FGC: An Efficient Constraint Based Frequent Set Miner

Russel Pears and Sangeetha Kutty
School of Computer and Information Sciences
Auckland University of Technology, Auckland, New Zealand
rpears@aut.ac.nz

Abstract

Despite advances in algorithmic design, association rule mining remains problematic from a performance viewpoint when the size of the underlying transaction database is large. The well-known Apriori approach, while reducing the computational effort involved still suffers from the problem of scalability due to its reliance on generating candidate itemsets. In this paper we present a novel approach that combines the power of pre-processing with the application of user-defined constraints to prune the itemset space prior to building a compact FP-tree. Experimentation shows that that our algorithm significantly outperforms the current state of the art algorithm, FP-bonsai.

1. Introduction

The problem of incorporating user-defined constraints into association rule mining algorithms has received a great deal of attention recently. It has been recognised that embedding constraints into the mining process leads to both performance gains and higher levels of user interaction [8, 14, 17]. Performance improves as itemsets that violate an anti-monotone constraint can be pruned from the search space as any supersets of these itemsets are guaranteed to violate the constraint as well. On the other hand, monotone constraints have been shown to be effective in trimming the transaction database of items that violate the monotone constraint. Constraints also promote a higher level of user involvement as users can precisely specify what ietmsets that they are interested in. For example, a user interested in mining dairy products and may specify that all frequent itemsets containing dairy products whose total value exceeds $100 be returned.

Various approaches such as the ExAnte [5] and FP-bonsai [6] embed both monotone and anti-monotone constraints into base association rule mining algorithms such as Apriori [3] and FP-growth [20]. Although these constraint embedding techniques have produced significant performance gains in base algorithms there still exists room for improvement from a performance point of view due to the ever-increasing size of databases.

Basically the performance of any association rule mining algorithm is limited by the sheer number of 1- and 2- frequent itemsets present in the dataset, especially when mining at low support thresholds [19]. The FOLD-Growth approach [10] overcomes this limitation by making use of a pre-processing structure, the SOTrieIT (Support Ordered Trie Itemset), to identify 1- and 2- frequent itemsets with minimal cost. However, it does not exploit the use of constraints in pruning itemsets and relies purely on the use of pre-processing to speed up the process of frequent itemset generation.

In this paper we propose an approach that combines the strengths of pre-processing together with a constraint embedding technique that prunes itemsets that survive the pre-processing phase. We show that anti-monotone constraints are very effective when applied to the SoTrieIT structure and that monotone constraints play an important role in pruning the transaction dataset.

The contributions that we make are as follows. Firstly, we attempt to bridge the gap between specialised association rule mining methods and work done in constraint-based mining by providing a high performance mining algorithm that combines strengths from these two areas. Secondly, we study the nature of monotone and anti-monotone constraints and exploit them deep inside the algorithm. We then study the impact of each type of constraint on our new algorithm. Finally, we carry out a systematic analysis to assess the sensitivity of factors that impact on performance such as constraint selectivity, support threshold and dataset type.

The rest of the paper is organized as follows. In section 2 we present a formal definition of the problem. A review of related work is discussed in section 3. Section 4 contains a description of the FGC algorithm, while section 5 presents a performance comparison of running the FGC algorithm against an implementation of FP-bonsai algorithm [6]. The FP-bonsai algorithm was chosen as the baseline as it is the state-of-the-art algorithm in the area of association rule mining. We conclude in section 6 with a presentation on some ideas for future work.

2. Problem definition

Let $I = \{I_1, I_2, I_3, \ldots\ldots, I_n\}$ be the universal set of items.

A k-frequent itemset F is some subset of I such that cardinality (F) ≥ *min_supp*, where *min_supp* is the support threshold. We are interested in the set of all frequent itemsets that satisfy user-defined constraints. In this research we focus on monotone and anti-monotone constraints.

