EFIM: A Fast and Memory Efficient Algorithm for High-Utility Itemset Mining

Souleymane Zida[1], Philippe Fournier-Viger[2], Jerry Chun-Wei Lin[3], Cheng-Wei Wu[4] and Vincent S. Tseng[2]

[1]Department of Computer Science, University of Moncton, Moncton NB, Canada;
[2]School of Natural Sciences and Humanities, Harbin Institute of Technology (Shenzhen), Shenzhen GD, China;
[3]School of Computer Science and Technology, Harbin Institute of Technology (Shenzhen), Shenzhen GD, China;
[4]Department of Computer Science, National Chiao Tung University, Taiwan

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)*

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 $\text{utility} \geq minutil$)

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

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 the 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$**

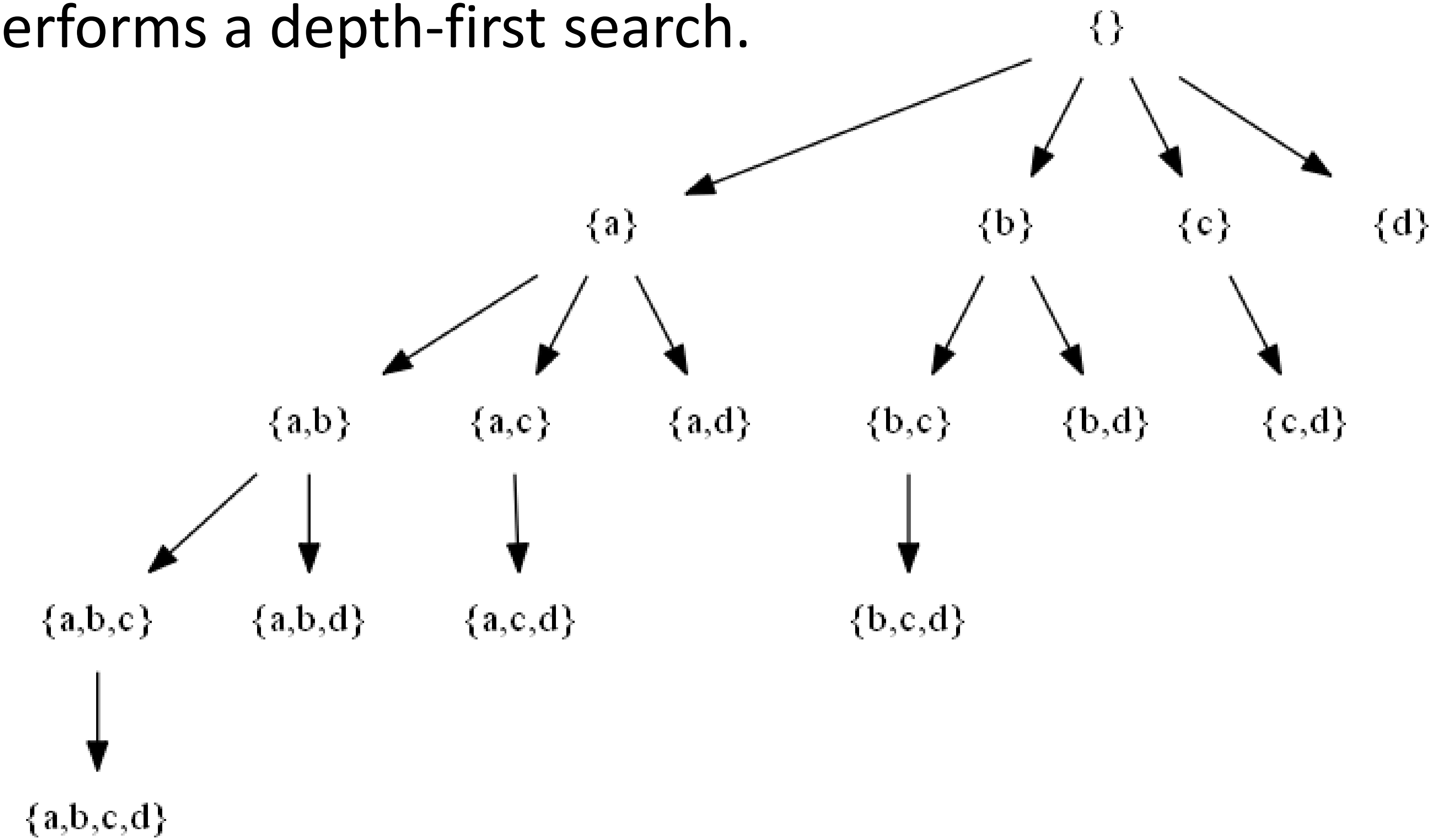
Challenge: utility is not anti-monotonic

Contribution

- A new algorithm named **EFIM**
- Three main ideas:
  - High-utility Database Projection
  - High-utility Transaction Merging
  - Utility-Bin Array to calculate utility and upper-bounds

The EFIM algorithm

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



- It applies pruning strategies to prune the search space based on upper-bounds on the utility.

Definition 4.2 (Extension of an itemset). *Let be an itemset α. An itemset Z is an* extension *of α (appears in a sub-tree of α in the set-enumeration tree) if $Z = \alpha \cup W$ for an itemset $W \in 2^{E(\alpha)}$ such that $W \neq \emptyset$.*

Definition 4.3 (Single-item extension of an itemset). *Let be an itemset α. An itemset Z is a* single-item extension *of α (is a child of α in the set-enumeration tree) if $Z = \alpha \cup \{z\}$ for an item $z \in E(\alpha)$.*

Example 4.1. Consider the database of our running example and $\alpha = \{d\}$. The set $E(\alpha)$ is $\{e, f, g\}$. Single-item extensions of α are $\{d, e\}$, $\{d, f\}$ and $\{d, g\}$. Other extensions of α are $\{d, e, f\}$, $\{d, f, g\}$ and $\{d, e, f, g\}$.

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	$(b, 2), (c, 2), (e, 1), (g, 2)$

Database projection

Original database

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

Projected database of {c}

TID	Transaction
T_1	$(d,3),(e,1),(f,5)$
T_2	$(d,3),(e,1)$
T_3	$(d,1)$
T_4	$(e,2),(g,5)$
T_5	$(e,1),(g,2)$

EFIM is a pattern-growth algorithm

It scans the horizontal database to calculate the utility and upper-bound

Using projected databases reduce the cost of database scans.

To reduce memory usage, EFIM performs <u>pseudo</u>-projections.

Database merging

Original 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	$(b,2),(c,2),(e,1),(g,2)$

To further reduce the database, we can merge transactions:

- In the original database
- In each projected database

Projected database of {c}

TID	Transaction
T_1	$(d,3),(e,1),(f,5)$
T_2	$(d,3),(e,1)$
T_3	$(d,1)$
T_4	$(e,2),(g,5)$
T_5	$(e,1),(g,2)$

Merged database of {c}

TID	Transaction
T_1	$(d,3),(e,1),(f,5)$
T_2	$(d,3),(e,1)$
T_3	$(d,1)$
T_{45}	$(e,3),(g,7)$

Optimization 1 - merging

- **An idea**: merge identical transactions to reduce the size of the database.
- This can reduce the size of the database and the cost of reading the database.

