Detailed Description of an Algorithm for Enumeration of Maximal Frequent Sets with Irredundant Dualization

Takeaki Uno, Ken Satoh
National Institute of Informatics
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo, 101-8430, Japan
Email: uno, ksatoh@nii.ac.jp

Abstract

We describe an implementation of an algorithm for enumerating all maximal frequent sets using irredundant dualization, which is an improved version of that of Gunopulos et al. The algorithm of Gunopulos et al. solves many dualization problems, and takes long computation time. We interleaves dualization with the main algorithm, and reduce the computation time for dualization by as long as one dualization. This also reduces the space complexity. Moreover, we accelerate the computation by using sparseness.

1. Introduction

Let E be an item set and $\mathcal{T}$ be a set of transactions defined on E. For an item set $S \subseteq E$, we denote the set of transactions including S by $X(S)$. We define the *frequency* of S by $|X(S)|$. For a given constant α, if an item set S satisfies $|X(S)| \geq \alpha$, then S is said to be *frequent*. A frequent item set included in no other frequent item set is said to be *maximal*. An item set not frequent is called *infrequent*. An infrequent item set including no other infrequent item set is said to be *minimal*.

This paper describes an implementation of an algorithm for enumerating all maximal frequent sets using dualization in detail presented at [SatohUno03]. The algorithm is an improved version of that of Gunopulos et al. [Gunopulos97a, Gunopulos97b]. The algorithm computes maximal frequent sets based on **computing minimal transversals of a hypergraph**, **computing minimal hitting set**, or, in other words, **computing a dualization of a monotone function** [Fredman96]. The algorithm finds all minimal item sets not included in any current obtained maximal frequent set by dualization. If a frequent item set is in those minimal item sets, then the algorithm finds a new maximal frequent set including the frequent item set. In this way, the algorithm avoids checking all frequent item sets. However, this algorithm solves dualization problems many times, hence it is not fast for practical purpose. Moreover, the algorithm uses the dualization algorithm of Fredman and Khachiyan [Fredman96] which is said to be slow in practice.

We improved the algorithm in [SatohUno03] by using incremental dualization algorithms proposed by Kavvadias and Stavropoulos [Kavvadias99], and Uno [Uno02]. We developed an algorithm by interleaving dualization with finding maximal frequent sets. Roughly speaking, our algorithm solves one dualization problem with the size $|Bd^+|$, in which Bd^+ is the set of maximal frequent sets, while the algorithm of Gunopulos et al. solves $|Bd^+|$ dualization problems with sizes from 1 through $|Bd^+|$. This reduces the computation time by a factor of $1/|Bd^+|$.

To reduce the computation time more, we used Uno's dualization algorithm [Uno02]. The experimental computation time of Uno's algorithm is linear in the number of outputs, and $O(|E|)$ per output, while that of Kavvadias and Stavropoulos seems to be $O(|E|^2)$. This reduces the computation time by a factor of $1/|E|$. Moreover, we add an improvement based on sparseness of input. By this, the experimental computation time per output is reduced to $O(ave(Bd^+))$ where $ave(Bd^+)$ is the average size of maximal frequent sets. In summary, we reduced the computation time by a factor of $ave(Bd^+)$ / $(|Bd^+| \times |E|^2)$ by using the combination of the algorithm of Gunopulos et al. and the algorithm of Kavvadias and Stavropoulos.

In the following sections, we describe our algorithm and the computational result. Section 2 describes the algorithm of Gunopulos et al. and Section 3 describes our algorithm and Uno's algorithm. Section 4 explains our improvement using sparseness. Computational experiments for FIMI'03 instances are shown in Section 5, and we conclude the paper in Section 6.

Dualize and Advance[Gunopulos97a]
1 $Bd^+ := \{go\_up(\emptyset)\}$
2 Compute $MHS(\overline{Bd^+})$.
3 If no set in $MHS(\overline{Bd^+})$ is frequent, output $MHS(\overline{Bd^+})$.
4 If there exists a frequent set S in $MHS(\overline{Bd^+})$, $Bd^+ := Bd^+ \cup \{go\_up(S)\}$ and go to 2.

Figure 1: Dualize and Advance Algorithm

2. Enumerating maximal frequent sets by dualization

In this section, we describe the algorithm of Gunopulos et al. Explanations are also in [Gunopulos97a, Gunopulos97b, SatohUno03], however, those are written with general terms. In this section, we explain in terms of frequent set mining.

Let Bd^- be the set of minimal infrequent sets. For a subset family $\mathcal{H}$ of E, a **hitting set** HS of $\mathcal{H}$ is a set such that for every $S \in \mathcal{H}$, $S \cap HS \neq \emptyset$. If a hitting set includes no other hitting set, then it is said to be **minimal**. We denote the set of all minimal hitting sets of $\mathcal{H}$ by $MHS(\mathcal{H})$. We denote the complement of a subset S w.r.t. E by $\overline{S}$. For a subset family $\mathcal{H}$, we denote $\{\overline{S}|S \in \mathcal{H}\}$ by $\overline{\mathcal{H}}$.

