The EFIM algoritm

Philippe Fournier-Viger

http://www.philippe-Fournier-viger.com

Zida, S., Fournier-Viger, P., Lin, J. C.-W., Wu, C.-W., Tseng, V.-S. (2017).
EFIM: A Fast and Memory Efficient Algorithm for High-Utility Itemset Mining.
Knowledge and Information Systems (KAIS), Springer, 51(2), 595-625

Source code and datasets available in the SPMF library

Introduction

- **High utility itemset mining**
  - a data mining task
  - the goal is to find sets of values that appear together and have a high importance
  - (e.g. sets of products that yield a high profit)
- The **EFIM algorithm (2016)**
  - One of the fastest
  - Perhaps the most memory-efficient

PROBLEM DEFINITION

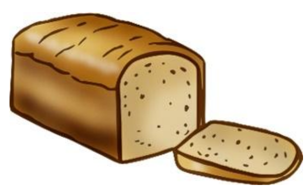
Definition: Items

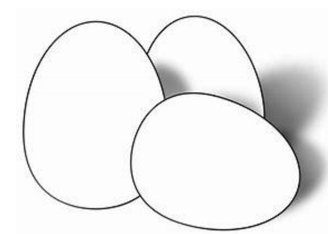
Let there be a **set** of **items** (symbols) called I.

Example: $I = \{a, b, c, d, e, f, g\}$

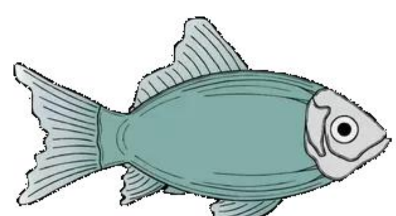
a = apple 

d = dattes 

b = bread 

e = eggs 

c = cake 

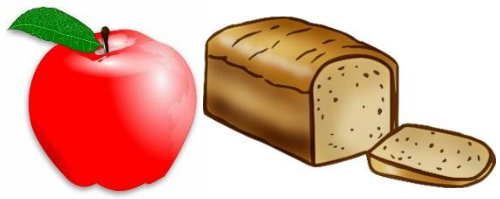
f = fish 

g = grapes 

Definition: Itemset

An **itemset** X is a finite set of items such that X $\subseteq$ I.

Example: $\{a, b\}$ means buying apple and bread together.



It is an itemset of size 2 (it contains 2 items)

Definition: Transaction database

A **transaction database** is a set of transactions $D = \{T_1, T_2, \ldots, T_n\}$

A **transaction T** is a set of items purchased by a customer ($T \subseteq I$).

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

Definition: Transaction ID (TID)

In a **transaction database**, each transaction has an ID.

$$D = \{T_1, T_2, \ldots, T_n\}$$

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

7

The TID of transaction T_1 is 1.

The TID of transaction T_2 is 2.

...

Definition: External Utility (unit profit)

- Each item $i \in I$ is associated with a positive number $p(i)$, called its **external utility, or unit profit**.
- It represents the relative importance of the item to the user.

Example:

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

Selling 1 apple yields a 5$ profit

Definition: External Utility (unit profit)

- Each item $i \in I$ is associated with a positive number $p(i)$, called its **external utility, or unit profit**.
- It represents the relative importance of the item to the user.

Example:

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

Selling 1 bread yields a 2$ profit

Definition: Internal Utility (quantity)

Every item i appearing in a transaction T_c is associated with a positive number $q(i, T_c)$, called its **internal utility**, or **quantity**.

Example:

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

Definition: Internal Utility (quantity)

Every item i appearing in a transaction T_c is associated with a positive number $q(i, T_c)$, called its **internal utility**, or **quantity**.

Example:

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

$q(a, T_1) = 1$ one apple was bought in transaction T_1

Definition: Internal Utility (quantity)

Every item i appearing in a transaction T_c is associated with a positive number $q(i, T_c)$, called its **internal utility**, or **quantity**.

Example:

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

$q(b, T_1) = 2$ two apples were bought in transaction T_2

Definition: Utility of an item in a transaction

- The **utility of an item** i **in a transaction** T_c is defined as $u(i, Tc) = p(i) \times q(i, Tc)$.
- It represents the profit obtained by the sale of item i in that transaction

Example:

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

Definition: Utility of an item in a transaction

- The **utility of an item** i **in a transaction** T_c is defined as $u(i, Tc) = p(i) \times q(i, Tc)$.
- It represents the profit obtained by the sale of item i in that transaction

Example:

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

The utility of item a in transaction T_1 is $u(a, T_1) = 1 \times 5\$ = 5\ \$$

Definition: Utility of an item in a transaction

- The **utility of an item** i **in a transaction** T_c is defined as $u(i, Tc) = p(i) \times q(i, Tc)$.
- It represents the profit obtained by the sale of item i in that transaction

Example:

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

The utility of item a in transaction T_2 is $u(a, T_2) = 2 \times 5\$ = 10\ \$$

Definition:

Utility of an itemset in a transaction

The **utility of an itemset** X in a **transaction** T_c is:

$$u(X, T_c) = \sum_{i \in X} u(i, T_c)$$

Example:

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

Definition:
Utility of an itemset in a transaction

The **utility of an itemset** X in a **transaction** T_c is:

$$u(X, T_c) = \sum_{i \in X} u(i, T_c)$$

Example:

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

The utility of itemset $\{a, c\}$ in transaction T_1 is

$$u(\{a,c\}, T_1) = u(a, T_1) = 1 \times 5\$ = 5\$$$

$$+\ u(c, T_1) = 1 \times 1\$ = 1\$$$

$$= 6\$$$

Definition:

Utility of an itemset in a database

The **utility of an itemset** X in a database is defined as

$$u(X) = \sum_{T_c \in g(X)} u(X, T_c)$$

where g(x) is the set of transactions containing X

Example:

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

The utility of the itemset $\{a, c\}$ is

Definition:

Utility of an itemset in a database

The **utility of an itemset** X in a database is defined as

$$u(X) = \sum_{T_c \in g(X)} u(X, T_c)$$

where g(x) is the set of transactions containing X

Example:

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

The utility of the itemset $\{a, c\}$ is

$$\begin{aligned} u(\{a,c\}) &= u(\{a,c\},T_1) + u(\{a,c\},T_2) + u(\{a,c\},T_3) \\ &= u(a,T_1) + u(c,T_1) + u(a,T_2) + u(c,T_2) + u(a,T_3) + u(c,T_3) \\ &= 5+1+10+6+5+1 \\ &= 28. \end{aligned}$$

Definition:
High utility itemset

A**n itemset** X is a **high utility itemset** if its utility is no less than a threshold $minutil$, chosen by the user. (i.e., $\mathrm{u(X)} \geq minutil$)

Example:

If *minutil = 20*
then $\{\mathrm{a, c}\}$ is a high utility itemset
because $\mathrm{u(\{a, c\})} = 28 \geq minutil$

High utility itemset mining

Input

a transaction database

TID	Transaction
T_1	$(a,1)(c,1)(d,1)$
T_2	$(a,2)(c,6)(e,2)(g,5)$
T_3	$(a,1)(b,2)(c,1)(d,6)(e,1)(f,5)$
T_4	$(b,4)(c,3)(d,3)(e,1)$
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	$(a,1)(c,1)(d,1)$
T_2	$(a,2)(c,6)(e,2)(g,5)$
T_3	$(a,1)(b,2)(c,1)(d,6)(e,1)(f,5)$
T_4	$(b,4)(c,3)(d,3)(e,1)$
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.
For example, if *minutil = 30$*,
the high-utility itemsets are:

Itemset	Utility
$\{b,d\}$	30
$\{a,c,e\}$	31
$\{b,c,d\}$	34
$\{b,c,e\}$	31
$\{b,d,e\}$	36
$\{b,c,d,e\}$	40
$\{a,b,c,d,e,f\}$	30

A challenging task!