	Items	Weight
T1	{A,B,C,D,E,F}	1
T2	{A,B,C,D,E,F}	1
T3	{C,D,E, F}	1
T4	{B, E, F}	1

→

	Items	Weight
TX	{A,B,C,D,E,F}	2
T3	{C,D,E, F}	1
T4	{B, E, F}	1

- If we apply transaction merging only on the original database, this optimization will not make the algorithm much faster.

Optimization 1 – merging

A better idea

- Merge transactions avec each projection.
- For example, after having projected the database with an item D:

Prefix = {D}

	Items	Weight
T1	{~~A,B,C,D,~~E,F}	1
T2	{~~A,B,C,D,~~E,F}	1
T3	{~~C,D,~~E, F}	1
T4	{~~B,~~ E, F}	1

Optimization 1 – merging

A better idea

- Merge transactions avec each projection.
- For example, after having projected the database with an item D:

Prefix = {D}

	Items	Weight
T1	{~~A,B,C,D,~~E,F}	1
T2	{~~A,B,C,D,~~E,F}	1
T3	{~~C,D,~~E, F}	1
T4	{~~B,~~ E, F}	1

→

	Items	Weight
T1	{E,F}	4

→Good, but we must be able to implement transaction merging efficiently!

Optimization 1 – merging

Solution: Sort transactions by the total order when transactions are read backward.

e.g.

	Items	Weight
T1	{A,B,C,D,E,F}	1
T2	{A,B,C,D,E,F}	1
T3	{C,D,E, F}	1
T4	{D, A, G}	1
T5	{B, C, A, G}	1
T6	{A, E, F}	1
T7	{B, C, F, G}	1
T8	{A, C, D, F, G}	1

sort →

	Items	Weight
T6	{A, E, F}	1
T3	{C,D,E, F}	1
T1	{A,B,C,D,E,F}	1
T2	{A,B,C,D,E,F}	1
T5	{B, C, A, G}	1
T4	{D, A, G}	1
T7	{B, C, F, G}	1
T8	{A, C, D, F, G}	1

Optimization 1 – merging

Solution: Sort transactions by the total order when transactions are read backward.

e.g.

Property: After the sort, no matter where we « cut » transactions, transactions to be merged will always appear one after the other.

e.g. consider **prefix = {a}**

	Items	Weight
T6	{A, E, F}	1
T3	{C,D,E, F}	1
T1	{A,B,C,D,E,F}	1
T2	{A,B,C,D,E,F}	1
T5	{B, C, A, G}	1
T4	{D, A, G}	1
T7	{B, C, F, G}	1
T8	{A, C, D, F, G}	1

Optimization 1 – merging

Solution: Sort transactions by the total order, but by reading the transactions backwards.

e.g.

Property: After the sort, no matter where we « cut » transactions, transactions to be merged will always appear one after the other.

e.g. consider **prefix = {a}**

	Items	Weight
T6	~~{A,~~ E, F}	1
T3	~~{C,D,E, F}~~	1
T1	~~{A,~~**B,C,D,E,F}**	1
T2	~~{A,~~**B,C,D,E,F}**	1
T5	~~{B, C, A,~~ **G}**	1
T4	~~{D, A,~~ **G}**	1
T7	~~{B, C, F, G}~~	1
T8	~~{A~~, C, D, F, G}	1

Optimization 1 – merging

Solution: Sort transactions by the total order, but by reading the transactions backwards.

e.g.

	Items	Weight
T6	{~~A~~, E, F}	1
TX	**{~~A~~,B,C,D,E,F}**	**2**
TY	**{~~B, C, A,~~ G}**	**2**
T8	{~~A~~, C, D, F, G}	1

Property: After the sort, no matter where we « cut » transactions, transactions to be merged will always appear one after the other.

e.g. consider **prefix = {a}**

Definition 4.6 (Identical transactions). *A transaction* T_a *is* identical to *a transaction* T_b *if it contains the same items as* T_b *(i.e.* $T_a = T_b$*). It is important to note that in this definition, two identical transactions are not required to have the same internal utility values.*

Definition 4.7 (Transaction merging). Transaction merging *consists of replacing a set of identical transactions* $Tr_1, Tr_2, ...Tr_m$ *in a database* D *by a single new transaction* $T_M = Tr_1 = Tr_2 = ... = Tr_m$ *where the quantity of each item* $i \in T_M$ *is defined as* $q(i, T_M) = \sum_{k=1...m} q(i, Tr_k)$.

Definition 4.8 (Projected transaction merging). Projected transaction merging *consists of replacing a set of identical transactions* $Tr_1, Tr_2, ...Tr_m$ *in a projected database* α-D *by a single new transaction* $T_M = Tr_1 = Tr_2 = ... = Tr_m$ *where the quantity of each item* $i \in T_M$ *is defined as* $q(i, T_M) = \sum_{k=1...m} q(i, Tr_k)$.

Transaction merging is obviously desirable. However, a key problem is to implement it efficiently. The naive approach to identify identical transactions is to compare all transactions with each other. But this is inefficient because it requires $O((nw)^2)$ time. To find identical transactions in $O(nw)$ time, we propose the following novel approach. We initially sort the original database according to a new total order $\succ_T$ on transactions. Sorting is achieved in $O(nw\ log(nw))$ time. However, this cost is generally negligible compared to the other operations performed by the algorithm because it is performed only once.

Definition 4.9 (Total order on transactions). *The $\succ_T$ order is defined as the lexicographical order when the transactions are read backwards. Formally, let there be two transactions $T_a = \{i_1, i_2, ...i_m\}$ and $T_b = \{j_1, j_2, ...j_k\}$. The total order $\succ_T$ is defined by four cases. The first case is that $T_b \succ T_a$ if both transactions are identical and the TID of T_b is greater than the TID of T_a. The second case is that $T_b \succ_T T_a$ if $k > m$ and $i_{m-x} = j_{k-x}$ for any integer x such that $0 \leq x < m$. The third case is that $T_b \succ_T T_a$ if there exists an integer x such that $0 \leq x < \min(m, k)$, where $j_{k-x} \succ i_{m-x}$ and $i_{m-y} = j_{k-y}$ for all integer y such that $x < y < \min(m, k)$. The fourth case is that otherwise $T_a \succ_T T_b$.*

Example 4.4. Consider three transactions $T_x = \{b, c\}$, $T_y = \{a, b, c\}$ and $T_z = \{a, b, e\}$. We have that $T_z \succ_T T_y \succ_T T_x$.

A database sorted according to the $\succ_T$ order provides the following property.

Property 4.1 (Transaction order in an $\succ_T$ sorted database). Let there be a $\succ_T$ sorted database D and an itemset α. Identical transactions appear consecutively in the projected database α-D.

Pruning the search space using the utility

Merged database of {c}

TID	Transaction
T_1	$(d,3),(e,1),(f,5)$
T_2	$(d,3),(e,1)$
T_3	$(d,1)$
T_{45}	$(e,3)\ (g,7)$

Sub-tree utility upper-bound:

An upper-bound on the utility of itemsets that can be obtained starting with {c} is the utility of {c} plus the profit of items appearing after {c} that can be appended to {c}.