There is a strong connection between the maximal frequent sets and the minimal infrequent sets by the minimal hitting set operation.

Proposition 1 *[Mannila96]* $Bd^- = MHS(\overline{Bd^+})$

Using the following proposition, Gunopulos et al. proposed an algorithm called *Dualize and Advance* shown in Fig. 1 to compute the maximal frequent sets [Gunopulos97a].

Proposition 2 *[Gunopulos97a] Let $Bd^+ \subseteq Bd^+$. Then, for every $S \in MHS(\overline{Bd^+})$, either $S \in Bd^-$ or S is frequent (but not both).*

In the above algorithm, $go\_up(S)$ for a subset S of E is a maximal frequent set which is computed as follows.

1. Select one element e from $\overline{S}$ and check the frequency of $S \cup \{e\}$.
2. If it is frequent, $S := S \cup \{e\}$ and go to 1.
3. Otherwise, if there is no element e in $\overline{S}$ such that $S \cup \{e\}$ is frequent, then return S.

Proposition 3 *[Gunopulos97a] The number of frequency checks in the "Dualize and Advance" algorithm to compute Bd^+ is at most $|Bd^+| \cdot |Bd^-| + |Bd^+| \cdot |E|^2$.*

Basically, the algorithm of Gunopulos et al. solves dualization problems with sizes from 1 through $|Bd^+|$. Although we can terminate dualization when we find a new maximal frequent set, we may check each minimal infrequent item set again and again. This is one of the reasons that the algorithm of Gunopulos et al. is not fast in practice. In the next section, we propose a new algorithm obtained by interleaving *gp_up* into a dualization algorithm. The algorithm basically solves one dualization problem of size $|Bd^+|$.

3. Description of our algorithm

The key lemma of our algorithm is the following.

Lemma 1 *[SatohUno03] Let Bd_1^+ and Bd_2^+ be subsets of Bd^+. If $Bd_1^+ \subseteq Bd_2^+$,*

$$MHS(\overline{Bd_1^+}) \cap Bd^- \subseteq MHS(\overline{Bd_2^+}) \cap Bd^-$$

Suppose that we have already found minimal hitting sets corresponding to $\overline{Bd^+}$ of a subset Bd^+ of the maximal frequent sets. The above lemma means that if we add a maximal frequent set to Bd^+, any minimal hitting set we found which corresponds to a minimal infrequent set is still a minimal infrequent set. Therefore, if we can use an algorithm to visit each minimal hitting set based on an incremental addition of maximal frequent sets one by one, we no longer have to check the same minimal hitting set again even if maximal frequent sets are newly found. The dualization algorithms proposed by Kavvadias and Stavropoulos [Kavvadias99] and Uno[Uno02] are such kinds of algorithms. Using these algorithms, we reduce the number of checks.

Let us show Uno's algorithm [Uno02]. This is an improved version of Kavvadias and Stavropoulos's algorithm [Kavvadias99]. Here we introduce some notation. A set $S \in \mathcal{H}$ is called *critical* for $e \in hs$, if $S \cap hs = \{e\}$. We denote a family of critical sets for e w.r.t. hs and $\mathcal{H}$ as $crit(e, hs)$. Note that mhs is a minimal hitting set of $\mathcal{H}$ if and only if for every $e \in mhs$, $crit(e, mhs)$ is not empty.

```
global S_0, ..., S_m;
compute_mhs(i, mhs) /* mhs is a minimal hitting set of S_0, ..., S_i */
begin
1   if i == m then output mhs and return;
2   else if S_{i+1} ∩ mhs ≠ ∅ then compute_mhs(i + 1, mhs);
    else
    begin
3   for every e ∈ S_{i+1} do
4     if for every e' ∈ mhs, there exists S_j ∈ crit(e', mhs), j ≤ i
        s.t. S_j does not contain e then
5       comupute_mhs(i + 1, mhs ∪ {e});
    end
    return;
end
```

Figure 2: Algorithm to Enumerate Minimal Hitting Sets

Suppose that $\mathcal{H} = \{S_1, ..., S_m\}$, and let MHS_i be $MHS(\{S_0, ..., S_i\})(1 \leq i \leq n)$. We simply denote $MHS(\mathcal{H})$ by MHS. A hitting set hs for $\{S_1, ..., S_i\}$ is minimal if and only if $crit(e, hs) \cap \{S_1, ..., S_i\} \neq \emptyset$ for any $e \in hs$.

Lemma 2 *[Uno02] For any $mhs \in MHS_i(1 \leq i \leq n)$, there exists just one minimal hitting set $mhs' \in MHS_{i-1}$ satisfying either of the following conditions (but not both),*

- $mhs' = mhs$.
- $mhs' = mhs \setminus \{e\}$ *where* $crit(e, mhs) \cap \{S_0, ..., S_i\} = \{S_i\}$.