The frequent itemset mining problem in the presence of monotone and anti-monotone constraints requires the identification of all itemsets $P \subseteq I$ such that cardinality(P)≥ *min_supp* and $C_{AM}(P)$= true and $C_M(P)$ =*true*, where C_{AM} and C_M are anti-monotone and monotone constraints respectively. Henceforth in this paper we will use $C_{AM}(P)$ as a shorthand for $C_{AM}(P)$ = *true*, and likewise $C_M(P)$ as a shorthand for $C_M(P)$ =*true*.

3. Related research

Research in association rule mining has spanned over a decade due to its application in a wide variety of areas such as identifying correlations, multi-dimensional patterns, partial patterns and periodicity. The seminal work by Agrawal [2] resulted in the Apriori algorithm which sparked a flurry of research in this area. The algorithm uses a simple property called the *Apriori* heuristic that limits the number of candidate itemsets that need to be tested. The basic intuition is that any subset of a frequent itemset also has to be frequent. This eliminated the need for testing every possible combination of items and thus the time for generating the frequent itemsets was speeded up significantly.

The main drawback of the Apriori approach is that it requires repeated scans of the transaction database and the accompanying effort involved in candidate generation. Apriori works in a level-by-level fashion and to compute the 2-frequent itemsets it has to scan the entire transaction database, identify the 1-frequent itemsets and use the Apriori principle to generate the candidate 2-frequent itsemsets. These candidates then need to be checked against the transaction database to verify whether they qualify as 2-frequent itemsets. Thus to generate k-frequent itemsets a total of k database scans will be needed in general, with each scan requiring generation and testing of candidate itemsets.

In recent years, the FP-growth algorithm was proposed to eliminate the major bottleneck of repeated database scanning and candidate generation inherent in Apriori-like algorithms [11, 12]. FP-growth uses a new compact data structure, the FP-tree, to store transactions in a trie-like structure with every item having a linked list representing each transaction. This algorithm does not use the *generate-and-test* paradigm employed by Apriori-type algorithms, rather it uses a *divide-and-conquer* technique and thus represents a radical departure in strategy. Frequent itemset generation requires only 2 scans of the transaction database. In the first scan all 1-frequent itemsets are identified. The infrequent items are discarded and the 1-frequent items are sorted in support descending order. A second scan over the transaction dataset is conducted and a branch in the FP-tree is created for each transaction. Transactions that have a common item prefix share a common sub-tree that is rooted at the shared item. If many transactions share the same items, the FP-tree will represent a compact version of the transaction database. Each time a branch is traversed the support counts for all items along the branch are incremented by 1. Once the FP-tree has been constructed the transaction database can effectively be discarded. A tree traversal of the FP-tree can then be performed to generate frequent itemsets without the need for further scanning of the transaction database.

In comparison with Apriori, FP-growth performs well on dense datasets, where the former suffers due to the huge overheads of candidate generation, resulting in memory overload [12]. In spite of FP-growth's efficient data structure and mining techniques, there has been a significant amount of criticism levelled at it. The criticisms refer to FP-growth as a complex algorithm [22, 16] and one that has been tailored to dense datasets [13]. Indeed, the performance improvement over Apriori is not impressive for sparse datasets having short length transactions [13, 20]. This has resulted in a number of attempts to improve the performance of FP-growth on sparse datasets.

One such attempt is the FOLD-growth algorithm [23], which not only aims to provide an improved version of FP-growth but also distributes the mining effort over time by using a pre-processing data structure called the Support Ordered Trie Itemset (SOTrieIT). The first level of the SOTrieIT contains a node for every unique item that appears in the transaction database. The nodes in the first level are sorted in support decreasing order and each node is split into child nodes which represent items that appear in combination with the parent item in the transaction database.

The SOTrieIT structure supports the identification of 1 and 2- frequent itemsets by a simple tree traversal. For example, to identify itemsets which have a minimum support threshold of 3, Figure 1 shows that only nodes C and A with their corresponding children need to be examined.