- The search space is huge:
  {a}, {b}, {c} … {a,b}, {a,c}, {a,d}, … {b,c}, {b,d}…
  {a,b,c}… {a,b,c,de}
- The **utility** is not anti-monotonic or monotonic.
  - The utility of an itemset may be equal, more or less than the utility of its supersets.
  - Utility of {b,d} = 30$
  - Utility of {b,c,d} = 34$
  - Utility of {b,c,d,e} = 40$

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), EFIM (2016), ULB-Miner (2017), REX (2020), HAMM (2023)**
- **Key idea**: calculate an upper-bound (e.g. TWU) on the utility of itemsets that is anti-monotonic, to be able to reduce the search space.

THE EFIM ALGORITHM

Zida, S., Fournier-Viger, P., Lin, J. C.-W., Wu, C.-W., Tseng, V.-S. (2017).
EFIM: A Fast and Memory Efficient Algorithm for High-Utility Itemset Mining.
Knowledge and Information Systems (KAIS), Springer, 51(2), 595-625

The EFIM algorithm

- **EFIM**
  - performs a depth-first search
  - read the database to calculate the utility of each itemset and upper-bounds
  - uses a «pattern-growth approach» to only consider patterns that exist in the database
- Three main ideas to be more efficient:
  - **HDP**: High-utility Database Projection
  - **HTM:** High-utility Transaction Merging
  - **Utility-Bin Array** to calculate utility and upper-bounds

EXPLORING THE SEARCH SPACE

The search space

How to search for itemsets?

We will assume that there exists a **processing order $\succ$ of items**.

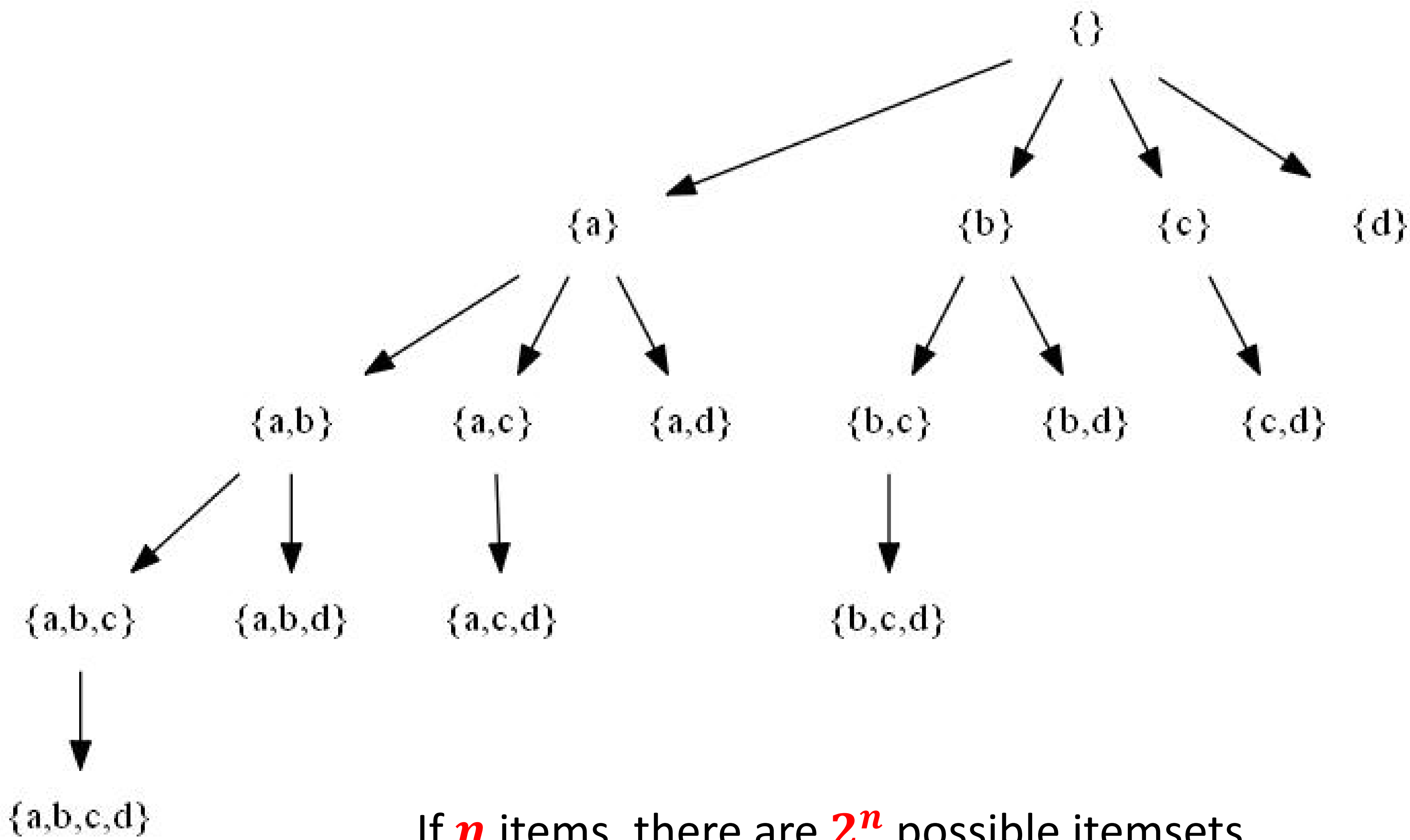
For example, the lexicographical order:

$e \succ d \succ c \succ b \succ a$

Note: in practice, EFIM uses the TWU ascending order. More on that later...

The search space

The **search space of all itemsets** can be visualized as a *set-enumeration tree (Rymon et al., 1992)*.



If n items, there are 2^n possible itemsets.

Exploring the search space

- EFIM starts to search from the itemset $\alpha = \emptyset$.
- EFIM recursively **extends** the current itemset α by appending an item i to obtain a larger itemset $Z = \alpha \cup \{i\}$.

Example

$\emptyset$ can be extended with a to obtain {a}.

Then, {a} can be extended with b to obtain {a, b}.

Then, {a, b} can be extended with c to obtain {a, b, c}.

....

Exploring the search space

To avoid exploring the same itemset twice, EFIM extends an itemset X with an item only if it follows the $\prec$ order.

Examples

$\{a\}$ can be extended with b

$\{a, b\}$ can be extended with c

$\{b, c\}$ can be extended with d

$\{b\}$ **cannot** be extended with a because $a \prec b$

$\{c, d\}$ **cannot** be extended with b $because\ b \prec c$

More formally...

The set of items that can be used to extend an itemset X is defined as:

$$E(\alpha) = \{z | z \in I \land z \succ x, \forall x \in \alpha\}$$

Example:

$$E(\{\}) = \{a, b, c, d, e, f, g\}$$

$$E(\{a\}) = \{b, c, d, e, f, g\}$$

$$E(\{a, c\}) = \{d, e, f, g\}$$

Definitions: Extensions

- An itemset obtained by adding some item(s) to an itemset X while respecting the $\prec$ order is called an **extension** of X.

$\{a, b, c\}$ is an extension of $\{a\}$

- A **single item extension** is an itemset obtained by extending an itemset with only one item.

