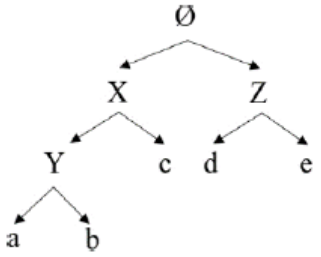
TID	*iutil*	*rutil*
T_3	24	1
T_4	17	3
T_5	7	1
T_6	16	6
T_7	7	2
Childs		
a	b	

utility-list

taxonomy information

utility of itemset in a transaction

remaining-utility



Construction of a tax-utility-list

- The **tax-utility-list** of a **single item** can be constructed by **scanning the database**.
- For other itemsets, it can be obtained by **joining their child itemset's tax-utility-lists**.

Tax-utility-list of {b}

TID	*iutil*	*rutil*
T_3	10	10
T_4	8	12
T_7	4	5
Childs		

Tax-utility-list of {c}

TID	*iutil*	*rutil*
T_1	1	0
T_3	1	9
T_4	3	9
T_5	1	2
T_6	6	6
T_7	2	3
Childs		

Tax-utility-list of {b,c}

TID	*iutil*	*rutil*
T_3	11	9
T_4	11	9
T_7	6	3
Childs		

How to reduce the search space?

To reduce the search space efficiently:

- **The GWU measure:**

$$GWU(X) = \sum_{T_c \in g(x)} TU(T_c), where\ TU(T_c) = \sum_{x \in T_c} u(x, T_c)$$

if $GWU(X) < minutil$, prune X and all its extensions.

- **The REU measure**:

$$REU(X) = \sum_{e \in tuList(X)} (e.iutil + e.rutil)$$

if $REU(X) < minutil$, prune all its tax-based extensions.

How to reduce the search space?

To raise *minutil* quickly:

- **Optimization 1:** Instead of initially setting *minutil=0*, *minutil* is set to the k-th largest utility value among initial items.

Item	{c}	{d}	{e}	{b}	{a}	{Z}	{Y}	{X}
Utility	14	14	18	22	25	32	47	61

e.g. If k=4, the 4-th largest utility is (u{a}=25). Thus, *minutil* is raised to 25.

if k=6, the 6-th largest utility is (u{e}=18). Thus, *minutil* is raised to 18.

Pseudocode

Algorithm 1: The TKC algorithm

input : k: the number of patterns to be found, τ: a taxonomy, D: a transaction database

output: the top-k cross-level HUIs

1 Read τ and D to compute the GWU and utility of each (generalized) item $i \in AI$;

2 Set $minutil$ to the k-th largest utility value among items in AI or 0 is $|AI| < k$;

3 Initialize a priority queue Q with the top-k cross-level HUIs from AI;

4 $GI^* \leftarrow \{g | g \in GI \wedge GWU(g) \geq minutil\}$;

5 $I^* \leftarrow \{i | i \in I \wedge GWU(i) \geq minutil \wedge \forall g \in GI$ such that $Desc(g, \tau) \ni i, GWU(g) \geq minutil\}$;

6 Compute $\prec$, the total order on items from $GI^* \cup I^*$;

7 Read τ and the database D to create the $tuList$ of each generalized item $g \in GI^*$ and item $i \in I^*$;

8 $TULs \leftarrow \{g | g \in GI^* \wedge level(g) = 1\}$;

9 `Find` $(TULs, k, Q, minutil)$;

Pseudocode

Algorithm 2: The *Find* procedure

input : $TULs$: a set of extensions of an itemset P, k: the number of patterns to find, Q: the top-k patterns until now, $minutil$: the internal threshold

output: Q is updated with top-k cross-level HUIs that are transitive extensions of P

```
1  foreach itemset X ∈ TULs such that GWU(X) ≥ minutil do
2      if SUM(X.tuList.iutils) ≥ minutil then
3          Insert X into Q;
4          Raises minutil to the k-th largest utility value in Q;
5          Remove from Q all patterns with utility less than minutil;
6      end
7      Extensions ← ∅;
8      foreach itemset Y ∈ TULs such that Y ≻ X do
9          Z.tuList ← ConstructJoinExtension(X, Y);
10         if GWU(Z) ≥ minutil then
               Extensions ← Extensions ∪ {Z};
11     end
12     if SUM(X.tuList.iutils)+SUM(X.tuList.rutils) ≥ minutil
         then
13         foreach itemset T ∈ X.childs do
14             Z.tuList ← ConstructTaxExtension(X, T);
15             if GWU(Z) ≥ minutil then
                   Extensions ← Extensions ∪ {Z};
16         end
17     end
18     Find(Extensions,k, Q, minutil);
19 end
```