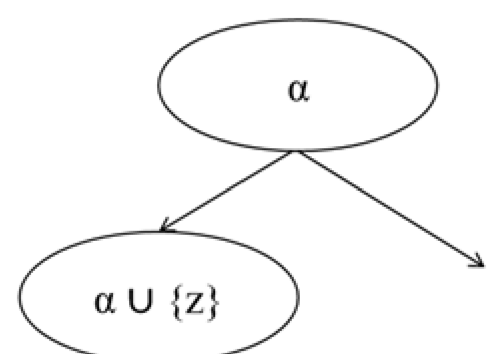
Upper-bound – local utility

Definition 3.3 (Remaining utility). *Let $\succ$ be a total order on items from I (e.g. the lexicographical order), and X be an itemset. The* remaining utility *of X in a transaction T_c is defined as $re(X, T_c) = \sum_{i \in T_c \wedge i \succ x \forall x \in X} u(i, T_c)$. Since*

Definition 4.10 (Local utility). *Let be an itemset α and an item $z \in E(\alpha)$. The* Local Utility *of z w.r.t. α is $lu(\alpha, z) = \sum_{T \in g(\alpha \cup \{z\})} [u(\alpha, T) + re(\alpha, T)]$.*

Property 4.2 (Overestimation using the local utility). Let be an itemset α and an item $z \in E(\alpha)$. Let Z be an extension of α such that $z \in Z$. The relationship $lu(\alpha, z) \geq u(Z)$ holds.

Example 4.1 (Pruning an item in all sub-trees using the local utility). *Let be an itemset α and an item $z \in E(\alpha)$. If $lu(\alpha, z) < minutil$, then all extensions of α containing z are low-utility. In other words, item z can be ignored when exploring all sub-trees of α.*



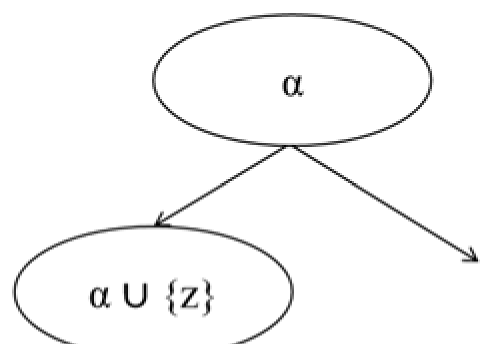
Upper-bound – subtree utility

Definition 4.11 (Sub-tree utility). *Let be an itemset α and an item z that can extend α according to the depth-first search ($z \in E(\alpha)$). The* Sub-tree Utility *of z w.r.t. α is*
$su(\alpha, z) = \sum_{T \in g(\alpha \cup \{z\})} [u(\alpha, T) + u(z, T) + \sum_{i \in T \wedge i \in E(\alpha \cup \{z\}} u(i, T)]$.

Property 4.3 (Overestimation using the sub-tree utility). Let be an itemset α and an item $z \in E(\alpha)$. The relationship $su(\alpha, z) \geq u(\alpha \cup \{z\})$ holds. And more generally, $su(\alpha, z) \geq u(Z)$ holds for any extension Z of $\alpha \cup \{z\}$.

Property 4.2 (Pruning a sub-tree using the sub-tree utility). *Let be an itemset α and an item $z \in E(\alpha)$. If $su(\alpha, z) < minutil$, then the single item extension $\alpha \cup \{z\}$ and its extensions are low-utility. In other words, the sub-tree of $\alpha \cup \{z\}$ in the set-enumeration tree can be pruned.*

Property 4.4 (Relationships between upper-bounds). Let be an itemset α, an item z and an itemset $Y = \alpha \cup \{z\}$. The relationship $TWU(Y) \geq lu(\alpha, z) \geq reu(Y) = su(\alpha, z)$ holds.



About the su upper-bound, one can ask what is the difference between this upper-bound and the reu upper-bound of HUI-Miner and FHM since they are mathematically equivalent. The major difference between the remaining-utility upper bound and the proposed su upper-bound is that the su upper-bound is calculated when the depth-first search is at itemset α in the search tree rather than at the child itemset Y. Thus, if $su(\alpha, z) < minutil$, EFIM prunes the whole sub-tree of z including node Y rather than only pruning the descendant nodes of Y. This is illustrated in Fig. 2, which compares the nodes pruned in the sub-tree of Y using the su and reu upper-bounds. Thus, as explained here, using su instead of reu upper-bound, is more effective for pruning the search space.

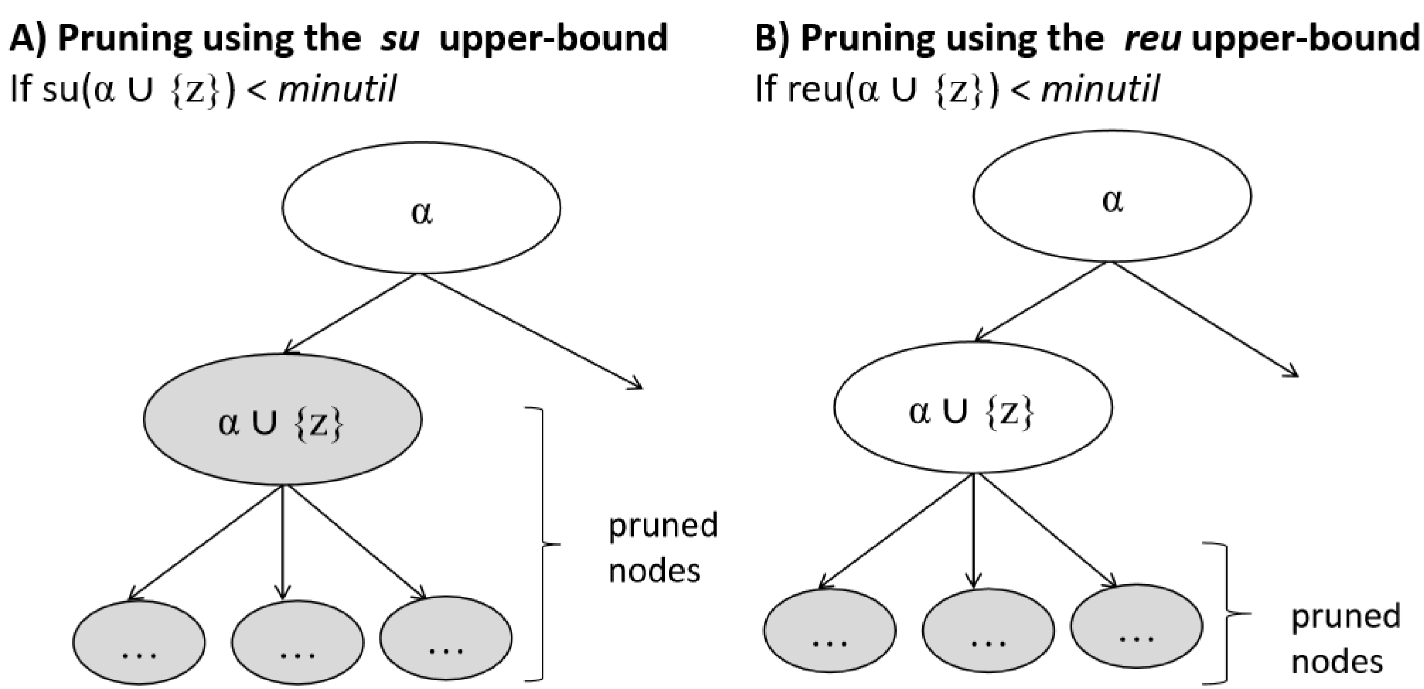


Fig. 2. Comparison of the su and reu upper-bounds

Redefined sub-tree utility

Moreover, we make the *su* upper-bound even tighter by redefining it as follows. This leads to the final *revised sub-tree utility upper-bound* used in the proposed EFIM algorithm.