$\{a, b\}$ is a single extension of $\{a\}$

$\{a, b, c\}$ is not a single extension of $\{a\}$

HIGH-UTILITY DATABASE PROJECTION (HDP)

Reading the database

- To find high utility itemsets, **EFIM reads** the **database** several times to calculate the utility of itemsets in the search space.
- But reading the database can be costly in time!
- **How to reduce the cost?**
  - EFIM assumes that items in transactions are sorted according to the order $\prec$
  - EFIM performs an operation called **high utility database projection** to reduce the size of the database.

Definition: Projected database

The **projection of a database by an itemset** α is obtained by keeping only transactions containing α and removing all items not in $E(\alpha)$.

Example 1

A database D

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

Definition: Projected database

The **projection of a database by an itemset** α is obtained by keeping only transactions containing α and removing all items not in E(α).

Example 1

A database D

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

If α = {a},

the projection {a}$-D$ is:

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

Definition: Projected database

The **projection of a database by an itemset** α is obtained by keeping only transactions containing α and removing all items not in E(α).

Example 1

If α = {a},
the projection {a}$-D$ is:

TID	Transaction
T_1	$(c,1)(d,1)$
T_2	$(c,6)(e,2)(g,5)$
T_3	$(b,2)(c,1)(d,6)(e,1)(f,5)$

Definition: Projected database

The **projection of a database by an itemset** α is obtained by keeping only transactions containing α and removing all items not in $E(\alpha)$.

Example 1

If α = {a},
the projection {a}$-D$ is:

TID	Transaction
T_1	$(c,1)(d,1)$
T_2	$(c,6)(e,2)(g,5)$
T_3	$(b,2)(c,1)(d,6)(e,1)(f,5)$

- **Why doing this**?
  Because, in the projected database of {a}, EFIM has all the information required to search for extensions of {a}.
- And the projected database is smaller than D, so it can be read more quickly.

Definition: Projected database

The **projection of a database by an itemset** α is obtained by keeping only transactions containing α and removing all items not in $E(\alpha)$.

Example 2

The database {a}-D

TID	Transaction
T_1	$(c,1)(d,1)$
T_2	$(c,6)(e,2)(g,5)$
T_3	$(b,2)(c,1)(d,6)(e,1)(f,5)$

Definition: Projected database

The **projection of a database by an itemset** α is obtained by keeping only transactions containing α and removing all items not in E(α).

Example 2

The database {a}-D

TID	Transaction
T_1	$(c,1)(d,1)$
T_2	$(c,6)(e,2)(g,5)$
T_3	$(b,2)(c,1)(d,6)(e,1)(f,5)$

The database {a,d}-D

TID	Transaction
T_1	$\cancel{(c,1)(d,1)}$
T_2	$\cancel{(c,6)(e,2)(g,5)}$
T_3	$\cancel{(b,2)(c,1)(d,6)}(e,1)(f,5)$

Definition: Projected database

The **projection of a database by an itemset** α is obtained by keeping only transactions containing α and removing all items not in $E(\alpha)$.

Example 2

The database {a}-D

TID	Transaction
T_1	$(c,1)(d,1)$
T_2	$(c,6)(e,2)(g,5)$
T_3	$(b,2)(c,1)(d,6)(e,1)(f,5)$

The database {a,d}-D

TID	Transaction
T_3	$(e,1)(f,5)$

Definition: Projected database

The **projection of a database by an itemset** α is obtained by keeping only transactions containing α and removing all items not in $E(\alpha)$.

Example 2

The database {a,d}-D

TID	Transaction
T_1	
T_2	
T_3	$(e,1)(f,5)$

Benefits of database projections

- As EFIM explores larger and larger itemsets, the **database** becomes **smaller** and **smaller**.
- This reduce the time required to read the database.
- But making multiple copies of the database in memory uses too much memory?

Solution -->

Pseudo-projection

Instead of making a copy of a database:

A database D

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

The projection $\{a\}\text{–}D$ is:

TID	Transaction
T_1	$(c,1)(d,1)$
T_2	$(c,6)(e,2)(g,5)$
T_3	$(b,2)(c,1)(d,6)(e,1)(f,5)$

Pseudo-projection

Instead of making a copy of a database:

A database D

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

The projection $\{a\}-D$ is:

TID	Transaction
T_1	$(c, 1)(d, 1)$
T_2	$(c, 6)(e, 2)(g, 5)$
T_3	$(b, 2)(c, 1)(d, 6)(e, 1)(f, 5)$

EFIM uses pointers on the original database:

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

Pseudo-projection

Instead of making a copy of a database:

A database D

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

The projection $\{a\}\text{–}D$ is:

TID	Transaction
T_1	$(c,1)(d,1)$
T_2	$(c,6)(e,2)(g,5)$
T_3	$(b,2)(c,1)(d,6)(e,1)(f,5)$

EFIM uses pointers on the original database:

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

Need to store at most 1 pointer per transaction

HIGH-UTILITY TRANSACTION MERGING (HTM)

To further reduce memory

- EFIM introduces a technique named **High-utilty Transaction Merging (HTM)**.
- **Two observations**:
  - (*Projected*) databases often contain identical transactions.
  - We can identify these transaction and merge them to reduce memory and also the time to scan the database.

Transaction merging

Original database

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

Projected database $\{c\}-D$

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

Transaction merging

Original database

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

Projected database $\{\mathbf{c}\} - \mathbf{D}$

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

$\Rightarrow$

Merged database of $\{\mathbf{c}\} - \mathbf{D}$

TID	Transaction
T_1	$(d,1)$
$T_{2,3}$	$\boxed{(e,3)(g,\mathbf{7})}$
T_3	$(d,6)(e,1)(f,5)$
T_4	$(d,3)(e,1)$

How to implement merging?

- **Challenge:**
  - How to find identical transactions efficiently in a database to merge them?
- **Naive solution (exponential time):**
  - Compare each transaction with all other transactions.
- **EFIM's solution:**
  - Sort transactions by the order $\prec$ but backward (*see EFIM's paper*)
  - Then, EFIM can find identical transactions in **linear time**!

HOW EFIM REDUCES THE SEARCH SPACE?

Reducing the search space

EFIM reduces the search space using two upper bounds on the utility that are anti-monotonic:

- the **local utility (lu)**
- the **sub-tree utility (su)**

I will explain them with examples: -->

Definition: remaining utility

Definition (Remaining utility). *The* remaining utility *of* an itemset X *in a transaction* T_c *is defined as* $re(X, T_c) = \sum_{i \in T_c \wedge i \succ x \forall x \in X} u(i, T_c)$

Example $\alpha = \{a\}$

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

Definition: remaining utility

Definition (Remaining utility). *The remaining utility of* an itemset X *in a transaction* T_c *is defined as* $re(X, T_c) = \sum_{i \in T_c \wedge i \succ x \forall x \in X} u(i, T_c)$

Example $\alpha = \{a\}$

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

We can obtain $\mathbf{re}(\boldsymbol{\alpha}) = \ldots. = \mathbf{3} + \mathbf{17} + \mathbf{25} = \mathbf{45}$

Definition: remaining utility

Definition (Remaining utility). *The* remaining utility *of* an itemset X *in a transaction* T_c *is defined as* $re(X, T_c) = \sum_{i \in T_c \wedge i \succ x \forall x \in X} u(i, T_c)$