We call mhs' the *parent* of mhs, and mhs a *child* of mhs'. Since the parent-child relationship is not cyclic, its graphic representation forms a forest in which each of its connected components is a tree rooted at a minimal hitting set of MHS_1. We consider the trees as traversal routes defined for all minimal hitting sets of all MHS_i. These transversal routes can be traced in a depth-first manner by generating children of the current visiting minimal hitting set, hence we can enumerate all minimal hitting sets of MHS in linear time of $\sum_i |MHS_i|$. Although $\sum_i |MHS_i|$ can be exponential to $|MHS|$, such cases are expected to be exceptional in practice. Experimentally, $\sum_i |MHS_i|$ is linear in $|MHS|$.

To find children of a minimal hitting set, we use the following proposition that immediately leads from the above lemma.

Proposition 4 *[Uno02]*
Any child mhs' of $mhs \in MHS_i$ satisfies one of the following conditions.
(1) $mhs' = mhs$
(2) $mhs' = mhs \cup \{e\}$
In particular, no mhs has a child satisfying (1) and a child satisfying (2).

If $mhs \cap S_{i+1} \neq \emptyset$ then $mhs \in MHS_{i+1}$, and (1) holds. If $mhs \cap S_{i+1} = \emptyset$, then $mhs \notin MHS_{i+1}$, and (2) can hold for some $e \in S_{i+1}$. If $mhs' = mhs \cup \{e\}$ is a child of mhs, then for any $e' \in mhs$, there is $S_j \in crit(e', mhs), j \leq i$ such that $e \notin S_j$. From these observations, we obtain the algorithm described in Fig. 2.

An iteration of the algorithm in Fig. 2 takes:

- O($|mhs|$) time for line 1.
- O($|S_{i+1} \cup mhs|$) time for line 2.
- O($(|E| - |mhs|) \times \sum_{e' \in mhs} |crit(e', mhs) \cap \{S_0, ..., S_i\}|$) time for lines 3 to 5, except for the computation of *crit*.

To compute *crit* quickly, we store $crit(e, mhs)$ in memory, and update them when we generate a recursive call. Note that this takes O(m) memory. Since $crit(e', mhs \cup \{e\})$ is obtained from $crit(e', mhs)$ by removing sets including e (i.e., $crit(e', mhs \cup \{e\}) = \{S | S \in crit(e', mhs), e' \notin S_{i+1}\}$), $crit(e', mhs \cup \{e\})$ for all e' can be computed in O(m) time. Hence the computation time of an iteration is bounded by O($|E| \times m$).

Based on this dualization algorithm, we developed a maximal frequent sets enumeration algorithm. First, the algorithm sets the input $\mathcal{H}$ of the dualization problem to the empty set. Then, the algorithm solves the dualization in the same way as the

Irredundant Border Enumerator
global integer $bdpnum$; sets $bd_0^+, bd_1^+ ....$;
main()
begin
  $bdpnum := 0$;
  $construct\_bdp(0, \emptyset)$;
  output all the $bd_j^+ (0 \leq j \leq bdpnum)$;
end

$construct\_bdp(i, mhs)$
begin
  if $i == bdpnum$ /* minimal hitting set for $\cup_{j:=0}^{bdpnum} \overline{bd_j^+}$ is found */
    then goto 1 **else goto** 2

1. **if** mhs is not frequent, **return**; /* new Bd^- element is found */
   $bd^+_{bdpnum} := go\_up2(mhs)$; /* new Bd^+ element is found */
   $bdpnum := bdpnum + 1$; /* proceed to 2 */

2. **if** $\overline{bd_i^+} \cap mhs \neq \emptyset$ **then** $construct\_bdp(i+1, mhs)$;
   else
   begin
   for every $e \in \overline{bd_i^+}$ **do**
     if $bd_i^+ \cup \{e\}$ is a minimal hitting set of $\{\overline{bd_0^+}, \overline{bd_1^+} ..., \overline{bd_{i-1}^+}\})$ **then** $construct\_bdp(i+1, mhs \cup \{e\})$;
   return;

end

Figure 3: Algorithm to Check Each Minimal Hitting Set Only Once

above algorithm. When a minimal hitting set mhs is found, the algorithm checks its frequency. If mhs is frequent, the algorithm finds a maximal frequent set S including it, and adds $\overline{S}$ to $\mathcal{H}$ as a new element of $\mathcal{H}$. Now mhs is not a minimal hitting set since $\overline{S} \cap mhs = \emptyset$. The algorithm continues generating a recursive call to find a minimal hitting set of the updated $\mathcal{H}$. In the case that mhs is not frequent, from Lemma 1, mhs continues to be a minimal hitting set even when $\mathcal{H}$ is updated. Hence, we backtrack and find other minimal hitting sets.