Definition 4.12 (Primary and secondary items). *Let be an itemset* α*. The* primary items of α *is the set of items defined as* $Primary(\alpha) = \{z|z \in E(\alpha) \wedge su(\alpha, z) \geq minutil\}$*. The* secondary items of α *is the set of items defined as* $Secondary(\alpha) = \{z|z \in E(\alpha) \wedge lu(\alpha, z) \geq minutil\}$*. Because* $lu(\alpha, z) \geq su(\alpha, z)$*,* $Primary(\alpha) \subseteq Secondary(\alpha)$*.*

Definition 4.12 (Primary and secondary items). *Let be an itemset* α*. The* primary items of α *is the set of items defined as* $Primary(\alpha) = \{z|z \in E(\alpha) \wedge su(\alpha, z) \geq minutil\}$*. The* secondary items of α *is the set of items defined as* $Secondary(\alpha) = \{z|z \in E(\alpha) \wedge lu(\alpha, z) \geq minutil\}$*. Because* $lu(\alpha, z) \geq su(\alpha, z)$*,* $Primary(\alpha) \subseteq Secondary(\alpha)$*.*

Example 4.7. Consider the running example and $\alpha = \{a\}$. $Primary(\alpha) = \{c, e\}$. $Secondary(\alpha) = \{c, d, e\}$. This means that w.r.t. α, only the sub-trees rooted at nodes $\alpha \cup \{c\}$ and $\alpha \cup \{e\}$ should be explored. Furthermore, in these subtrees, no items other than c, d and e should be considered.

Redefined sub-tree utility

Definition 4.13 (Redefined Sub-tree utility). *Let be an itemset α and an item z. The redefined sub-tree utility of item z w.r.t. itemset α is defined as:* $su(\alpha, z) = \sum_{T \in g(\alpha \cup \{z\})} [u(\alpha, T) + u(z, T) + \sum_{i \in T \wedge i \in E(\alpha \cup \{z\} \wedge i \in Secondary(\alpha)} u(i, T)]$.

The difference between the su upper-bound and the redefined su upper-bound is that in the latter, items not in $Secondary(\alpha)$ will not be included in the calculation of the su upper-bound. Thus, this redefined upper-bound is always less than or equal to the original su upper-bound and the reu upper-bound. It

Counting upper-bounds, utility and support using arrays

Merged database of {c}

TID	Transaction
T_1	$(d,3),(e,1),(f,5)$
T_2	$(d,3),(e,1)$
T_3	$(d,1)$
T_{45}	$(e,3)(g,7)$

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

Utility

a	b	c	d	e	f	g
0	0	0	14	15	5	7

Sub-tree upper-bound

a	b	c	d	e	f	g
0	0	0	11	12	0	0

Support

a	b	c	d	e	f	g
0	0	0	3	3	1	1

	U[a]	U[b]	U[c]	U[d]	U[e]	U[f]	U[g]
A) Initialization	0	0	0	0	0	0	0
B) After reading transaction T_1	8	0	8	8	0	0	0
C) After reading transaction T_2	35	0	35	8	27	0	27
D) After reading transaction T_3	65	30	65	38	57	30	27
E) After reading transaction T_4	65	50	85	58	77	30	27
F) After reading transaction T_5	65	61	96	58	88	30	38

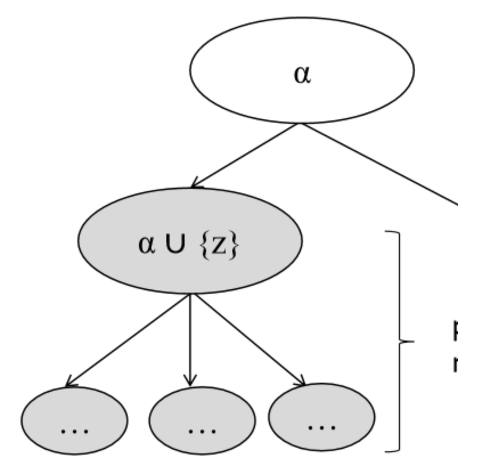
Fig. 3. Calculating the TWU using a utility-bin array

EFIM

Algorithm 1: The EFIM algorithm

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

$\alpha = \emptyset$;
Calculate $lu(\alpha, i)$ for all items $i \in I$ by scanning D, using a utility-bin array;
$Secondary(\alpha) = \{i | i \in I \wedge lu(\alpha, i) \geq minutil\}$;
Let $\succ$ be the total order of TWU ascending values on $Secondary(\alpha)$;
Scan D to remove each item $i \notin Secondary(\alpha)$ from the transactions, sort items in each transaction according to $\succ$, and delete empty transactions;
Sort transactions in D according to $\succ_T$;
Calculate the sub-tree utility $su(\alpha, i)$ of each item $i \in Secondary(\alpha)$ by scanning D, using a utility-bin array;
$Primary(\alpha) = \{i | i \in Secondary(\alpha) \wedge su(\alpha, i) \geq minutil\}$;
`Search` $(\alpha, D, Primary(\alpha), Secondary(\alpha), minutil)$;



EFIM (2)

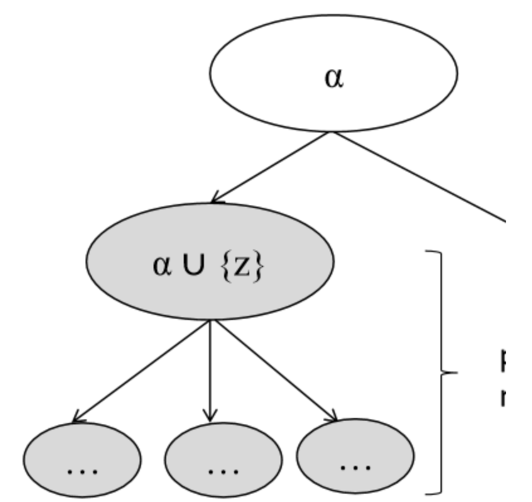
Algorithm 2: The *Search* procedure

input : α: an itemset, α-D: the α projected database, $Primary(\alpha)$: the primary items of α, $Secondary(\alpha)$: the secondary items of α, the $minutil$ threshold