The effectiveness of the SOTrieIT stems from two factors: firstly, as mentioned before the major bottleneck in association rule mining is the identification of the 1- and 2-frequent itemsets. Given a support threshold these itemsets can be found very quickly without the need for candidate generation and testing by a simple tree traversal of the relevant portion of the pre-processed tree structure.

Secondly, the SOTrieIT, unlike the FP-Tree does not have to be re-built if the support threshold changes. This makes it very attractive from a performance viewpoint as the structure can be incrementally maintained without the need for major reorganization. Once the 1 and 2-frequent itemsets are identified the FOLD-Growth method trims the transaction database by removing items which are not included in the 1- and 2-frequent itemsets. The trimmed dataset is then used to build a compact version of the FP-tree, which is then mined to produce the required frequent itemsets.

TID	Items
100	AC
200	BC
300	AC
400	ABCD

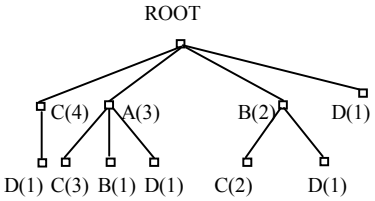

Figure 1: SOTrieIT constructed from sample database

In parallel with improvements in mining frequent itemsets a considerable amount of research has been devoted to the subject of constraint-based mining. All previous research in this area has focused on embedding constraints into either Apriori type algorithms [18, 4] or the FP-tree algorithm [6, 21].

The FP-bonsai approach [6] prunes using both monotone and anti-monotone constraints, extending the ideas of Ex-Ante [4] to an FP-growth algorithm. The advantage of this algorithm is that it uses monotone constraints to complement anti-monotone pruning of the candidate itemsets and the input database. The major strength of this algorithm is that it performs very well for dense datasets. However it performs less efficiently at higher selectivities. Its performance on sparse datasets is also poor due to the time spent in building the FP-tree [6].

The ExAMiner [4] algorithm on the other hand uses anti-monotone pruning efficiently. However, due to its Apriori-like framework it suffers heavily in performance for dense datasets. The higher efficiency of ExAMiner in sparse datasets clearly indicates that FP-bonsai would have benefited from a more efficient strategy of exploiting anti-monotone pruning.

From previous research it is clear that neither the Apriori nor the FP-tree algorithms by themselves are efficient vehicles for exploiting the full power of constraints. The impressive gains in performance made by the FOLD-GROWTH algorithm encouraged us to investigate the effect of applying constraints to the pre-processed SOTrieIT structure to further reduce the itemset space prior to frequent itemset mining.

4. Constraint-based algorithms

We first examine, with the help of a running example, the performance of the FP-bonsai algorithm which exploits constraints but does not make explicit use of pre-processing. We then present our algorithm, called FOLD-Growth with Constraints (FGC) that exploits constraints during both pre-processing phase (involving the SOTrieIT) and mining phases.

We make use of the following transactional database to illustrate the working of each algorithm. Figure 2 (a) shows a sample database with the itemcode-price file for the items in the sample transactional database. We will use *min_supp* = 4 and use *Sum(Price)>10* and *Sum(Price)<25* as the monotone and anti-monotone constraints respectively.

4.1. Mining with FP-bonsai

In FP-bonsai, the database is first scanned to identify the 1-frequent itemsets. The anti-monotone constraint is then applied to identify 1-frequent constraint satisfied itemsets. For each transaction, items that do not appear in the 1-frequent itemset are removed. Figures 2 (a) and (b) show that items F and H are infrequent, and D, though frequent, does not satisfy the anti-monotone constraint. Items D, F and H are thus eliminated from all transactions. The remaining items are then arranged in support descending order. Thereafter, a root node labelled *null* is used to connect the first level nodes. Each transaction is checked against the tree for matching nodes. If at a given node a branch containing the itemsets exists, then the support of that node is incremented by 1, otherwise, a new branch representing the transaction is inserted at that node. Finally, the nodes in the tree are linked to the corresponding items in the header table. Figure 2(c) shows that there are 17 nodes in the resulting FP-tree.

