Efficient Data Mining for Path Traversal Patterns

Qiang Ye

Nov. 29, 2000

Content

Introduction

- Objective

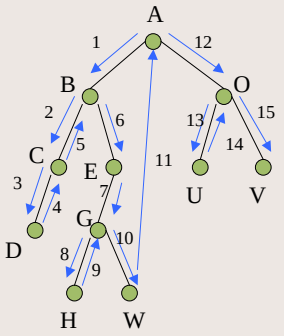
  -To understand user access patterns in a distributed information-provided environment.

- Benefit

  -Improving the system design: e.g., providing efficient access between highly correlated objects.

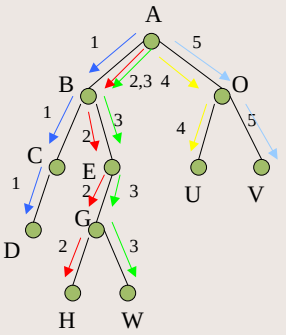
  -Leading to better marketing decisions: e.g., putting advertisements in proper places.

Maximal Forward Reference



Traversal path for a user:

{A,B,C,D,C,B,E,G,H,G,W,A,O,U,O,V}



Maximal Forward References

{ABCD,ABEGH,ABEGW,AOU,AOV}

Problem of finding frequent traversal patterns

Using the concept of maximal froward references

Problem of finding frequent occurring consecutive subsequences among all maximal forward references

- **Large reference sequence**: a reference sequence that appeared in a sufficient number of times.
- **Maximal reference sequence**: a large reference sequence that is not contained in any other maximal reference sequence.

Procedure of mining traversal patterns

- **Step 1**: Determine **maximal forward references** from the original log data.
- **Step 2**: Determine **large reference sequences** from the set of maximal forward references.
- **Step 3**: Determine **maximal reference sequences** from large reference sequences.

Step 1: Set $i = 1$ and string Y to null for initialization, where string Y is used to store the current forward reference path. Also, set the flag $F = 1$ to indicate a forward traversal.

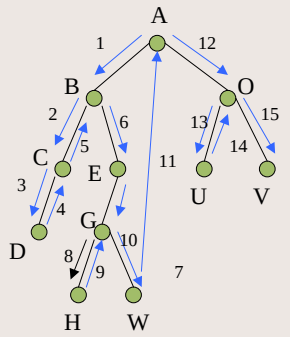
Step 2: Let $A = s_i$ and $B = d_i$.
If A is equal to null then
/* this is the beginning of a new traversal */
begin
Write out the current string Y (if not null) to the database D_F;
Set string $Y = B$;
Go to Step 5.
end

Step 3: If B is equal to some reference (say the jth reference) in string Y then
/* this is a cross-referencing back to a previous reference */
begin
If F is equal to 1 then write out string Y to database D_F;
Discard all the references after the jth one in string Y;
$F = 0$;
Go to Step 5.
end

Step 4: Otherwise, append B to the end of string Y.
/* we are continuing a forward traversal */
If F is equal to 0, set $F = 1$.

Step 5: Set $i = i + 1$. If the sequence is not completed scanned then go to Step 2.

Example of Algorithm MF



Traversal path for a user:

{A,B,C,D,C,B,E,G,H,G,W,A,O,U,O,V}

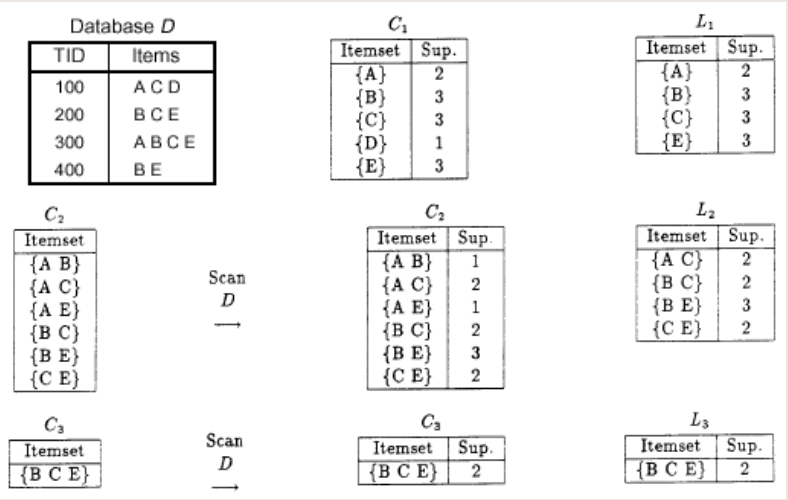
move	string Y	output to D_F
1	AB	–
2	ABC	–
3	ABCD	–
4	ABC	ABCD
5	AB	–
6	ABE	–
7	ABEG	–
8	ABEGH	–
9	ABEG	ABEGH
10	ABEGW	–
11	A	ABEGW
12	AO	–
13	AOU	–
14	AO	AOU
15	AOV	AOV (end)