Example $\alpha = \{a, b\}$

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

Definition: remaining utility

Definition (Remaining utility). *The remaining utility of* an itemset X *in a transaction* T_c *is defined as* $re(X, T_c) = \sum_{i \in T_c \wedge i \succ x \forall x \in X} u(i, T_c)$

Example $\boldsymbol{\alpha} = \{a, b\}$

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

We can obtain $\mathbf{re}(\boldsymbol{\alpha}) = \ldots. = \mathbf{21}$

Definition: local utility

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

Example

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

$\boldsymbol{\alpha} = \{\}$ $\qquad$ $\mathbf{z} = a$

Definition: local utility

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

Example

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

$\boldsymbol{\alpha} = \{\}$ $\quad\quad$ $\mathbf{z} = a$

We can obtain $\mathbf{lu(\alpha, z)} = \ldots = \mathbf{8 + 27 + 30 = 65}$

Definition: local utility

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

Example

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

$\boldsymbol{\alpha} = \{\}$ $\mathbf{z} = a$

We can obtain $\mathbf{lu}(\boldsymbol{\alpha}, \mathbf{z}) = \ldots. = \mathbf{8} + \mathbf{27} + \mathbf{30} = \mathbf{65}$

==> Any itemset containing «a» cannot have a utility greater than 65!

Definition: local utility

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

Example

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

$\alpha = \{\mathbf{d}\}$ $\quad$ $\mathbf{z} = e$

Definition: local utility

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

Example

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

$\alpha = \{d\}$ $\quad\quad$ $z = e$

We can obtain $\mathbf{lu}(\alpha, \mathbf{z}) = \ldots. = \mathbf{20} + \mathbf{9} = \mathbf{29}$

==> Any itemset extending {d} and containing «e» cannot have a utility greater than 29!

Definition: local utility

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

Example

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

$\alpha = \{d\}$ $\qquad$ $z = f$

Definition: local utility

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

Example

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

$\boldsymbol{\alpha} = \{\mathbf{d}\}$ $\quad\quad$ $\mathbf{z} = f$

We can obtain $\mathbf{lu(\alpha, z)} = \ldots\ldots = \mathbf{20}$

==> Any itemset extending {d} and containing «f» cannot have a utility greater than 20!

Definition: sub-tree utility

Definition (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*
$su(\alpha, z) = \sum_{T \in g(\alpha \cup \{z\})} [u(\alpha, T) + u(z, T) + \sum_{i \in T \wedge i \in E(\alpha \cup \{z\}} u(i, T)]$.

Example

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

$\alpha = \{d\}$ $\qquad$ $z = f$

Definition: sub-tree utility

Definition (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*
$su(\alpha, z) = \sum_{T \in g(\alpha \cup \{z\})} [u(\alpha, T) + u(z, T) + \sum_{i \in T \wedge i \in E(\alpha \cup \{z\}} u(i, T)]$.

Example

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

$\boldsymbol{\alpha} = \{\mathbf{d}\}$ $\quad\quad$ $\mathbf{z} = f$

We can obtain $\mathbf{su}(\boldsymbol{\alpha}, \mathbf{z}) = \ldots. = \mathbf{17}$

==> Any itemset extending {d,f} cannot have a utility greater than 17!

Definition: sub-tree utility

Definition (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*
$su(\alpha, z) = \sum_{T \in g(\alpha \cup \{z\})} [u(\alpha, T) + u(z, T) + \sum_{i \in T \wedge i \in E(\alpha \cup \{z\}} u(i, T)]$.

Example

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

$\boldsymbol{\alpha} = \{\mathbf{a}, \mathbf{c}\}$ $\mathbf{z} = e$

Definition: sub-tree utility

Definition (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*
$su(\alpha, z) = \sum_{T \in g(\alpha \cup \{z\})} [u(\alpha, T) + u(z, T) + \sum_{i \in T \wedge i \in E(\alpha \cup \{z\}} u(i, T)]$.

Example

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

$\boldsymbol{\alpha} = \{\mathbf{a}, \mathbf{c}\}$ $\qquad$ $\mathbf{z} = e$

We can obtain $\mathbf{su}(\boldsymbol{\alpha}, \mathbf{z}) = \mathbf{27} + \mathbf{14} = \mathbf{41}$

==> Any itemset extending {a,c,e} cannot have a utility greater than 41!

Primary and secondary items

For an itemset α:

- **The primary items are:**

$$Primary(\alpha) = \{z | z \in E(\alpha) \wedge su(\alpha, z) \geq minutil\}$$

- **The secondary items are:**

$$Secondary(\alpha) = \{z | z \in E(\alpha) \wedge lu(\alpha, z) \geq minutil\}$$

Note:

$Because\ lu(\alpha, z) \geq su(\alpha, z),\ Primary(\alpha) \subseteq Secondary(\alpha).$

For reducing the search...

For α = {a}, we have:

Primary(α) = {c, e}

Secondary(α) = {c, d, e}

This means that we should only explore extensions of {a,c} and {a,e}.

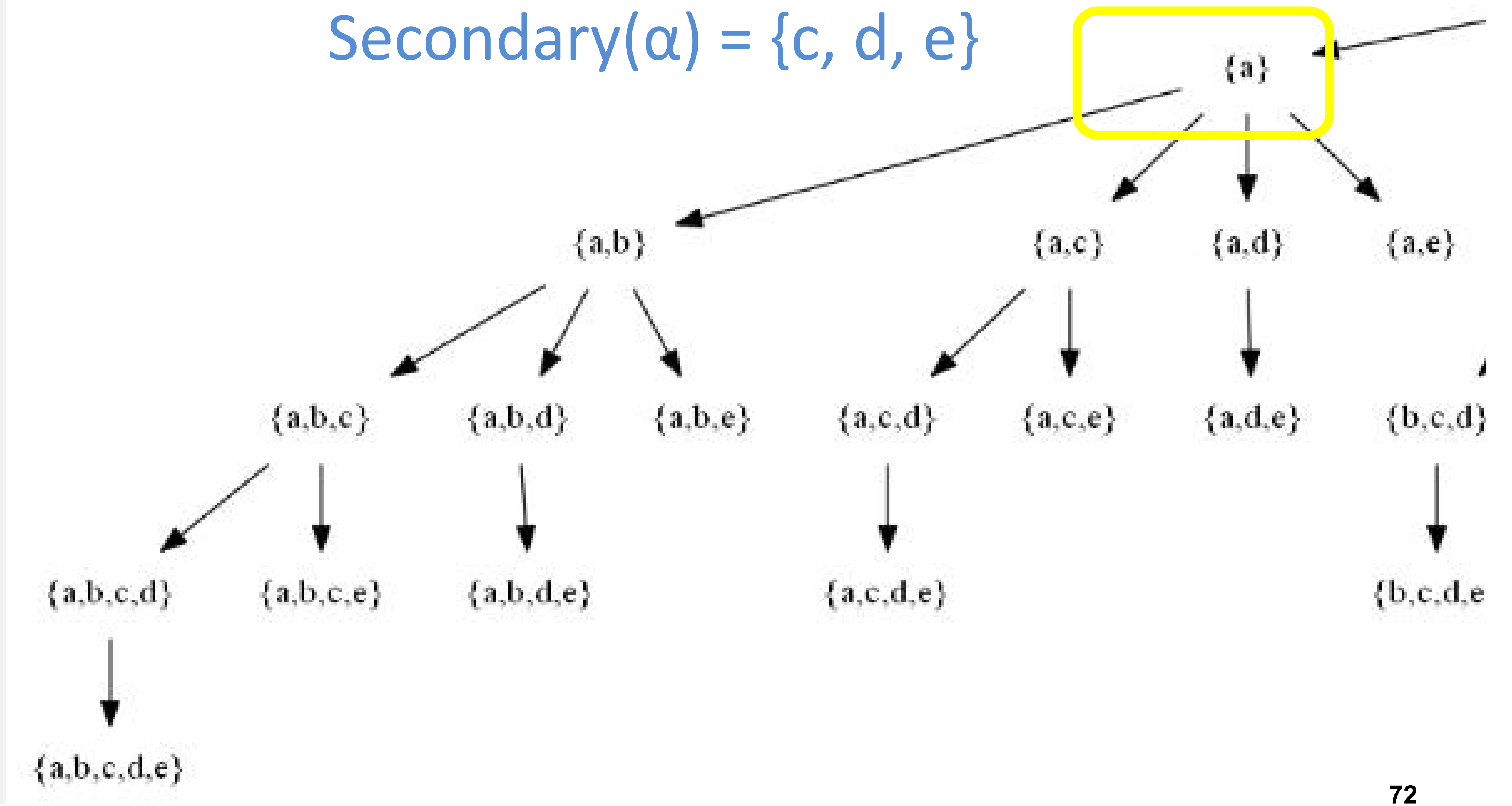
And in to further extend {a,c} and {a,e}, we should only use items c, d or e .