output: the set of high-utility itemsets that are extensions of α

foreach $item\ i \in Primary(\alpha)$ **do**
- $\beta = \alpha \cup \{i\}$;
- Scan α-D to calculate $u(\beta)$ and create β-D; `// uses transaction merging`
- **if** $u(\beta) \geq minutil$ **then** output β;
- Calculate $su(\beta, z)$ and $lu(\beta, z)$ for all item $z \in Secondary(\alpha)$ by scanning β-D once, using two utility-bin arrays;
- $Primary(\beta) = \{z \in Secondary(\alpha) | su(\beta, z) \geq minutil\}$;
- $Secondary(\beta) = \{z \in Secondary(\alpha) | lu(\beta, z) \geq minutil\}$;
- `Search` $(\beta,\ \beta\text{-}D,\ Primary(\beta),\ Secondary(\beta),\ minutil)$;

end



Experimental Evaluation

Datasets' characterictics

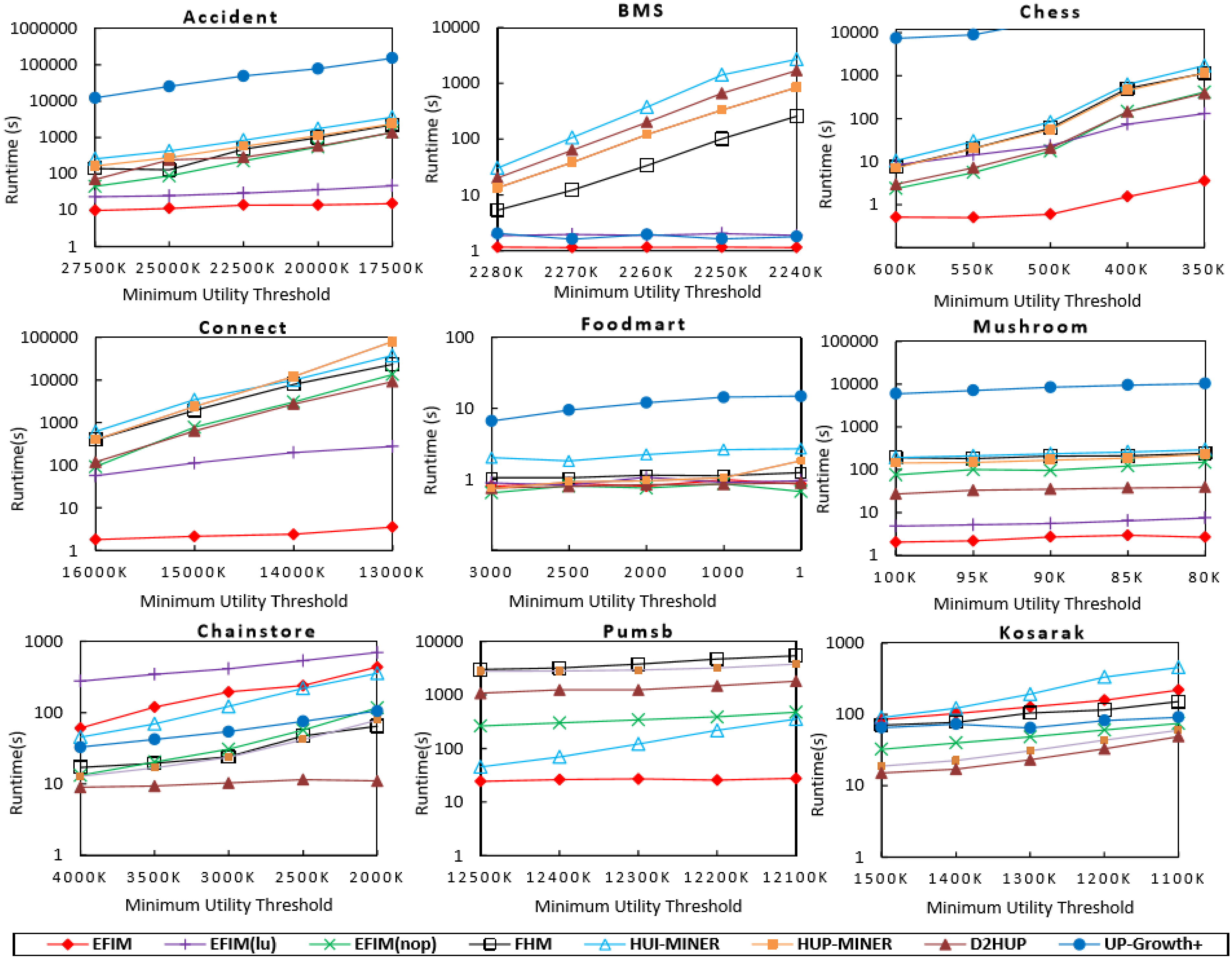
Dataset	transaction count	distinct item count	average transaction length
Accidents	340,183	468	33.8
BMS	59,601	497	4.8
Chess	3,196	75	37
Connect	67,557	129	43
Foodmart	4,141	1,559	1,559
Mushroom	8,124	119	23

- **Foodmart** is a real-life transaction datasets from retail stores.
- External and internal utility values have been generated in the [1, 000] and [1, 5] intervals using a log-normal distribution

Experimental Evaluation

- We compared the performance of EFIM-Closed with
  - the state-of-the-art CHUD algorithm
  - two versions of EFIM-Closed without optimizations named EFIM-Closed(lu) and EFIM-Closed(nop)

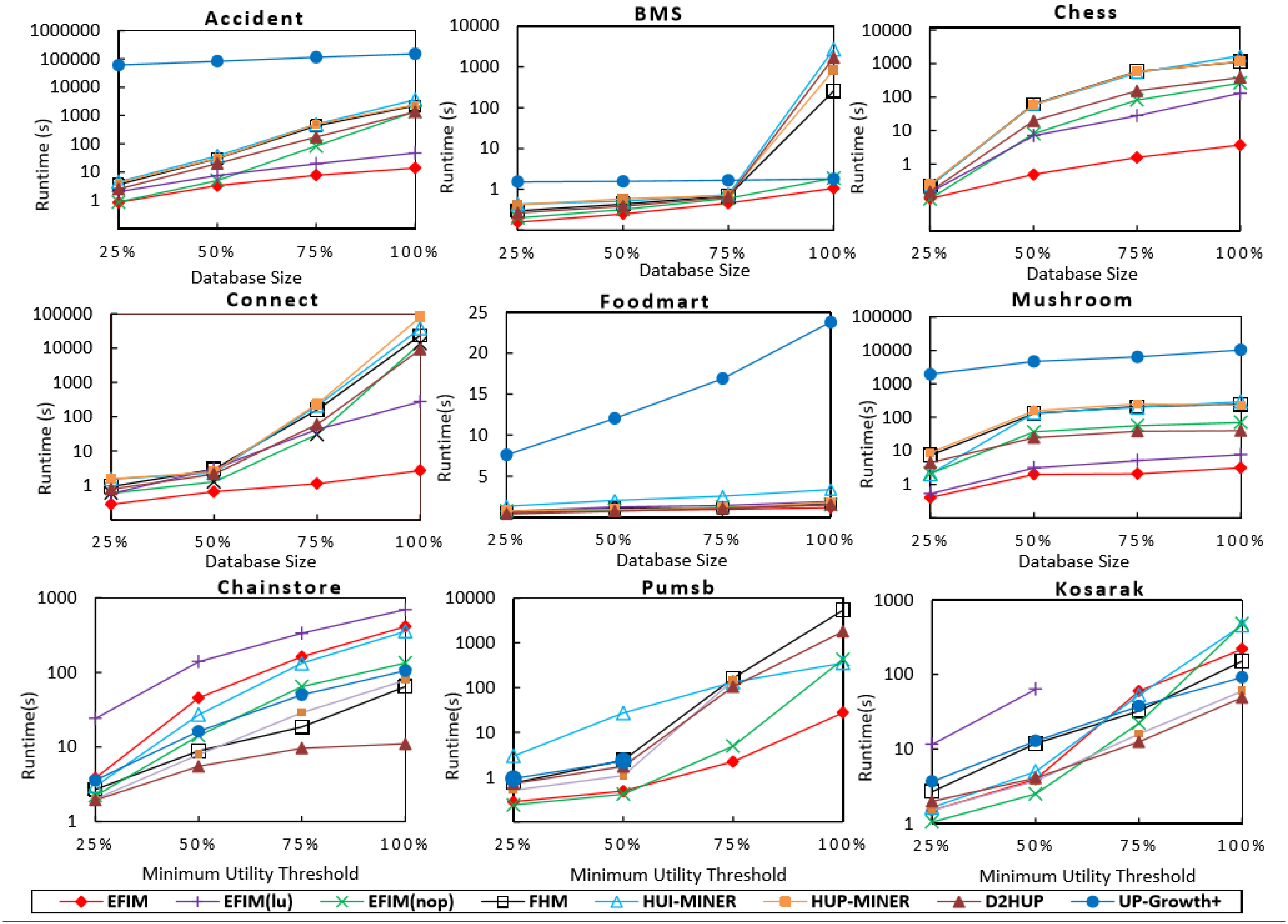
- We varied the *minutil* threshold and compared execution time and memory usage
- Computer with 12 GB of RAM, Java, Windows 7, 64 bit Core i5 Processor