Experimental evaluation

- **Datasets' characterictics**

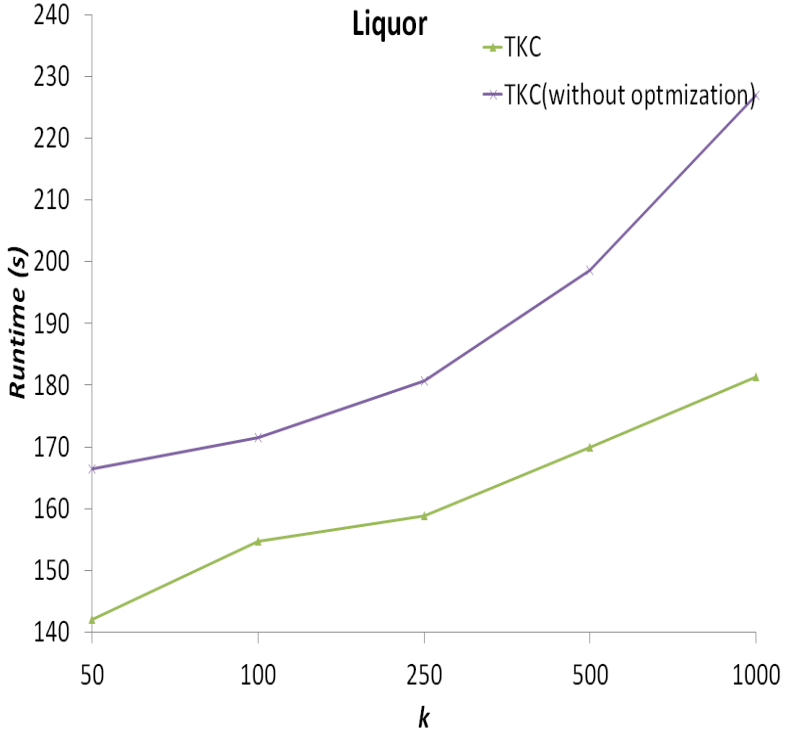
Dataset	$\|D\|$	$\|I\|$	$\|GI\|$	T_{min}	T_{max}	T_{avg}	$Level$
Liquor	9,284	2,626	77	1	5	2.7	7
Fruithut	181,970	1,265	43	23	36	3.58	4

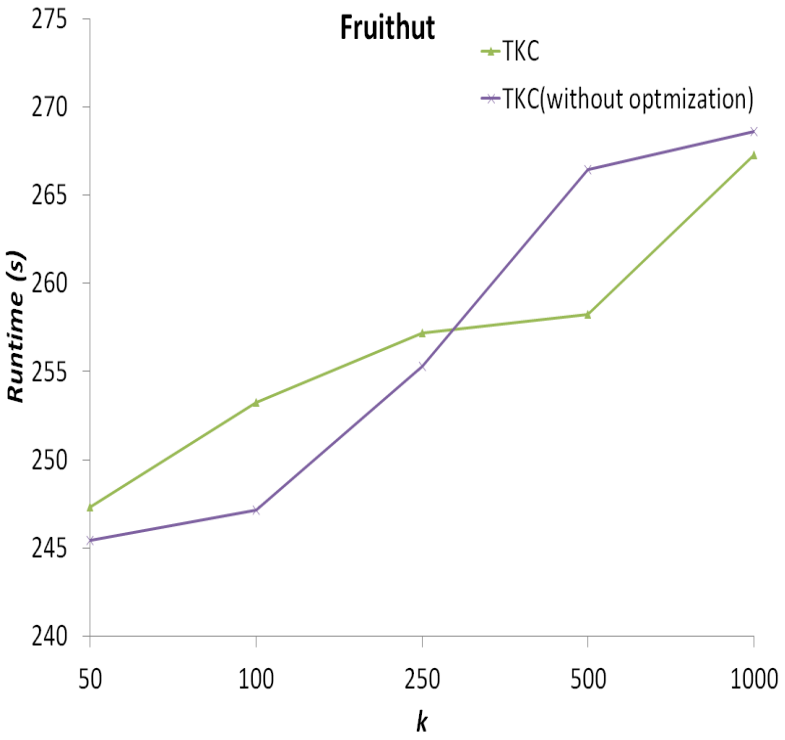
- |D|: number of transactions
- |I|: number of distinct items
- |GI|: number of distinct generalized items
- T_{min}: minimum transaction length
- T_{max}: maximum transaction length
- T_{avg}: average transaction length
- $Level$: the maximum level of items

Both **Liquor** and **Fruithut** are real-life datasets.