When the algorithm terminates, $\overline{\mathcal{H}}$ is the set of maximal frequent sets, and the set of all minimal hitting sets the algorithm found is the set of minimal infrequent sets. The recursive tree the algorithm generated is a subtree of the recursive tree obtained by Uno's dualization algorithm inputting $\overline{Bd^+}$, which is the set of the complement of maximal frequent sets.

This algorithm is described in Fig. 3. We call the algorithm *Irredundant Border Enumerator* (IBE algorithm, for short).

Theorem 1 *The computation time of IBE is* $O(Dual(\overline{Bd^+}) + |Bd^+|g)$, *where* $Dual(\overline{Bd^+})$ *is the computation time of Uno's algorithm for dualizing* $\overline{Bd^+}$, *and* g *is the computation time for go_up.*

Note also that, the space complexity of the IBE algorithm is $O(\Sigma_{S \in Bd^+} |S|)$ since all we need to memorize is Bd^+ and once a set in Bd^- is checked, it is no longer need to be recorded. On the other hand, Gunopulos et al. [Gunopulos97a] suggests a usage of Fredman and Khachiyan's algorithm [Fredman96] which needs a space of $O(\Sigma_{S \in (Bd^+ \cup Bd^-)} |S|)$ since the algorithm needs both Bd^+ and Bd^- at the last stage.

4. Using sparseness

In this section, we speed up the dualization phase of our algorithm by using a sparseness of $\mathcal{H}$. In real data, the sizes of maximal frequent sets are usually small. They are often bounded by a constant. We use this sparse structure for accelerating the algorithm.

```
global S_0, ..., S_m;
compute_mhs(i, mhs) /* mhs is a minimal hitting set of S_0, ..., S_i */
begin
1   if uncov(mhs) == ∅ then output mhs and return;
2   i := minimum index of uncov(mhs) ;
3   for every e ∈ mhs do
4     increase the counter of items in ∪_{S∈crit(mhs,e)} S̄ by one
    end
5   for every e' ∉ mhs s.t. counter is increased by |mhs| do /* items included in all ∪_{S∈crit(mhs,e)} S̄ */
6     compute_mhs(i + 1, mhs ∪ {e});
    return;
end
```

Figure 4: Improved Dualization Algorithm Using Sparseness

First, we consider a way to reduce the computation time of iterations. Let us see the algorithm described in Fig. 2. The bottle neck part of the computation of an iteration of the algorithm is lines 3 to 5, which check the existence of a critical set $S_j \in crit(mhs, e'), j < i$ such that $e \notin S_j$. To check this condition for an item $e \notin mhs$, we spend $O(\sum_{e' \in mhs} |crit(mhs, e')|)$ time, hence this check for all $e \notin mhs$ takes $O((|E| - |mhs|) \times \sum_{e' \in mhs} |crit(mhs, e')|)$ time.

Instead of this, we compute $\bigcup_{S \in crit(mhs,e)} \overline{S}$ for each $e \in mhs$. If and only if $e' \in \bigcup_{S \in crit(mhs,e)} \overline{S}$ for all $e \in mhs$, e' satisfies the condition of "if" at line 4. To compute $\bigcup_{S \in crit(mhs,e)} \overline{S}$ for all $e \in mhs$, we take $O(\sum_{e \in mhs} \sum_{S \in crit(mhs,e)} |\overline{S}|)$ time. In the case of IBE algorithm, $\overline{S}$ is a maximal frequent set, hence the average size of $|\overline{S}|$ is expected to be small. The sizes of minimal infrequent sets are not greater than the maximum size of maximal frequent sets, and they are usually smaller than the average size of the maximal frequent sets. Hence, $|mhs|$ is also expected to be small.

Second, we reduce the number of iterations. For $mhs \subseteq E$, we define $uncov(mhs)$ by the set of $S \in \mathcal{H}$ satisfying $S \cap mhs = \emptyset$. If $mhs \cap S_i \neq \emptyset$, the iteration inputting mhs and i does nothing but generates a recursive call with increasing i by one. This type of iterations should be skipped. Only iterations executing lines 3 to 5 are crucial. Hence, in each iteration, we set i to the minimum index among $uncov(mhs)$. As a result of this, we need not execute line 2, and the number of iterations is reduced from $\sum_i |MHS_i|$ to $|\bigcup_i MHS_i|$. We describe the improved algorithm in Fig. 4.

In our implementation, when we generate a recursive call, we allocate memory for each variable used in the recursive call. Hence, the memory required by the algorithm can be up to $O(|E| \times m)$. However, experimentally the required memory is always linear in the input size. Note that we can reduce the worst case memory complexity by some sophisticated algorithms.