Accident
Runtime (s)
1000000
100000
10000
1000
100
10
1
27500K 25000K 22500K 20000K 17500K
Minimum Utility Threshold
BMS
Runtime (s)
10000
1000
100
10
1
2280K 2270K 2260K 2250K 2240K
Minimum Utility Threshold
Chess
Runtime (s)
10000
1000
100
10
1
600K 550K 500K 400K 350K
Minimum Utility Threshold
Connect
Runtime(s)
100000
10000
1000
100
10
1
16000K 15000K 14000K 13000K
Minimum Utility Threshold
Foodmart
Runtime (s)
100
10
1
3000 2500 2000 1000 1
Minimum Utility Threshold
Mushroom
Runtime (s)
100000
10000
1000
100
10
1
100K 95K 90K 85K 80K
Minimum Utility Threshold
Chainstore
Runtime(s)
1000
100
10
1
4000K 3500K 3000K 2500K 2000K
Minimum Utility Threshold
Pumsb
Runtime(s)
10000
1000
100
10
1
12500K 12400K 12300K 12200K 12100K
Minimum Utility Threshold
Kosarak
Runtime(s)
1000
100
10
1
1500K 1400K 1300K 1200K 1100K
Minimum Utility Threshold
EFIM
EFIM(lu)
EFIM(nop)
FHM
HUI-MINER
HUP-MINER
D2HUP
UP-Growth+

Table 10. Comparison of maximum memory usage (MB)

Dataset	HUI-MINER	FHM	EFIM	UP-Growth+	HUP-Miner	d^2HUP
Accident	1,656	1,480	**895**	765	1,787	1,691
BMS	210	590	**64**	64	758	282
Chess	405	305	**65**	–	406	970
Connect	2,565	3,141	**385**	–	1,204	1,734
Foodmart	808	211	**64**	819	68	84
Mushroom	194	224	**71**	1,507	196	468
Chainstore	1,164	1,270	**460**	1,058	1,034	878
Pumsb	1,221	1,436	**986**	–	1,021	2,046
Kosarak	1,163	1,409	**576**	1,207	712	1,260



Accident
BMS
Chess
Connect
Foodmart
Mushroom
Chainstore
Pumsb
Kosarak
Runtime (s)
Runtime(s)
Database Size
Minimum Utility Threshold
25%
50%
75%
100%
EFIM
EFIM(lu)
EFIM(nop)
FHM
HUI-MINER
HUP-MINER
D2HUP
UP-Growth+

Table 11. Average projected database size (number of transactions)

Dataset	EFIM	EFIM(nop)	Size reduction (%)
Accident	784	113,304	99.3%
BMS	112.6	204.1	44.8%
Chess	2.6	1363.9	99.8%
Connect	1.4	43687	99.9 %
Foodmart	1.12	1.21	7.1%
Mushroom	1.3	573	99.7%
Chainstore	1,085	1,326	18.1%
Pumsb	1,075	22,326	95.2%
Kosarak	1,727	3,653	53%

Thank you. Questions?

Open source Java data mining software, 120 algorithms
http://www.phillippe-fournier-viger.com/spmf/

EFIM-CLOSED

What is a closed high utility itemset?

It is a **high-utility itemset** that has no proper superset having the same **support** (frequency).

For example:

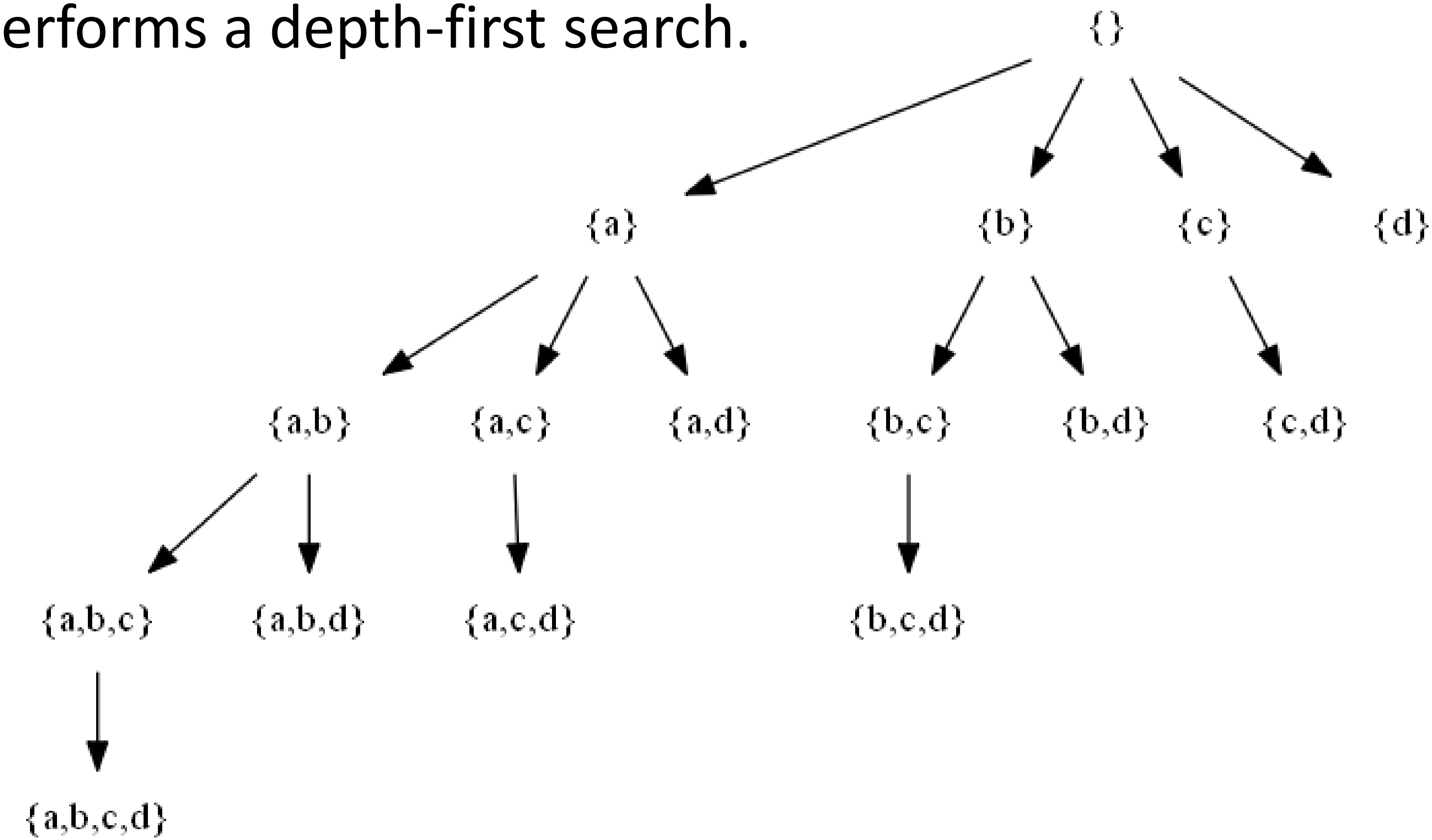
~~{b,d,e} 36$ 2 transactions~~	~~{b,c,d} 34$ 2 transactions~~
{b,c,d,e} 40$ 2 transactions	{b,c,e} 37 $ 3 transactions