Influence of Optimization with variation of K

- **Execution times**

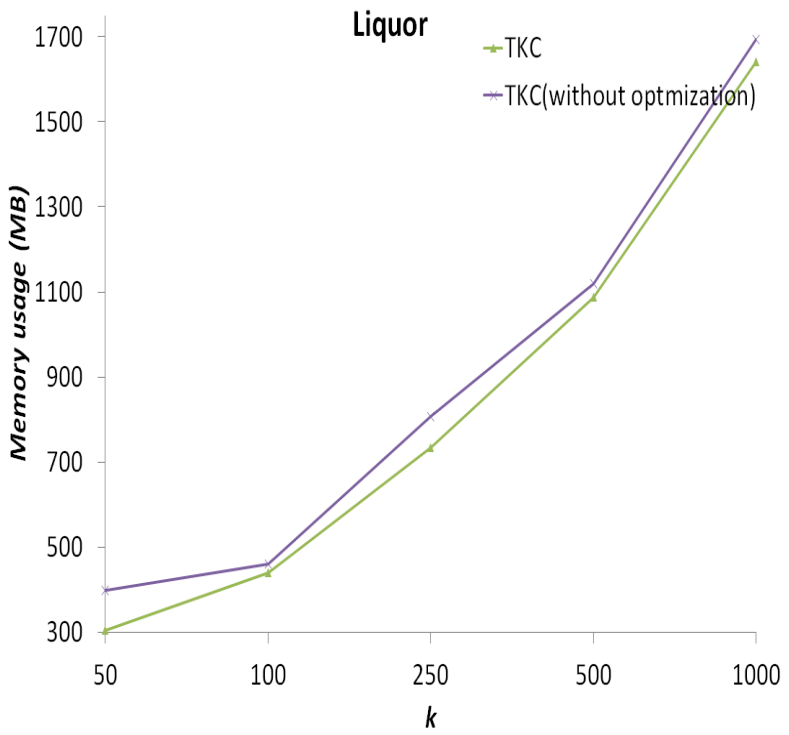


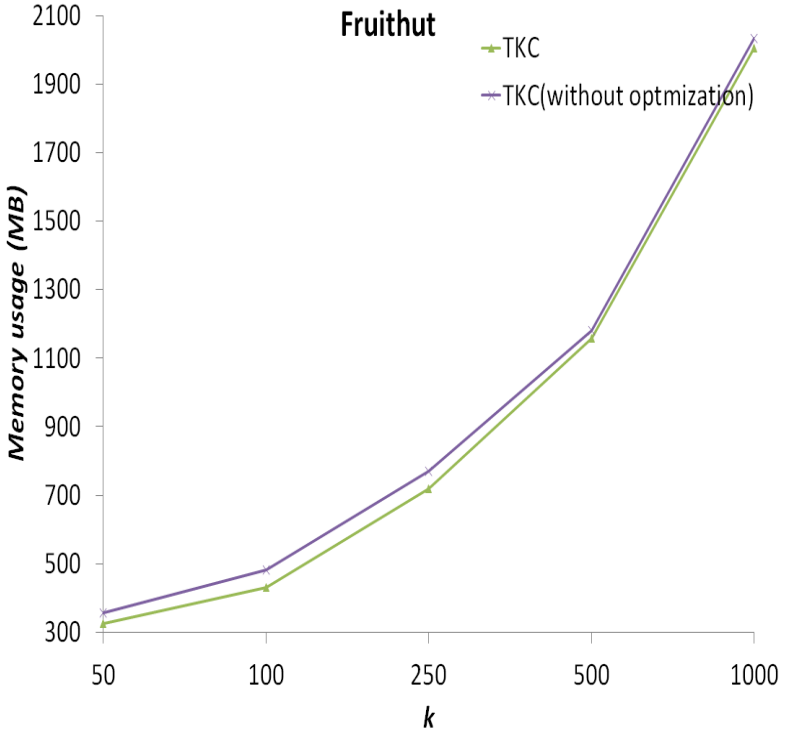


- Increasing *minutil* often decrease the runtime.
- As **k** is increased, **TKC with Optimization** becomes faster that TKC without optimization.