4.2. Mining with the Fold Growth with Constraints (FGC) algorithm

The FGC algorithm works in three phases. In phase 1, the SOTrieIT structure enables us to find 1 and 2-frequent constrained satisfied itemsets (CSI12) quickly as the structure has been pre-built and contains just two levels. As shown in Figure 3, the surviving items are then used to trim the transaction database prior to building the FP-tree in phase 2. Items not present in the 1 and 2 frequent itemsets are removed from each transaction. Finally, the FP-tree that is built in phase 2 is mined in phase 3 to produce the constraint satisfied n-frequent itemsets.

Tid	Itemsets for FP-tree
1	BGC
2	BAE
3	BGCE
4	GAE
5	GCE
6	BCAE
7	BGA
8	BC
9	BGE
10	BGC
11	BGC

(a)

Item code	Support	Price
A	4	5
B	9	3
C	7	14
D	7	30
E	5	23
F	2	15
G	7	6
H	3	12

(b)

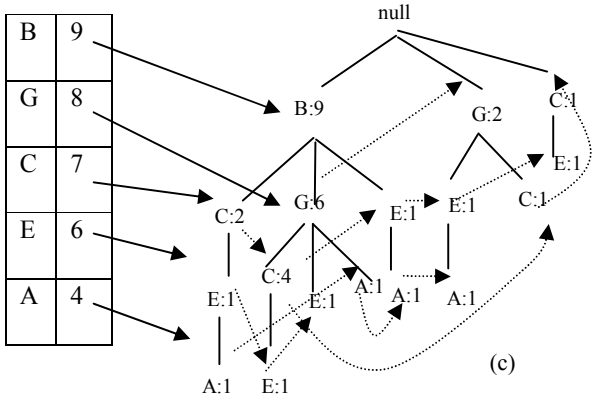

Figure 2: Mining with FP-bonsai

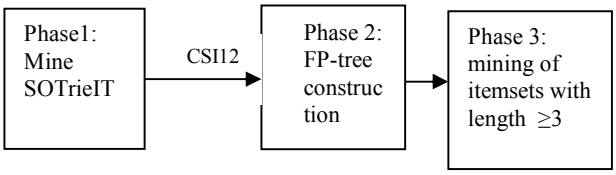

Figure 3: Overview of the FGC Algorithm

Figure 4 shows the results of mining the SOTrieIT and trimming the transaction database of items that do not satisfy either the support threshold or the anti-monotone constraint. Since all 1- and 2-frequent constraint satisfied itemsets have already been identified, any transaction containing less than 3 items can be ignored. This helps to further reduce the number of transactions to 4, as only transactions 1, 3, 10 and 11 now meet the criteria. As shown in Figure 4, these transactions give rise to a tree with just one branch and 3 nodes, which is minimal when compared to the 17 nodes generated by the FP-bonsai algorithm.

We now present an outline of the FGC algorithm. Figure 5 details the algorithm used in phase 1. The rationale behind mining the SOtrieIT is to apply the most effective constraint types first. The support and anti-monotone constraints both enable entire branches to be pruned if they are violated. We first use the support constraint to filter nodes at level 1. The existence of the anti-monotone constraint is then checked, and if it exists it is applied to all surviving nodes. Thereafter the monotone constraint is applied if it is present. With the monotone constraint children of nodes that do not satisfy the constraint need to be checked, unlike with the anti-monotone constraint type.

Figure 6 illustrates the algorithm used in phases 2 and 3 of FGC. Figure 6a illustrates that 2-frequent itemsets are used to trim transactions in addition to the 1-frequent itemsets. This is one of the advantages of FGC over the FP-bonsai approach and is a direct result of mining the SoTrieIT. A further advantage is the fact that FGC only requires one scan of the transaction database, as opposed to FP-bonsai that requires two as the latter does not exploit any pre-processed data structures.