5. Experiments

In this section, we show some results of the computational experiments of our algorithms. We implement our algorithm using C programming language, and examined instances of FIMI2003. For instances of KDD-cup 2000[KDDcup00], we compared the results to the computational experiments of CHARM[Zaki02], closed[Pei00], FP-growth[Han00], and Apriori[Agrawal96] shown in [Zheng01]. The experiments in [Zheng01] were done on a PC with a Duron 550MHz CPU and 1GB RAM memory. Our experiments were done on a PC with a Pentium III 500MHz CPU and 256MB RAM memory, which is little slower than a Duron 550MHz CPU. The results are shown in Figs. 4 – 14. Note that our algorithm uses at most 170MB for any following instance. We also show the number of frequent sets, frequent closed/maximal item sets, and minimal frequent sets.

In our experiments, IBE algorithm takes approximately $O(|Bd^-| \times ave(Bd^+))$ time, while the computation time of other algorithms deeply depends on the number of frequent sets, the number of frequent closed item sets, and the minimum support. We recall that $ave(Bd^+)$ is the average size of maximal frequent sets. In some instances, our IBE algorithm performs rather well compared to other algorithms. In these cases, the number of maximal frequent item

sets is smaller than number of frequent item sets. IBE algorithm seems to give a good performance for difficult problems such that the number of maximal frequent sets is very small rather than those of frequent item sets and frequent closed item sets.

6. Conclusion

In this paper, we describe the detailed implementation method of our algorithm proposed in [SatohUno03] and we give some experimental results on test data.

Acknowledgments
We are grateful to Heikki Mannila for participating useful discussions about this research.

References

[Agrawal96] Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., and Verkamo, A. I., "Fast Discovery of Association Rules", U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, (eds), Advances in Knowledge Discovery and Data Mining, chapter 12, pp. 307–328 (1996).

[Fredman96] Fredman, M. L. and Khachiyan, L., "On the Complexity of Dualization of Monotone Disjunctive Normal Forms", Journal of Algorithms 21(3), pp. 618 – 628 (1996)

[Gunopulos97a] Gunopulos, D., Khardon, R., Mannila, H. and Toivonen, H., "Data mining, Hypergraph Transversals, and Machine Learning", *Proc. of PODS'97*, pp. 209 – 216 (1997).

[Gunopulos97b] Gunopulos, D., Mannila, H., and Saluja, S., "Discovering All Most Specific Sentences using Randomized Algorithms", *Proc. of ICDT'97*, pp. 215 – 229 (1997).

[Han00] Han, J., Pei, J., Yin, Y., "Mining Frequent Patterns without Candidate Generation," *SIGMOD Conference 2000*, pp. 1-12, 2000

[Kavvadias99] Kavvadias, D. J., and Stavropoulos, E. C., "Evaluation of an Algorithm for the Transversal Hypergraph Problem", *Algorithm Engineering*, pp 72 – 84 (1999).

[KDDcup00] Kohavi, R., Brodley, C. E., Frasca, B., Mason, L., and Zheng, Z., "KDD-Cup 2000 Organizers' Report: Peeling the Onion," *SIGKDD Explorations,* 2(2), pp. 86-98, 2000.

[Mannila96] Mannila, H. and Toivonen, T., "On an Algorithm for Finding All Interesting Sentences", *Cybernetics and Systems, Vol II, The Thirteen European Meeting on Cybernetics and Systems Research*, pp. 973 – 978 (1996).

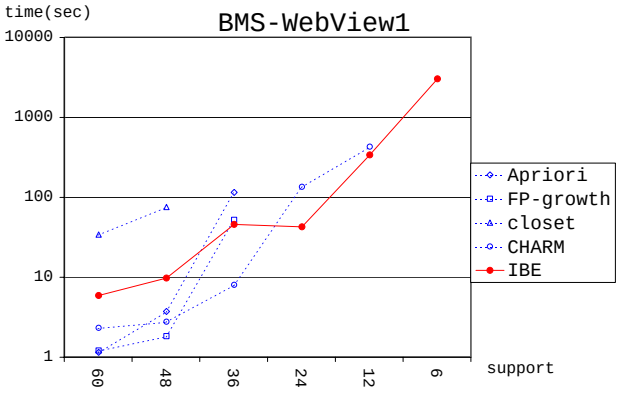
[Pei00] Pei, J., Han, J., Mao, R., "CLOSET: An Efficient Algorithm for Mining Frequent Closed Itemsets," *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery 2000*, pp. 21-30, 2000.