Influence of Optimization with variation of K

- **Peak memory usage**

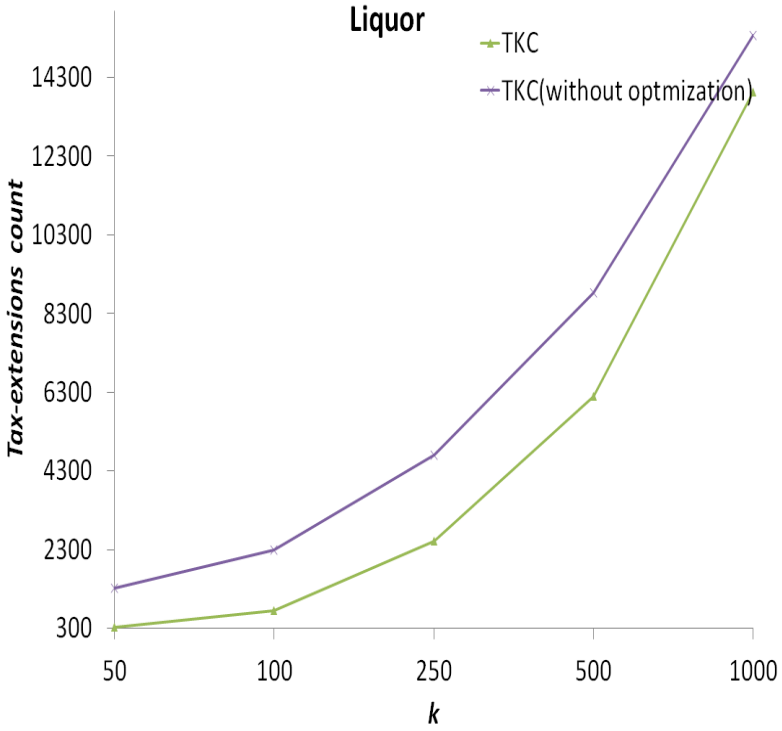


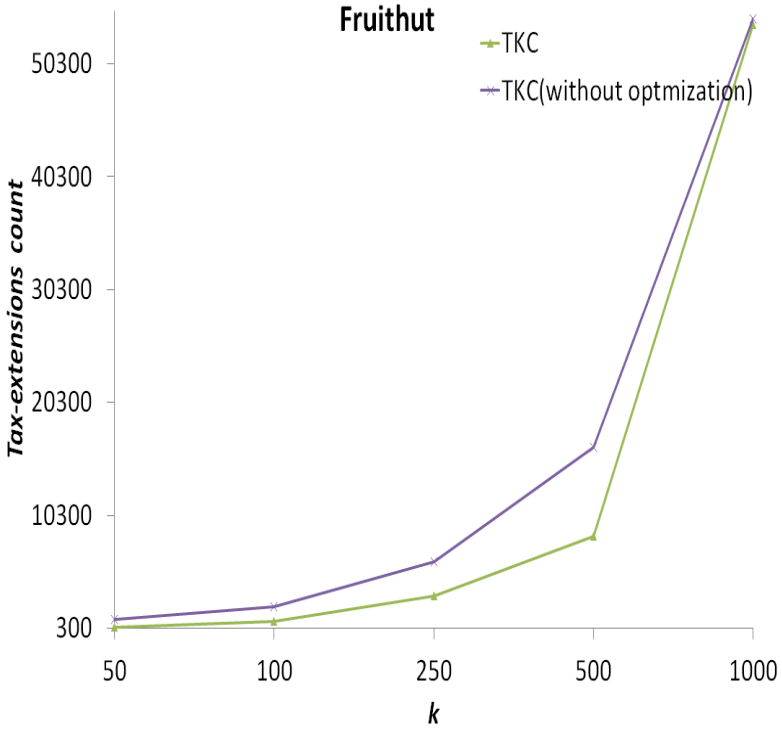


- **Optimization 1** allows to reduce the memory consumption.