Figure 6b illustrates the dual roles played by the monotone and anti-monotone constraints. The monotone constraint is very effective at trimming transactions in the conditional databases (line 3), while the anti-monotone constraint is effective at reducing the search space (line 6).

Tid	**2- frequent constraint-satisfied itemsets**
1	BG, GC
2	BA
3	BG, GC
4	GA
5	GC
6	BC
7	BG
8	BC
9	BG
10	BG, GC, BC
11	BG, GC, BC

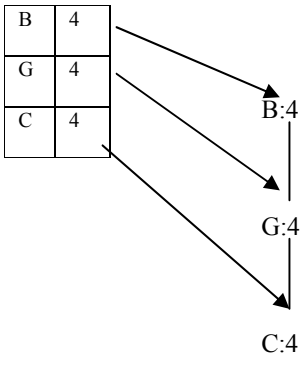

Figure 4: Mining with FGC

We now briefly analyse the space and time complexity of the FGC algorithm. With regards to space complexity, there are two structures to consider: the SOTrieIT generated in Phase 1 and the FP-tree used in phases 2 and 3. The worst-case space complexity of the SOTrieIT has been shown to be $O(n^2)$ in [10], where n represents the total number of items to be mined. The worst case space complexity of the FP-tree generated in Phase 2 is O(d),

where d is the size of the database after trimming of the transactions that contain infrequent items and/or those items that do not satisfy the anti-monotone constraint. This corresponds to the case where there is no sharing of nodes whatsoever in the FP-tree, which is almost always not the case in practice and thus the estimate is a pessimistic one.

If t the number of transactions in the database, then the worst case time complexity for mining of the SOTrieIT is O(nt), as t scans are needed of the SOTrieIT. In practice, this is also a pessimistic estimate as the number of items to be mined is < n, as some of the items will fail either the support or anti-monotone constraint. If m is the number of items surviving Phase1 and p is the average number of items in a transaction after database trimming, then the worst case time complexity of mining the FP tree in phases 2 and 3 is $O(plog(p))+O(tp)+\sum_{i=1}^{m}\text{supp}(a_i)$, where $\text{supp}(a_i)$ is the support of item a_i. The first term represents the time needed to sort each transaction in support descending order, while the second term represents the effort needed to scan the database. The last term accounts for the effort needed to insert nodes into the tree in the worst-case situation where there is no sharing of nodes. Simplifying the above expression, we get the worst case complexity as O(tp) as $\sum_{i=1}^{m}\text{supp}(a_i) \le t$ and $p \le t$.

```
Mine SOTrie(const_type, min_supp)
{     /* CSL1 and CSL2 denote the 1 and 2-constraint satisfied
         sets respectively. */
 for every node x under ROOT
 if supp(x)≥min_supp  // min_supp is the min support threshold
 {
   if (multiple) // if anti-monotone constraint is present use it
    for all x where Cam(x)
    {  // apply the anti-monotone constraint to prune branches
       add x to CLS1 where Cm(x);
       for all children y where Cam(xy) add xy to CSL2 if Cm(xy);
    }
    else if (anti-monotone)
   for all x where Cam(x)
   {
      add x to CLS1;
      for all children y add xy to CSL2 where Cam(xy);
   }
   else
      add x to CSL1 where Cm(x);  // check for monotone
      for all children y add xy to CSL2 where Cm(xy);
 else break,//  no more nodes need to be explored
  }
}
```