Interesting properties:

- A closed itemset is the largest group of items bought by a groups of customers.
- A closed pattern is always more profitable than its corresponding non closed patterns.

The EFIM-Closed algorithm

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



- It applies pruning strategies to prune the search space based on upper-bounds on the utility.

Forward/Backward extension checking

To determine if an itemset **X** is closed, check if there exists an item not in **X** that appears in all transactions where **X** appears.

If yes, then **X** is not closed

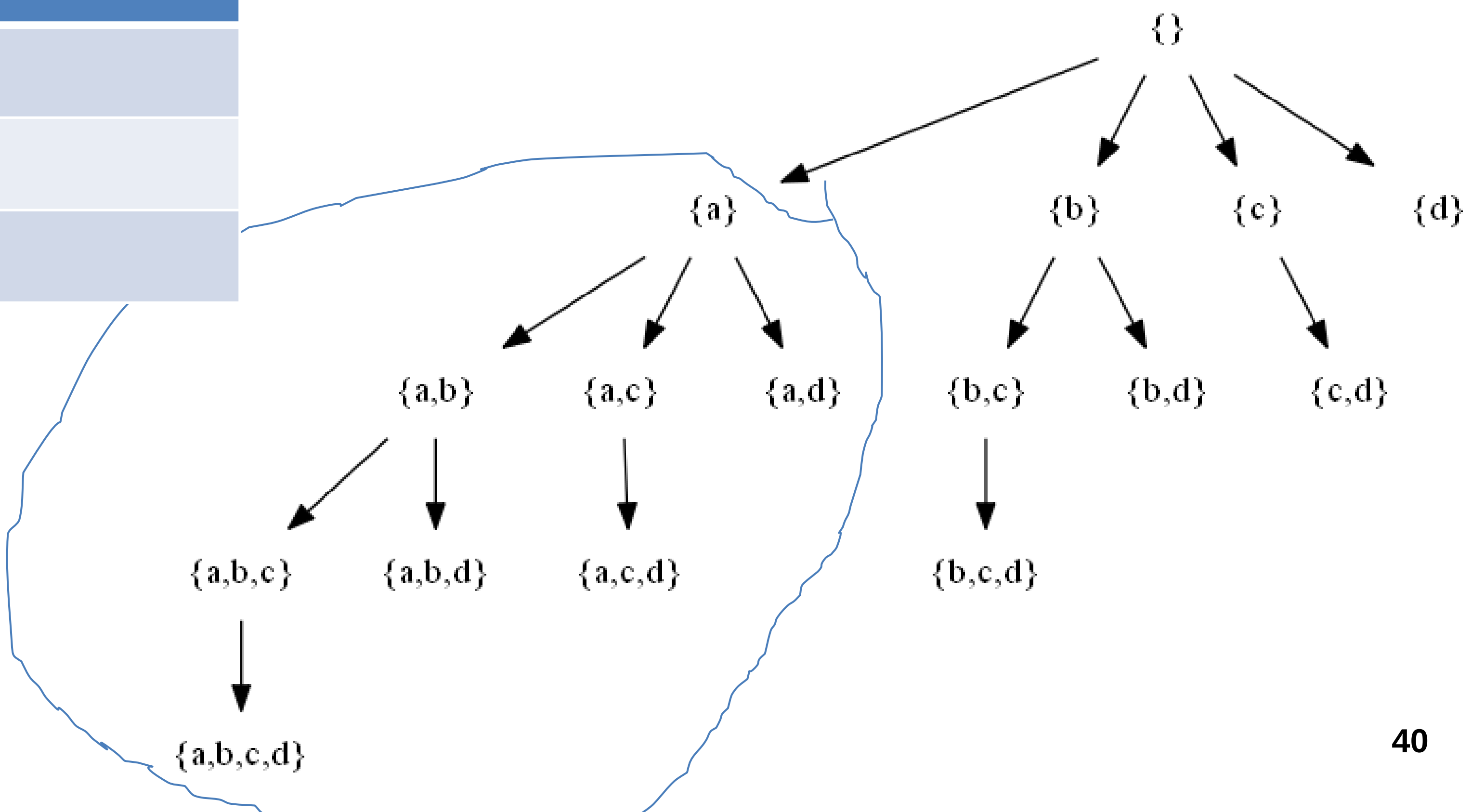
TID	Transaction
T_1	$(a,1),(b,5),(c,1),(d,3),(e,1),(f,5)$
T_2	$(b,4),(c,3),(d,3),(e,1)$
T_3	$(a,1),(c,1),(d,1)$
T_4	$(a,2),(c,6),(e,2),(g,5)$
T_5	$(b,2),(c,2),(e,1),(g,2)$

e.g. {b,d,e} is not closed because of item c

Closure jumping

For an itemset **X**, if all items not in **X** that can be appended to **X** have the same support as **X,** they can be directly appended to **X** to obtain a closed itemset.