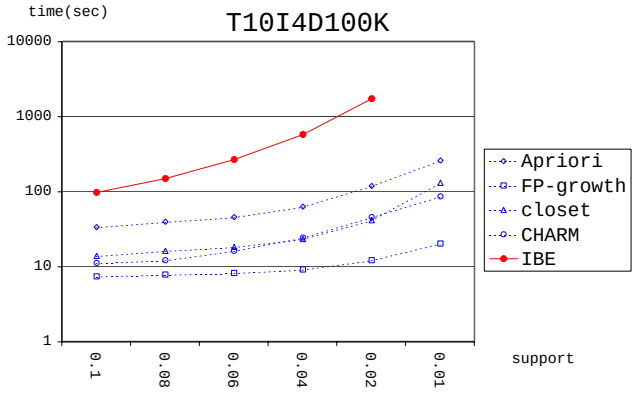
[SatohUno03] Satoh, K., Uno, T., "Enumerating Maximal Frequent Sets using Irredundant Dualization", *Lecture Notes in Artificial Intelligence* (Proc. of Discovery Science 2003), Springer-Varlag, pp. 192-201, 2003.

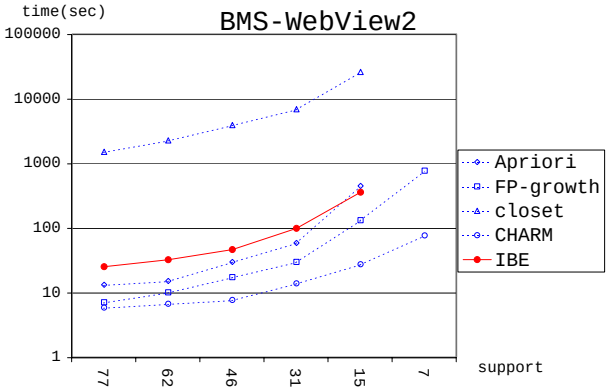
[Uno02] Uno, T., "A Practical Fast Algorithm for Enumerating Minimal Set Coverings", *SIGAL83*, Information Processing Society of Japan, pp. 9 – 16 (in Japanese) (2002).

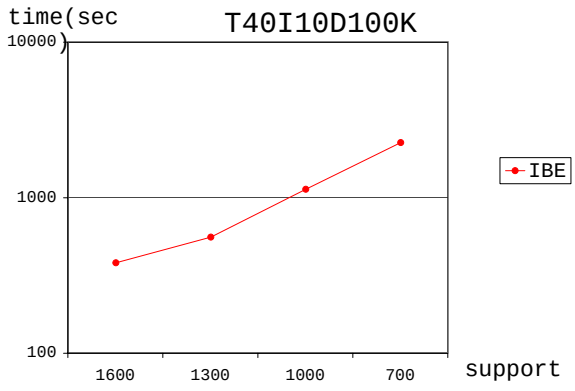
[Zaki02] Zaki, M. J., Hsiao, C., "CHARM: An Efficient Algorithm for Closed Itemset Mining," *2nd SIAM International Conference on Data Mining (SDM'02)*, pp. 457-473, 2002.

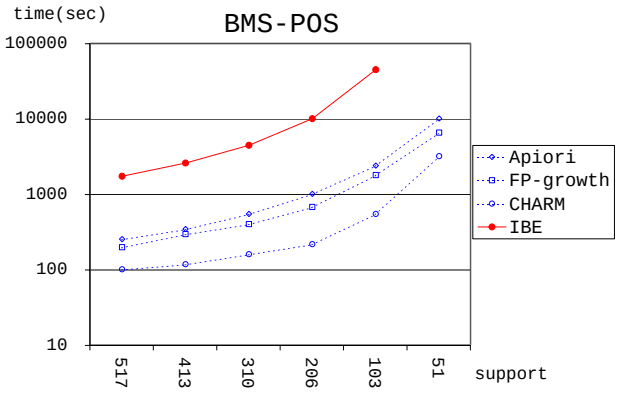
[Zheng01] Zheng, Z., Kohavi, R., and Mason, L., "Real World Performance of Association Rule Algorithms," *KDD 2001*, pp. 401-406, 2001.

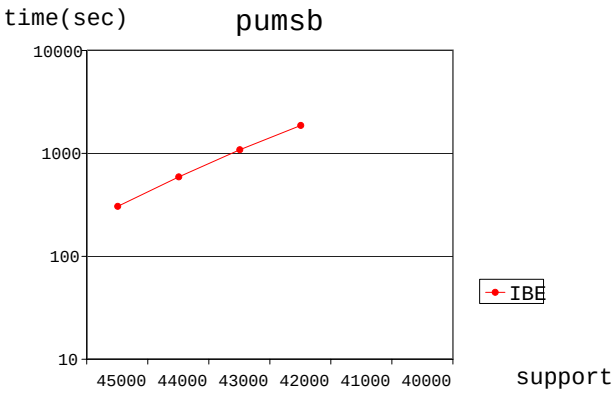

time(sec)
BMS-WebView1
10000
1000
100
10
1
60
48
36
24
12
6
support
Apriori
FP-growth
closet
CHARM
IBE


time(sec)
T10I4D100K
10000
1000
100
10
1
0.1
0.08
0.06
0.04
0.02
0.01
support
Apriori
FP-growth
closet
CHARM
IBE