Influence of Optimization with variation of K

- **Tax-extensions count**

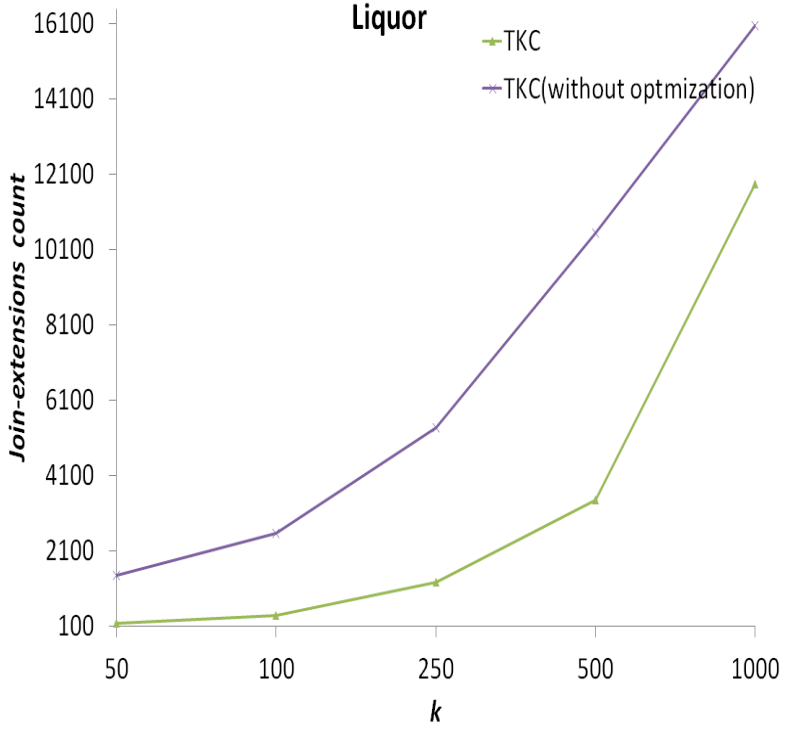


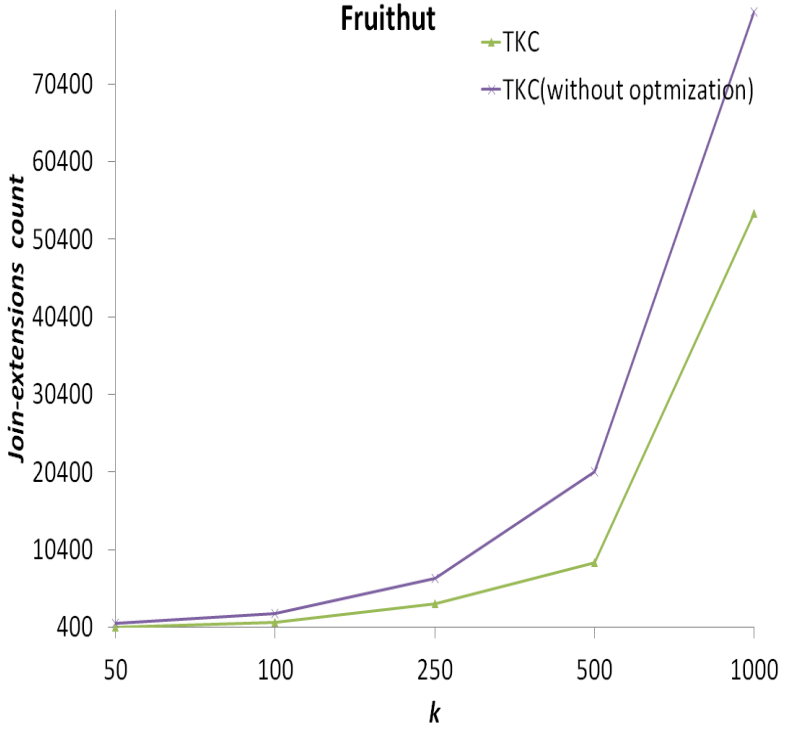


- The number of tax-extensions increases rapidly with an increase of **k**.
- **TKC with optimization** generates **less tax-extensions** than TKC without optimization.