What does it means?

For α = {a}, we have:

Primary(α) = {c, e}

Secondary(α) = {c, d, e}

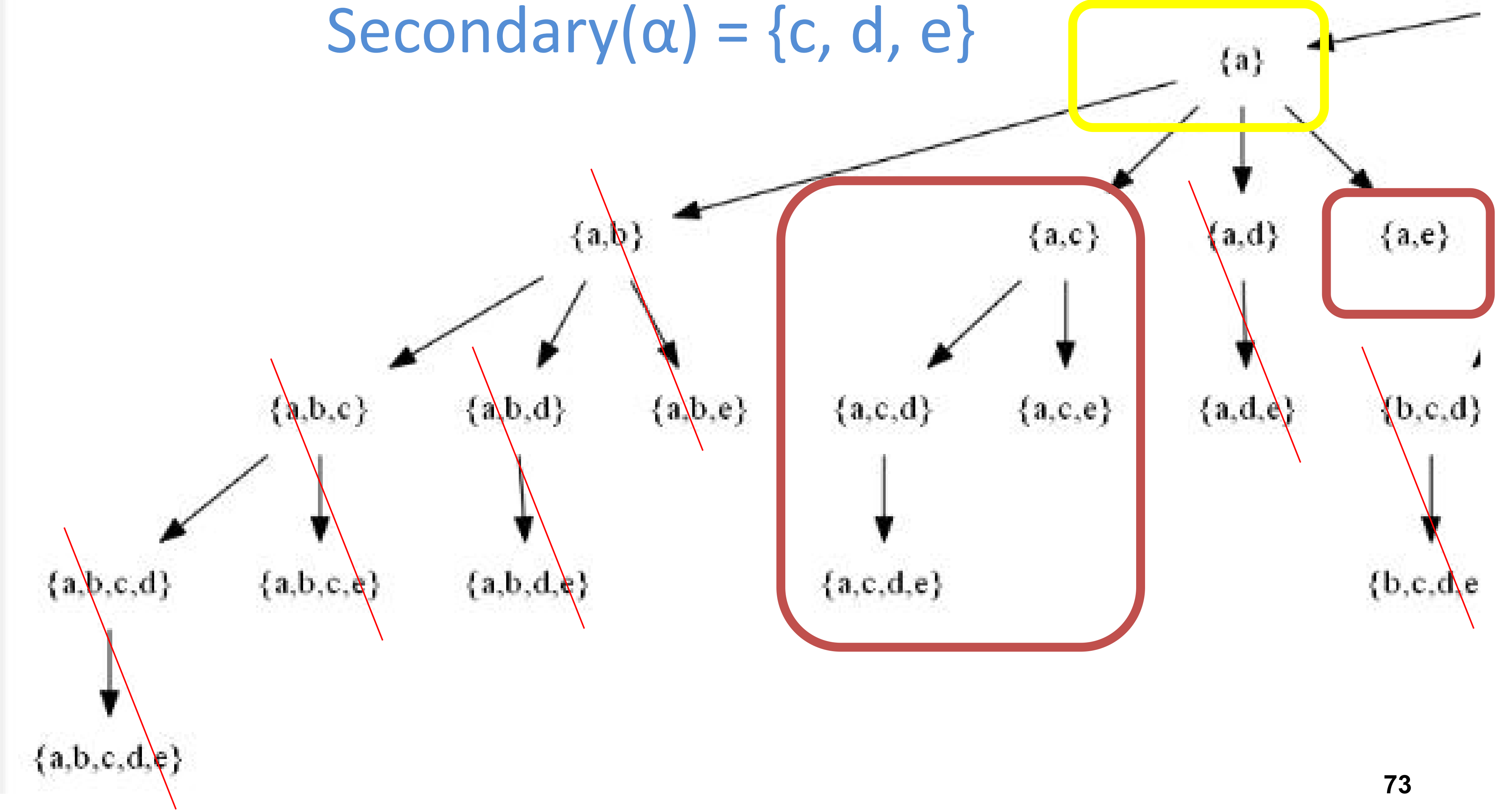


What does it means?

For α = {a}, we have:

Primary(α) = {c, e}

Secondary(α) = {c, d, e}

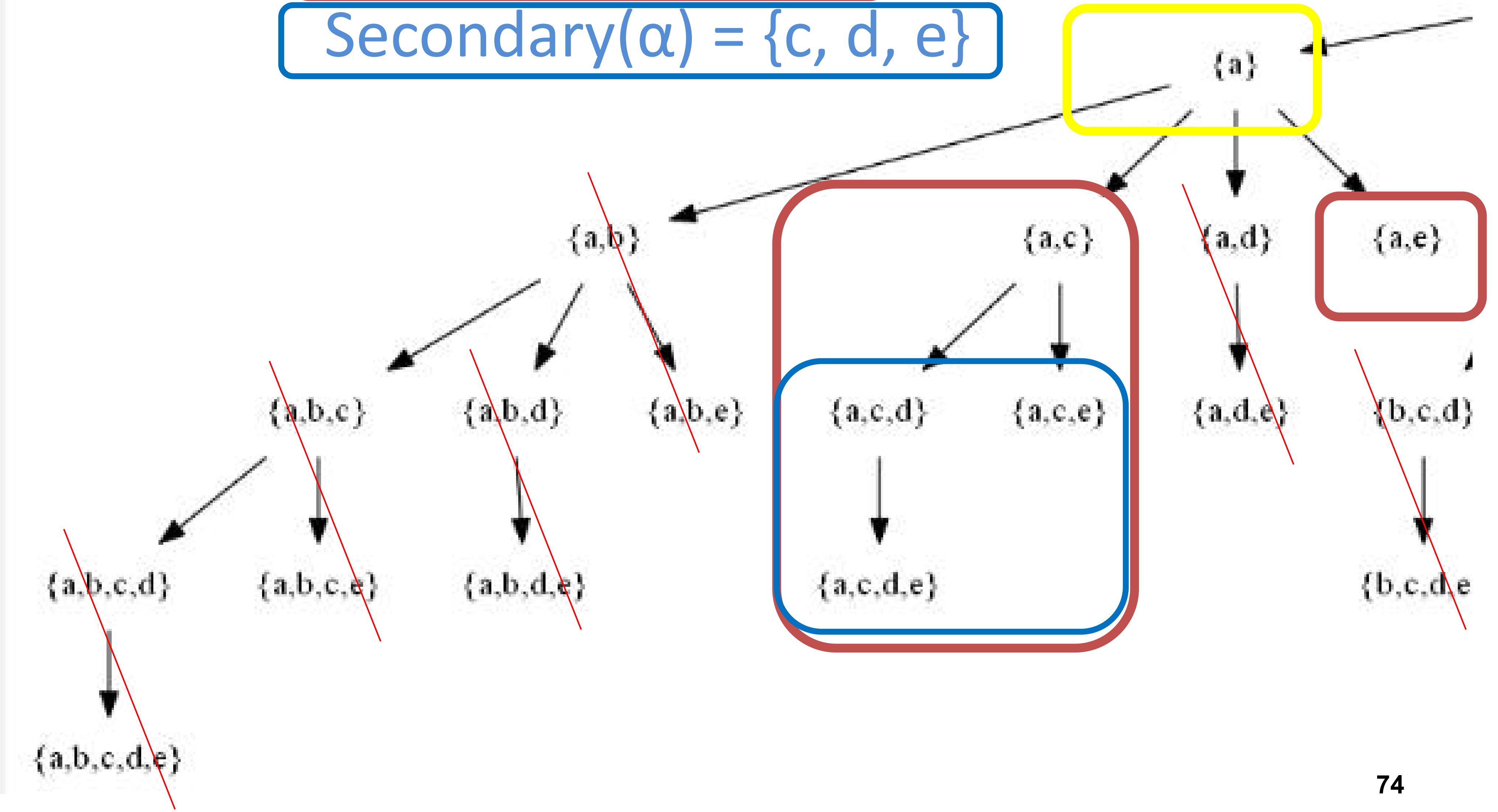


What does it means?

For α = {a}, we have:

Primary(α) = {c, e}

Secondary(α) = {c, d, e}



Other details...

- In the EFIM paper, there is also another upper bound called **revised sub-tree utility**.
- See the paper for more details.

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

EFFICIENTLY CALCULATING THE UTILITY AND UPPER-BOUNDS USING ARRAYS

How to efficiently calculate the utility of an itemset?

- EFIM reads the database
- EFIM uses a special data structure called **Utility Bins** to calculate the utility of all items at the same time.
- It is an **array** with a length equal to the number of different items in the database.
- EFIM also use this structure to calculate the local utility and sub-tree utility of all items.

Example: calculating lu(α, z) for all items

	U[a]	U[b]	U[c]	U[d]	U[e]	U[f]	U[g]
A) Initialization	0	0	0	0	0	0	0

Example: calculating lu(α, z) for all items

	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

Example: calculating $\text{lu}(\alpha, z)$ for all items

	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

Example: calculating $\text{lu}(\alpha, z)$ for all items

	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

Example: calculating lu(α, z) for all items

	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

Example: calculating $\mathrm{lu}(\alpha, z)$ for all items

	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

Efficient?

- EFIM reads the database to compute the utility and upper bounds of all items in **linear time.**
- The arrays requires **linear memory**.

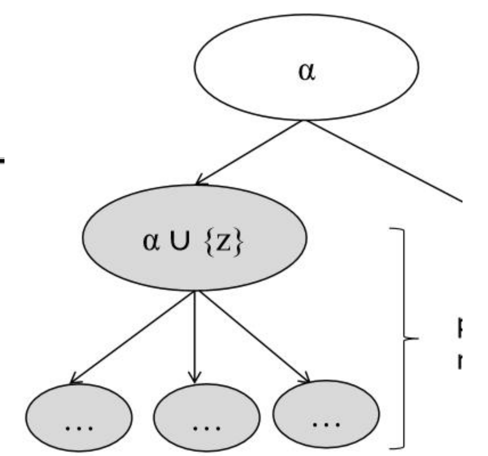
THE ALGORITHM

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)

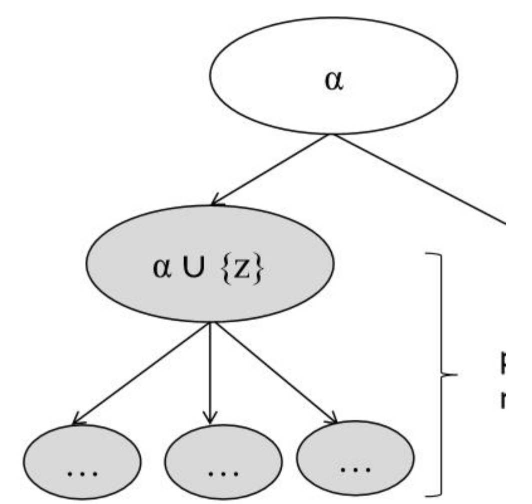
Algorithm 2: The *Search* procedure

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` (β, β-D, $Primary(\beta)$, $Secondary(\beta)$, $minutil$);