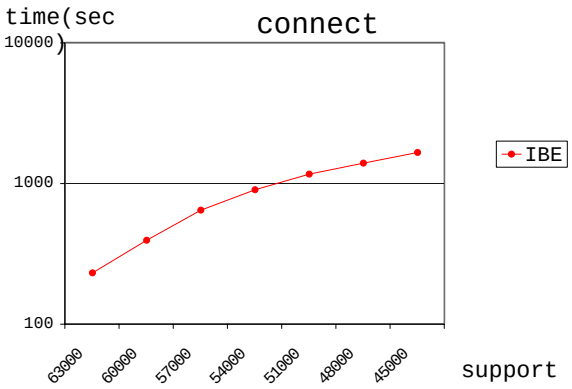

time(sec)
BMS-WebView2
100000
10000
1000
100
10
1
77
62
46
31
15
7
support
Apriori
FP-growth
closet
CHARM
IBE

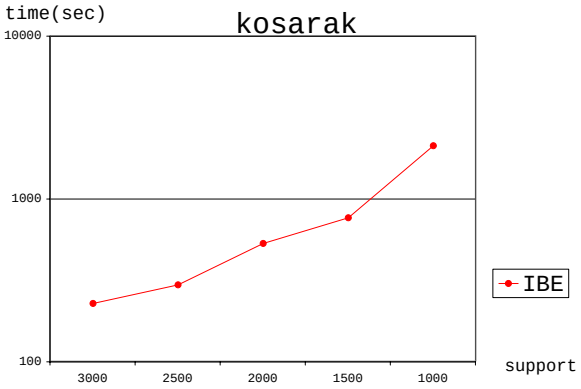

time(sec
T40I10D100K
10000)
1000
100
1600
1300
1000
700
support
IBE

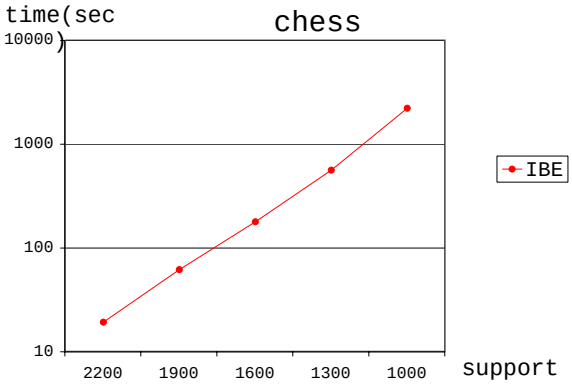

time(sec)
BMS-POS
100000
10000
1000
100
10
517
413
310
206
103
51
support
Apriori
FP-growth
CHARM
IBE

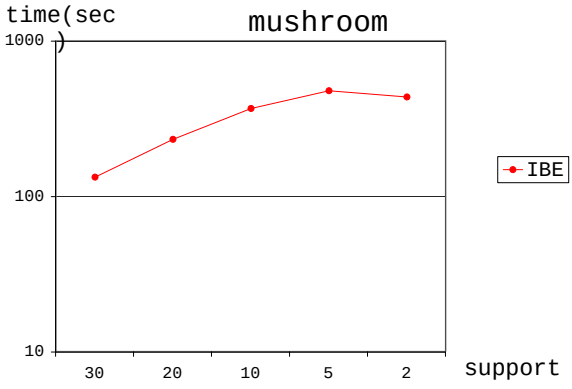

time(sec)
pumsb
10000
1000
100
10
45000
44000
43000
42000
41000
40000
support
IBE


time(sec)
pumsb_star
10000
1000
100
10
1
30000
25000
20000
15000
10000
support
IBE


time(sec
connect
10000)
1000
100
63000
60000
57000
54000
51000
48000
45000
support
IBE


time(sec)
kosarak
10000
1000
100
3000
2500
2000
1500
1000
support
IBE


time(sec
chess
10000)
1000
100
10
2200
1900
1600
1300
1000
support
IBE


time(sec
)
mushroom
1000
100
10
30
20
10
5
2
support
IBE

BMS-Web-View1: #item 497, #transactions, 59602, ave. size of transaction 2.51

support	60	48	36	24	12	6
Apriori	1.1	3.6	113	-	-	-
FP-growth	1.2	1.8	51	-	-	-
Closet	33	74	-	-	-	-
Charm	2.2	2.7	7.9	133	422	-
IBE	5.8	9.6	45	42	333	2982
#freq. sets	3992	10287	461522	-	-	-
#closed sets	3974	9391	64762	155651	422692	1240701
#max. freq. sets	2067	4028	15179	12956	84833	129754
#min. infreq. sets	66629	81393	150278	212073	579508	4320003