Characteristics of FS

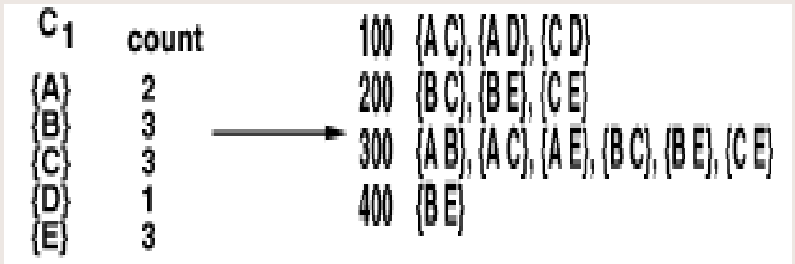
Using key ideas of the DHP (Direct Hashing and Pruning) algorithm, FS has two distinct characteristics:

- Hash Technique: efficient for the generation of candidate reference sequence
- Pruning technique: progressively reduce the transaction database size.

Traditional Generation Method

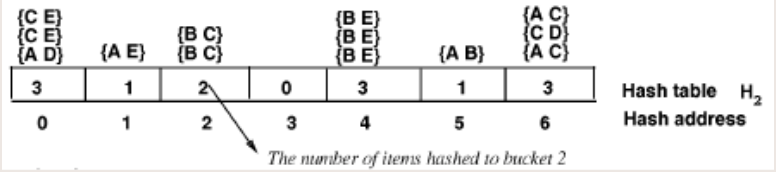


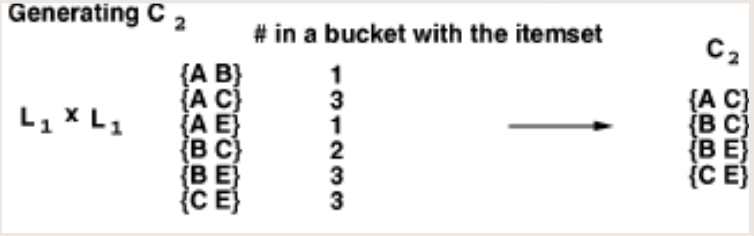
Hash Technique



Hash Technique

Hash Function: h({xy})=((order of x)*10+(order of y)) mod 7





Pruning Technique

- Trimming the size of each transaction:

{A B C D} ⇨ {A B C}

- Reducing the number of the transactions in the database:

TID	Items
100	ACD
200	BCE

⇨

TID	Items
200	BCE

Pruning Technique

- Any subset of a large itemsets must be a large itemsets by itself.
- A transaction can be used to determine the set of large (k+1) itemsets only if it consists of (K+1) large k-itemsets.
- An item in transaction t can be trimmed if it does not appear in at least k of the candidate k-itemsets.

Pruning Technique

C_2	count
{A C}	2
{B C}	2
{B E}	3
{C E}	2

→

L_2
{A C}
{B C}
{B E}
{C E}

TID	Items
100	ACD
200	BCE

{AC} ⟶ Discard

{BC}{BE}{CE} ⟶ Keep {BCE}

For 100:

A	B	C
a[0]=1<2	a[1]=1<2	a[2]=0<2

Algorithm on Selective Scan

- FS generates a **small number** of candidate two-references by using a hashing technique.
- If memory is enough, we can generate C3 from C2*C2 instead of L2*L2, hence saving one round of database scan.
- Using this concept, we can determine all L_ks by as few as **two** scans.

Algorithm on Selective Scan

- Using L2*L2 to generate C3, C3 would be : {{BCE}}

C_2

Itemset
{A B}
{A C}
{A E}
{B C}
{B E}
{C E}

Scan D →

C_2

Itemset	Sup.
{A B}	1
{A C}	2
{A E}	1
{B C}	2
{B E}	3
{C E}	2

L_2

Itemset	Sup.
{A C}	2
{B C}	2
{B E}	3
{C E}	2

C_3

Itemset
{B C E}

Scan D →

C_3

Itemset	Sup.
{B C E}	2

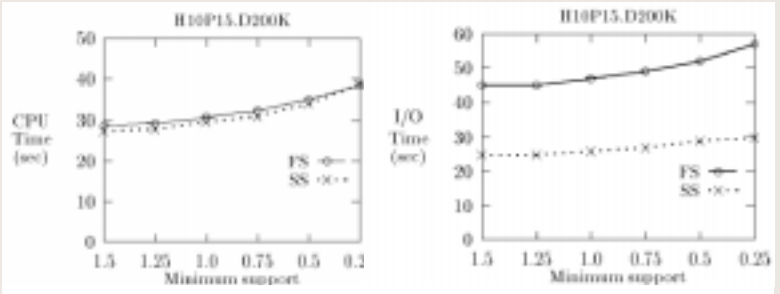
L_3

Itemset	Sup.
{B C E}	2

- Using C2*C2 to generate C3, C3 would be :

{{ABC},{ABE},{ACE},{BCE}}

Performance Evaluation



-SS in general outperforms FS.

-When I/O cost is taken into account, their difference becomes prominent.

Conclusion

- Data Mining is application-dependent.
- How to choose hash functions?
- Can we and need we use hash tree to count the number of candidate reference sequence?
- Are the algorithm assumptions reasonable?