end



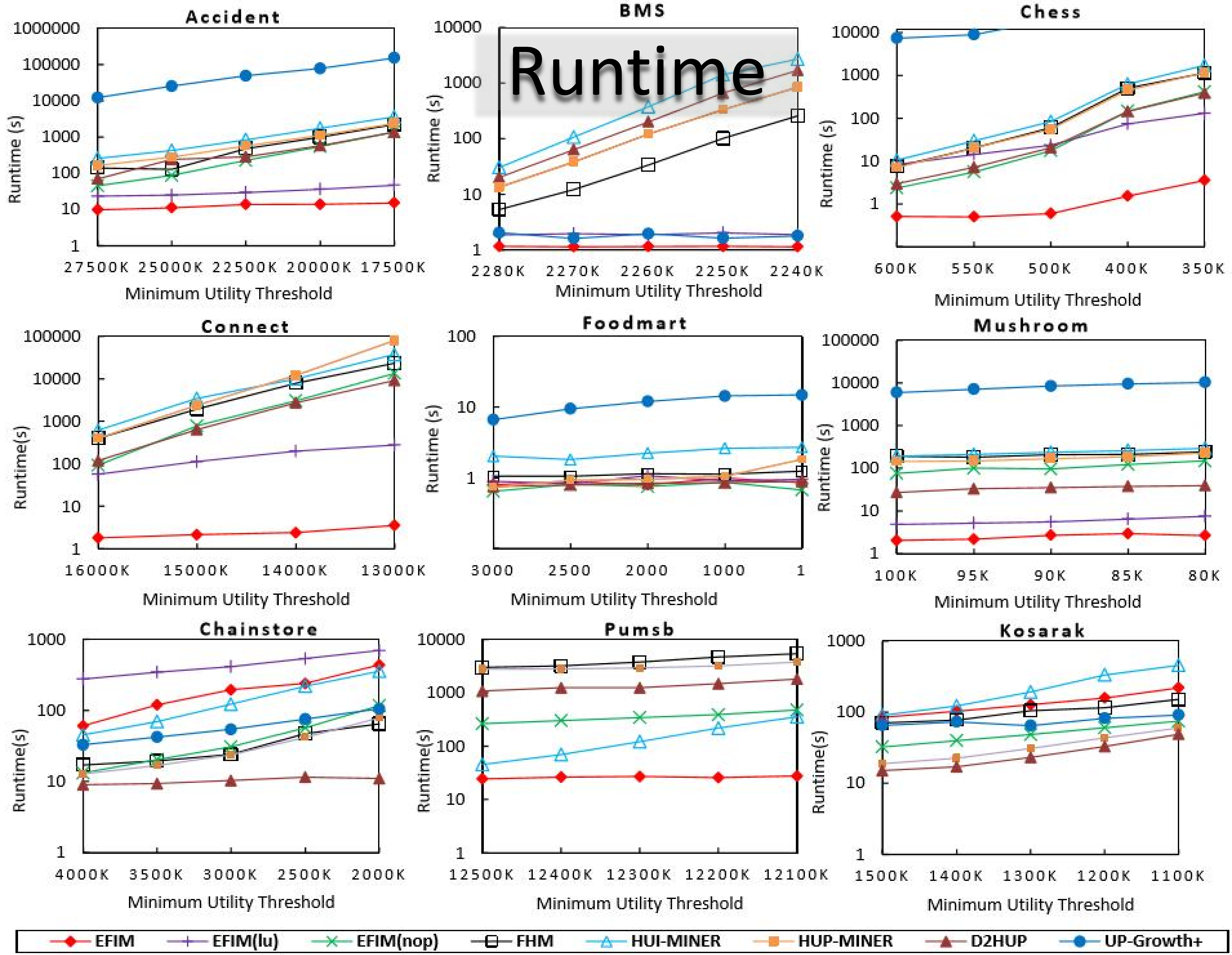
Experimental Evaluation

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

Runtime


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

Memory usage (MB)

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

Influence of merging on database size

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%

Conclusion

- The **EFIM algorithm** for high utility itemset mining
- Many operations in linear time.
- Outperforms many other algorithms

- Source code and datasets available as part of the **SPMF data mining library (**GPL 3).

SPMF

Open source Java data mining software, 275 algorithms
http://www.phillippe-fournier-viger.com/spmf/

EFIM-CLOSED

Fournier-Viger, P., Zida, S. Lin, C.W., Wu, C.-W., Tseng, V. S. (2016). **EFIM-Closed: Fast and Memory Efficient Discovery of Closed High-Utility Itemsets**. Proc. 12th Intern. Conference on Machine Learning and Data Mining (MLDM 2016). Springer, LNAI, pp. 199-213.

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:

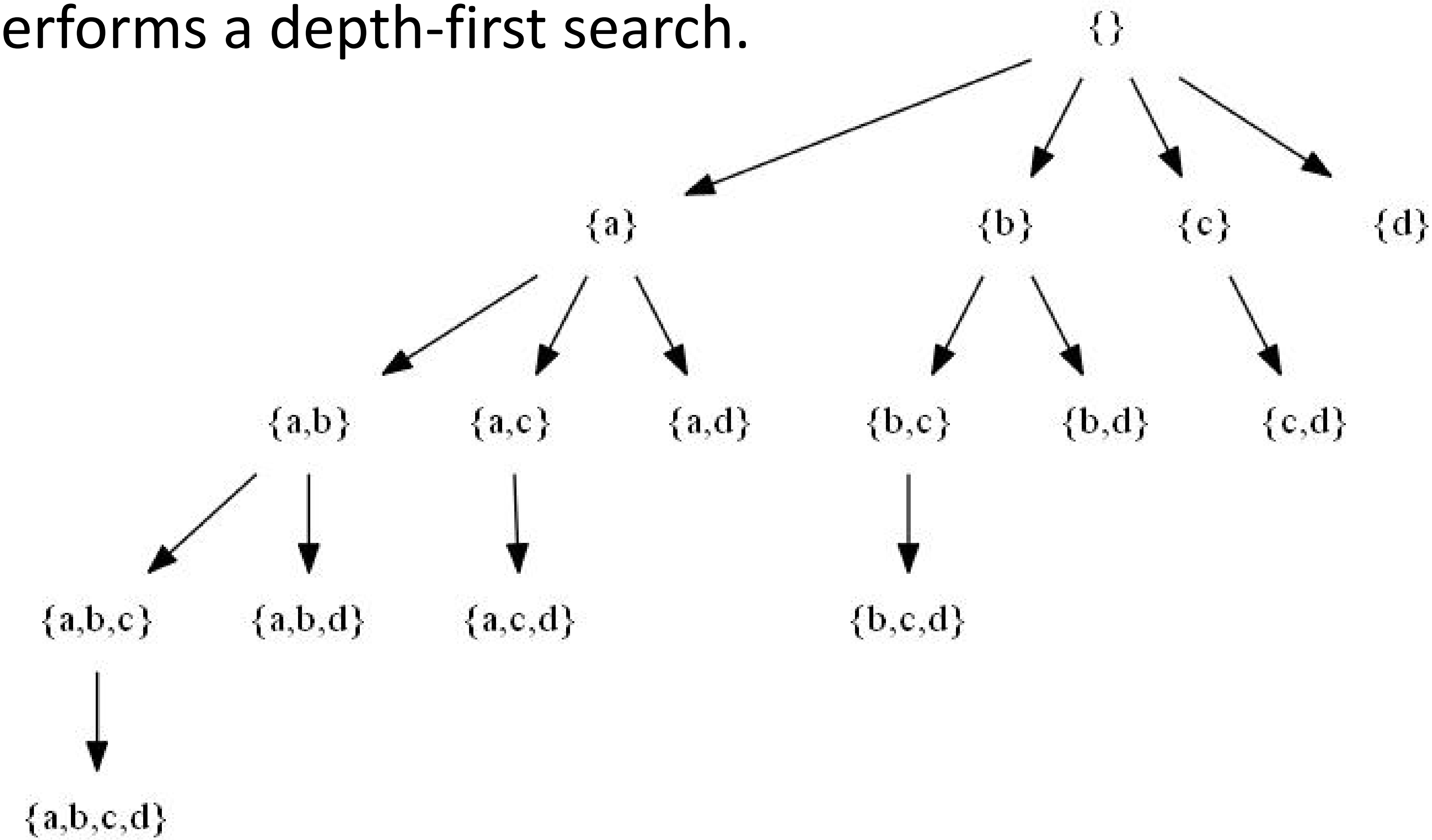
~~{b,d,e} 36$ 2 transactions~~	~~{b,c,d} 34$ 2 transactions~~
{b,c,d,e} 40$ 2 transactions	{b,c,e} 37 $ 3 transactions