TID	Items
T1	{a,b,c, d}
T2	{a,b,c, d}
T3	{b, d}



Pseudocode

Algorithm 1: The EFIM-Closed algorithm

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

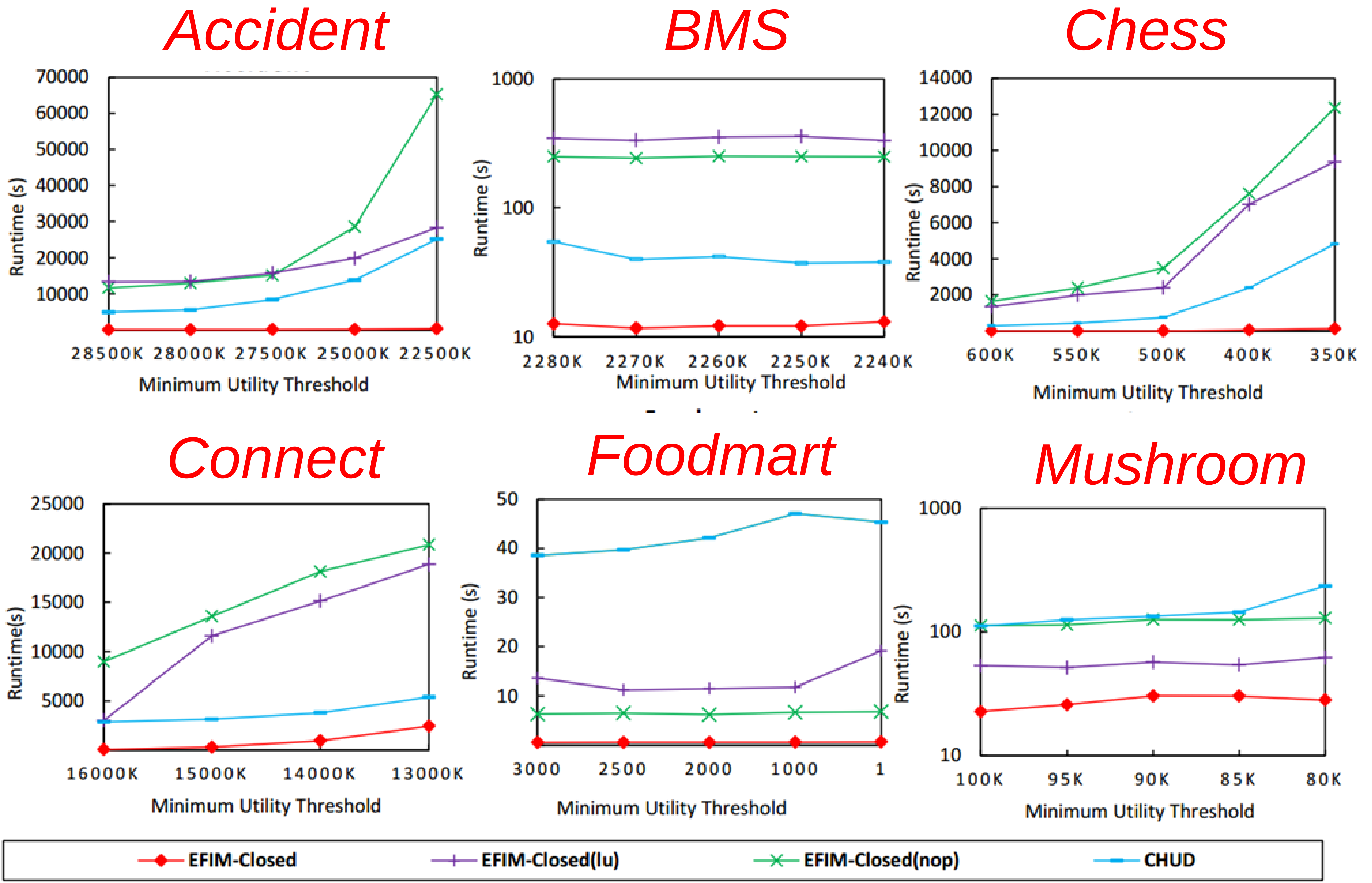
$\alpha = \emptyset$;
Calculate $lu(\alpha, i)$ for all items $i \in I$ by scanning D, using a utility-bin array;
$Secondary(\alpha) = \{i | i \in I \wedge lu(\alpha, i) \geq minutil\}$;
Let $\succ$ be the total order of TWU ascending values on $Secondary(\alpha)$;
Scan D to remove each item $i \notin Secondary(\alpha)$ from th transactions, and delete empty transactions;
Sort transactions in D according to $\succ_T$;
Calculate the sub-tree utility $su(\alpha, i)$ of each item $i \in Secondary(\alpha)$ by scanning D, using a utility-bin array;
$Primary(\alpha) = \{i | i \in Secondary(\alpha) \wedge su(\alpha, i) \geq minutil\}$;
`Search` (α, D, $Primary(\alpha)$, $Secondary(\alpha)$, $minutil$);

Algorithm 2: The *Search* procedure

input : α: an itemset, α-D: the α projected database, $Primary(\alpha)$: the primary items of α, $Secondary(\alpha)$: the secondary items of α, the $minutil$ threshold
output: the set of high-utility itemsets that are extensions of α

foreach *item* $i \in Primary(\alpha)$ **do**
    $\beta = \alpha \cup \{i\}$;
    Scan α-D to calculate $u(\beta)$ and create β-D; `// with transaction merging`
    if *β has no backward extension* **then**
        Calculate $sup(\beta, z)$, $su(\beta, z)$ and $lu(\beta, z)$ for all item $z \in Secondary(\alpha)$ by scanning β-D once, using three utility-bin arrays;
        if $sup(\beta) = sup(\alpha \cup \{z\}) \forall z \succ i \wedge z \in E(\alpha)$ **then**
            Output $\beta \cup \bigcup_{z \succ i \wedge z \in E(\alpha)} \{z\}$ if it is a HUI; `// closure jumping`
        else
            $Primary(\beta) = \{z \in Secondary(\alpha) | su(\beta, z) \geq minutil\}$;
            $Secondary(\beta) = \{z \in Secondary(\alpha) | lu(\beta, z) \geq minutil\}$;
            `Search` (β, β-D, $Primary(\beta)$, $Secondary(\beta)$, $minutil$);
            if *β has no forward extension and $u(\beta) \geq minutil$* **then** output β;
        end
    end
end

Execution times



EFIM-Closed is up to 71 times faster than **CHUD**

Maximum Memory usage (MB)

Dataset	EFIM-Closed	CHUD
Accidents	895	2,603
BMS	64	707
Chess	65	327
Connect	385	1,504
Foodmart	64	215
Mushroom	71	1,308

EFIM-Closed consumes up to **18 times less memory**

Number of visited nodes

Dataset	EFIM-Closed	CHUD
Accidents	1,341	29,932
BMS	7	27
Chess	348,633	7,759,252
Connect	19,336	218,059
Foodmart	6,680	6,680
Mushroom	8,017	17,621

- **EFIM-Closed** is generally more effective at pruning the search space.
- On Chess, 22 times less nodes are visited by **EFIM-Closed**

Conclusion

- Contribution:
  - ➢ New algorithm for mining **closed high utility itemsets** named **EFIM-Closed**
- ➢ Experimental results:
  - – **EFIM-Closed** is up to **71 times faster** and consumes up to **18 times less memory** than the state-of-the-art **CHUD** algorithm
- Source code and datasets available as part of the **SPMF data mining library (**GPL 3).

Open source Java data mining software, 120 algorithms
http://www.phillippe-fournier-viger.com/spmf/