Influence of Optimization with variation of K

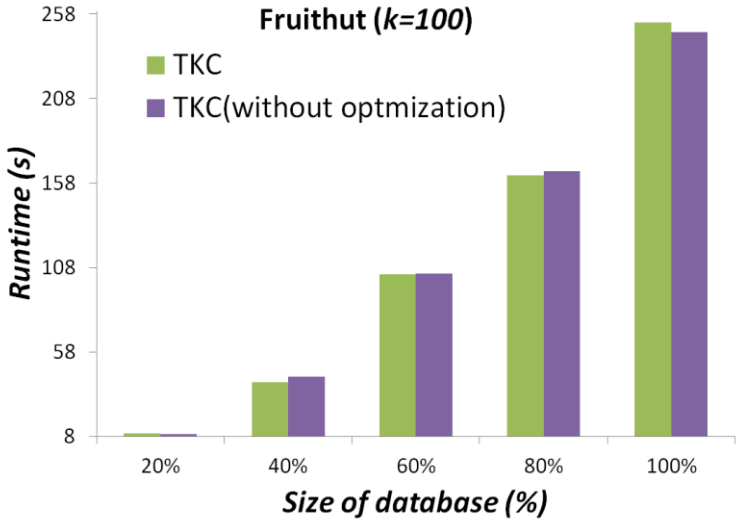
- **Join-extensions count**

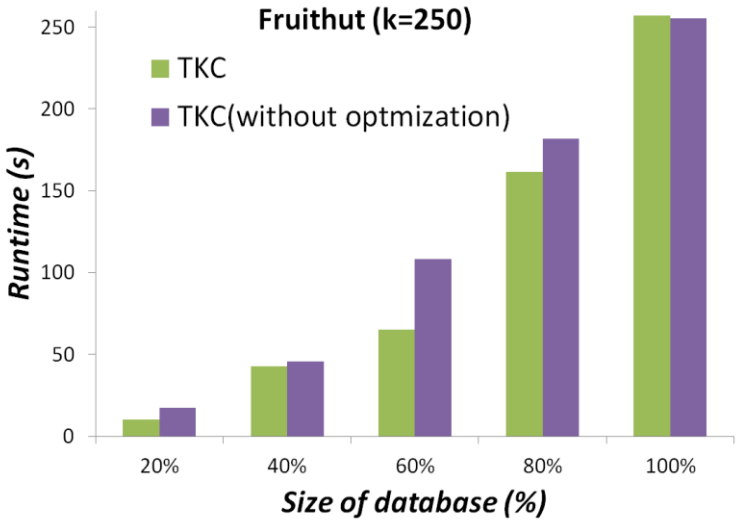


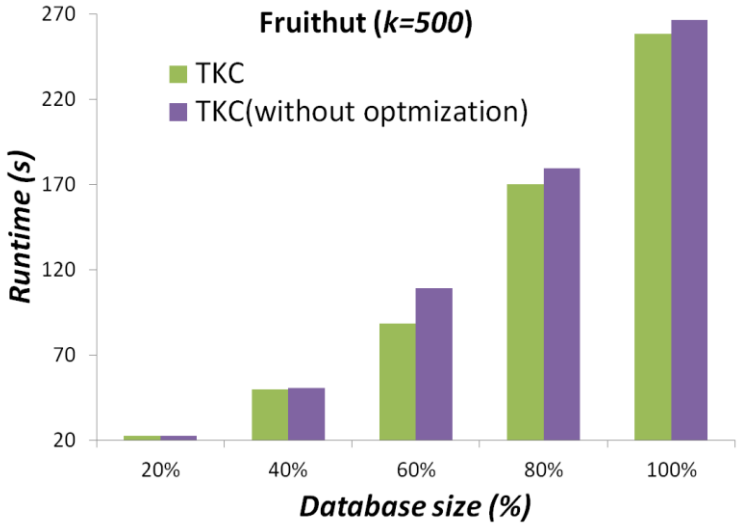


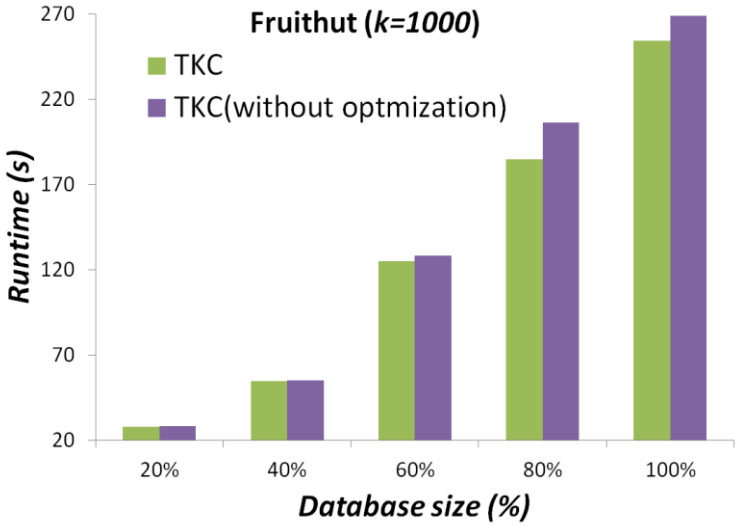
- **TKC with optimization 1** allows to greatly reduce the number of join-extensions by early pruning unpromising join-extensions.
- **TKC with optimization 1** is more efficient than TKC without optimization especially for Liquor dataset.

Scalability









Comparison of TKC with CLH-Miner

Fruithut

k	*minutil*	Runtime (s)		Memory usage (MB)	
		TKC	CLH-Miner	TKC	CLH-Miner
50	23059527	247.299	238.859	324.479	224.611
100	15945533	253.221	244.73	431.467	315.776
250	9686703	257.158	242.92	718.585	447.521
500	6573408	258.234	249.933	1157.21	640.544
1000	4554291	267.276	254.537	2003.701	852.516

Liquor

k	*minutil*	Runtime (s)		Memory usage (MB)	
		TKC	CLH-Miner	TKC	CLH-Miner
50	5224833.94	141.985	145.49	304.092	254.986
100	3928900.18	154.745	162.871	440.696	338.059
250	2356048.01	158.802	153.373	734.409	503.471
500	1613227.04	169.986	165.827	1086.902	701.085
1000	1112973.16	181.29	155.899	1640.205	920.048

- CLHMiner is not much faster that TKC for all values of *k*.

Benefit of selecting TKC

Fruithut

Fruithut (Total utility: 261871526)			
k_1-k_2	*minutil* for k_1	*minutil* for k_2	Interval size (%)
50-100	23059527	15945533	**2,71 %**
100-250	15945533	9686703	**2,39 %**