maximum use of memory: 45MB

BMS-Web-View2: #items 3340, #transactions 77512, ave. size of transaction 4.62

support	77	62	46	31	15	7
Apriori	13.1	15	29.6	58.2	444	-
Fp-growth	7.03	10	17.2	29.6	131	763
Closet	1500	2250	3890	6840	25800	-
Charm	5.82	6.66	7.63	13.8	27.2	76
IBE	25	32	46	98	355	1426
#closed sets	22976	37099	60352	116540	343818	754924
#freq. sets	24143	42762	84334	180386	1599210	9897303
#max. freq. sets	3901	5230	7841	16298	43837	118022
#min. infreq. sets	657461	958953	1440057	2222510	3674692	5506524

maximal use of memory: 100MB

BMS-POS: #items 1657, #transactions 517255, ave. size of transaction 6.5

support	517	413	310	206	103	51
Apriori	251	341	541	1000	2371	10000
Fp-growth	196	293	398	671	1778	6494
Closet	-	-	-	-	-	-
Charm	100	117	158	215	541	3162
IBE	1714	2564	4409	9951	44328	-
#closed sets	121879	200030	378217	840544	1742055	21885050
#freq. sets	121956	200595	382663	984531	5301939	33399782
#max. freq. sets	30564	48015	86175	201306	891763	4280416
#min. infreq. sets	236274	337309	530946	1047496	3518003	-

maximum use of memory: 110MB

T10I4D100K: #items 1000, #transactions 100000, ave. size of transaction 10

support	100	80	60	40	20	10
Apriori	33	39	45	62	117	256
Fp-growth	7.3	7.7	8.1	9.0	12	20
Closet	13	16	18	23	41	130
Charm	11	13	16	24	45	85
IBE	96	147	263	567	1705	-
#freq. sets	15010	28059	46646	84669	187679	335183
#closed sets	13774	22944	38437	67537	131342	229029
#max. freq. sets	7853	11311	16848	25937	50232	114114
#min. infreq. sets	392889	490203	736589	1462121	4776165	-

maximum use of memory: 60MB

T40I10D100K: #items 1000, #transactions 100000, ave. size of transaction 39.6

support	1600	1300	1000	700
IBE	378	552	1122	2238
#freq. sets	4591	10110	65236	550126
#closed sets	4591	10110	65236	548349
#max. freq. sets	4003	6944	21692	41473
#min. infreq. sets	245719	326716	521417	1079237

maximum memory use: 74MB

pumsb: #items 7117, #transactions 49046, ave. size of transaction 74

support	45000	44000	43000	42000
IBE	301	582	1069	1840
#freq. sets	1163	2993	7044	15757
#closed sets	685	1655	3582	7013
#max. freq. sets	144	288	541	932
#min. infreq. sets	7482	7737	8402	9468

maximum use of memory: 70MB

pumsb_star: #items 7117, #transactions 49046, ave. size of transaction 50

support	30000	25000	20000	15000	10000	5000
IBE	8	19	59	161	556	2947
#freq. sets	165	627	21334	356945	>2G	-
#closed sets	66	221	2314	14274	111849	-
#max. freq. sets	4	17	81	315	1666	15683
#min. infreq. sets	7143	7355	8020	9635	19087	98938

maximum use of memory: 44MB

kosarak: #items 41217, #transactions 990002, ave. size of transaction 8

support	3000	2500	2000	1500	1000
IBE	226	294	528	759	2101
#freq. sets	4894	8561	34483	219725	711424
#closed sets	4865	8503	31604	157393	496675
#max. freq. sets	792	1146	2858	4204	16231
#min. infreq. sets	87974	120591	200195	406287	875391

maximum use of memory: 170MB

mushroom: #items 120, #transactions 8124, ave. size of transaction 23

support	30	20	10	5	2
IBE	132	231	365	475	433
#freq. sets	505205917	781458545	1662769667	>2G	>2G
#closed sets	91122	109304	145482	181243	230585
#max. freq. sets	15232	21396	30809	34131	27299
#min. infreq. sets	66085	79481	81746	69945	31880

maximum use of memory 47MB

connect: #items 130, #transactions 67577, ave. size of transaction 43

support	63000	60000	57000	54000	51000	48000	45000
IBE	229	391	640	893	1154	1381	1643
#freq. sets	6327	41143	171239	541911	1436863	-	-
#closed sets	1566	4372	9041	15210	23329	-	-
#max. freq. sets	152	269	464	671	913	1166	1466
#min. infreq. sets	297	486	703	980	1291	1622	1969

maximum use of memory: 60MB

chess: #items 76, #transactions 3196, ave. size of transaction 37

support	2200	1900	1600	1300	1000
IBE	19	61	176	555	2191
#freq. sets	59181	278734	1261227	5764922	29442848
#closed sets	28358	106125	366529	1247700	4445373
#max. freq. sets	1047	3673	11209	35417	114382
#min. infreq. sets	1725	5202	14969	46727	152317

maximum use of memory: 50MB