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

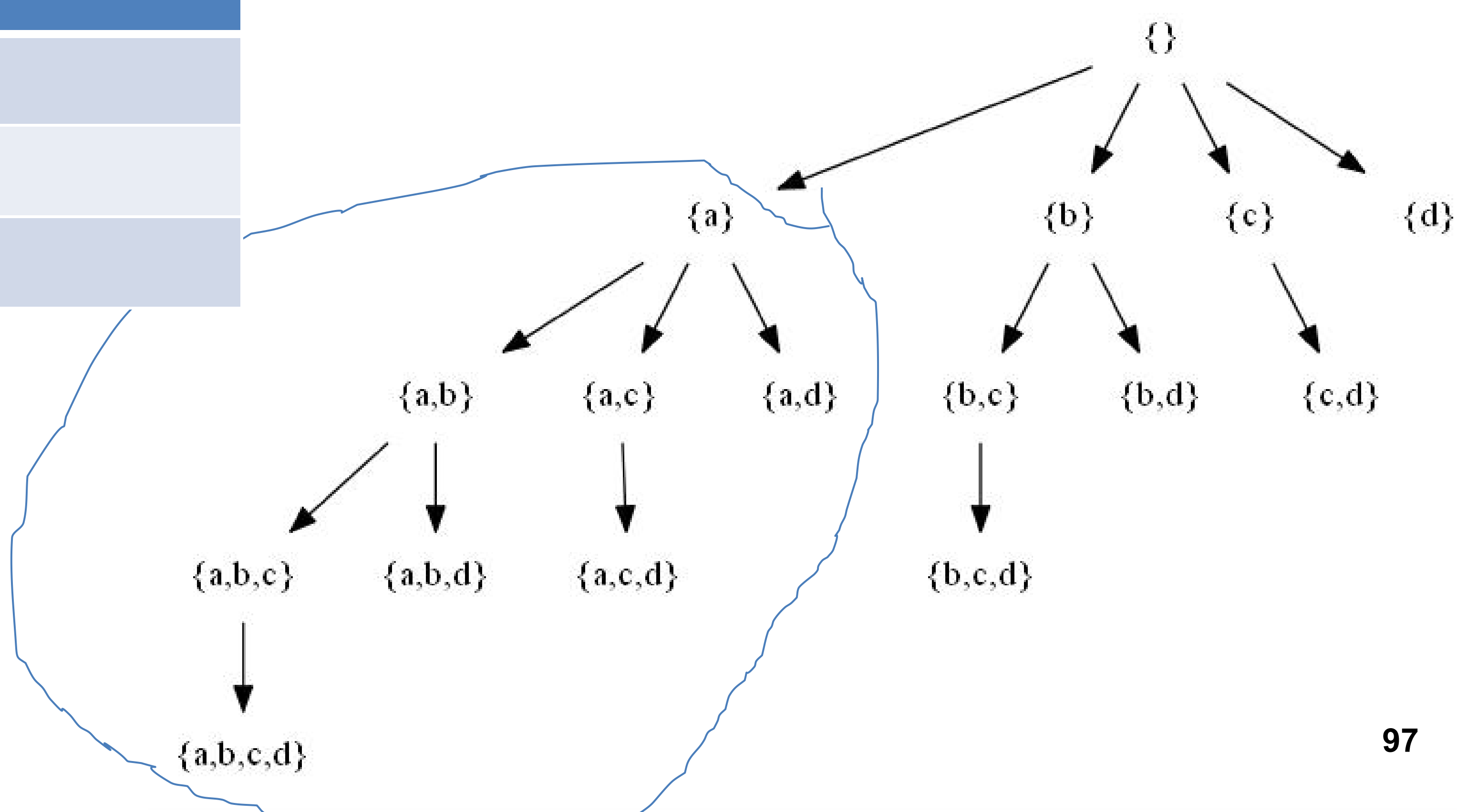
TID	Transaction
T_1	$(a,1),(b,5),(c,1),(d,3),(e,1),(f,5)$
T_2	$(b,4),(c,3),(d,3),(e,1)$
T_3	$(a,1),(c,1),(d,1)$
T_4	$(a,2),(c,6),(e,2),(g,5)$
T_5	$(b,2),(c,2),(e,1),(g,2)$

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

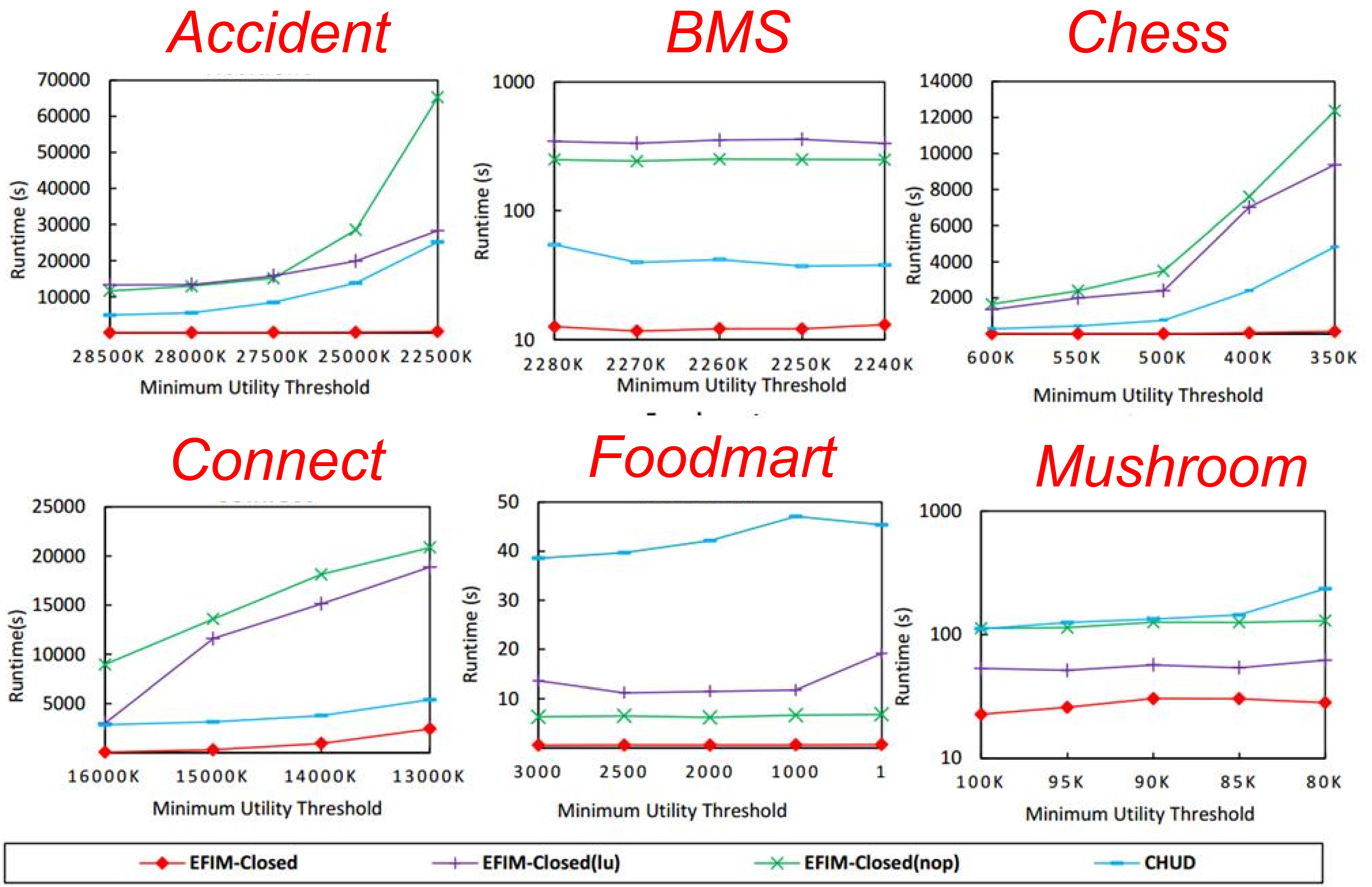
$\alpha = \emptyset$;
Calculate $lu(\alpha, i)$ for all items $i \in I$ by scanning D, using a utility-bin array;
$Secondary(\alpha) = \{i | i \in I \wedge lu(\alpha, i) \geq minutil\}$;
Let $\succ$ be the total order of TWU ascending values on $Secondary(\alpha)$;
Scan D to remove each item $i \notin Secondary(\alpha)$ 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` $(\alpha, D, Primary(\alpha), Secondary(\alpha), minutil)$;

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` $(\beta, \beta\text{-}D, Primary(\beta), Secondary(\beta), minutil)$;
      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, **275 algorithms**
http://www.phillippe-fournier-viger.com/spmf/