250-500	9686703	6573408	**1,18 %**
500-1000	6573408	4554291	**0,77 %**

Liquor

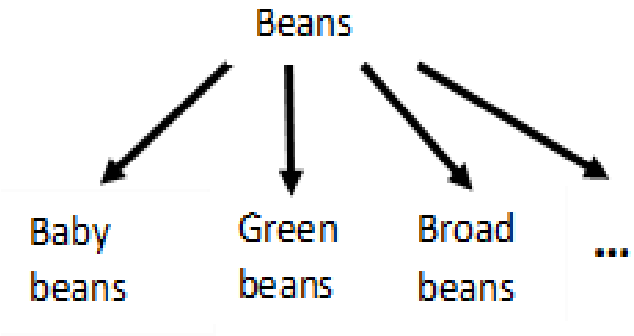
Liquor (Total utility: 22394937,43)			
k_1-k_2	*minutil* for k_1	*minutil* for k_1	Interval size (%)
50-100	5224833,94	3928900,19	**5,78 %**
100-250	3928900,19	2356048,02	**7,02 %**
250-500	2356048,02	1613227,04	**0,28 %**
500-1000	1613227,04	1112973,17	**0,19 %**

- it is recommended to use TKC instead of using CLH-Miner to obtain exactly the top-k desired cross-level HUIs.

Examples of patterns found by TKC

- Dataset= fruithut, k=1000, TKC finds the pattern *{Beans, Cherries}.*

Beans is a generalized item.



- Dataset= liquor, k=1000, TKC find the pattern *{Cordials& Liqueurs, Neutral Grain Spirits}*.

Cordials& Liqueurs and *Neutral Grain Spirits* are generalized items.

Conclusion

- Contributions
  - Cross-level high utility itemsets is redefined as **top-k cross level high utility itemsets.**
  - An efficient algorithm, named **TKC.**
  - Three pruning strategies are adopted: **GWU, REU and Optimization1.**
- Future work
  - Consider other types of user-defined taxonomy-based constraints.
  - Consider other types of patterns such as **sequential patterns** and **episodes.**
  - Mine cross-level patterns in other domains such as **social network mining** and **computer networks**.

Thank you. Questions?

Source code and datasets: SPMF data mining library
http://www.philippe-fournier-viger.com/spmf/