Figure 5: Mining the SOTrieIT in Phase 1 of FGC

```
If there are no items the 1-frequent itemset CSL1,
then Terminate algorithm
Else {
    Build a FP-tree using the transaction database, D with a
    root node, R and label it as "null';
    for every transaction, T ∈ D {
       // trim transaction database prior to building the FP-tree
       remove transactions that do not satisfy the constraint Cm
       reduce support value of all affected itemsets in CSL1
        and CSL2;.
     if the support value of an itemset is less than the threshold
     then {
       remove the item/itemset from CSL1 and CSL2;.
       remove items from T not present in CSL1 or CSL2;
       select and sort the items in T in the order CSL1;.
       recursively insert all items in T into the tree;
     }
    // now use the constraints to prune the FP tree during the
    // mining stage. D'- Trimmed Transaction Database;
    //FCI – list of frequent constraint satisfied itemsets
    To mine the constructed FP-tree call the function
     (Constrained FP Growth) as CFP(D', CSL2,, FCI, Cm ,Cam);
 }
```

Figure 6a: Phase 2 of FGC

```
Function CFP(DB, flist, FCI, Cm,, Cam)
Parameters: DB: conditional transaction database; FCI: set
                of frequent itemsets found so far
for every element ai in flist
{
   identify all transactions which contain ai as ai's
       conditional database;
   remove trans in ai's conditional database which do not
       satisfy the constraint Cm;
  flist ai = set of local frequent items in ais conditonal database
  E=flist ai ∪ {ai};
 // use the anti-monotone constraint to prune search space
 for all itemsets ai ∈ E such that supp(aiaj)≥min_supp and
      Cam(E) and j≠I
 {
       create a conditional database DBai aj;
       flist ai aj = set of all local frequent items in DBai aj;
       if Cm(ai aj) then add aiaj to FCI;
       call CFP(DBai aj,flistaiaj,FCI,Cm,Cam);
   }
 }
```

Figure 6b: Phase 3 of FGC

5. Experimental results

In our experimentation we used three synthetic datasets and one real-life dataset. All three datasets were generated using the [9] synthetic dataset generator. By varying three major parameters such as average size of the transaction, number of unique items and the number of transactions,

three different datasets (D2, D3 and D4) were generated. A real life dataset, (D1) supplied by an anonymous Belgian retail store was also used. There are 88,163 transactions with over 16,470 items. D2 is typical of those used in data mining and has been used in previous research [22, 14, 15, 23] for benchmarking performance. On the other hand, D3 is based on the study done by National Association of Merchandisers which discovered that, on an average, retail customers buy a maximum of two items per transaction (2000). Dataset D4 to test the scalability with respect to both number of transactions and number of unique items. The price information needed to define the constraints was generated using a Gaussian distribution. Table1 shows the parameters involved.

In section 5.1 we present the results of running FGC against the FP-bonsai algorithm at different support thresholds for the real-life dataset D1. We then go on to analyze the sensitivity of FGC on constraint type in section 5.2.

Table 1: Parameter settings for datasets used

Dataset	*Avg size of transaction*	No. of unique items	No. of transactions
D1	13	16K	88K
D2	25	10K	100K
D3	2	32K	640K
D4	25	32K	640K

5.1. Performance of FGC against FP-bonsai

To benchmark FGC against FP-bonsai we ran both algorithms on the real life dataset (D1) at two different support thresholds, 0.1% (low) and 10% (high). The 0.1% support threshold was chosen as this was the smallest level of support that enabled us to run the FP-bonsai algorithm in a reasonable amount of time; lower support levels caused an exponential increase in the timing for FP-bonsai. Constraint selectivity was varied in the range 10% to 90% and the mining time in milliseconds was measured for each algorithm. Figures 7 (a) and (b) illustrate the results for the low and high support thresholds respectively.

Figure 7 (a) illustrates that FGC clearly outperforms FP-bonsai throughout the selectivity range for both types of constraints. Both algorithms take advantage of selectivity, but FGC has the added advantage of pre-processing that trims the transaction dataset of items and pairs of items that do not meet the support and selectivity criteria prior to mining the FP-tree. Figure 7 (b) shows that the two algorithms perform very differently from each other at the higher support level. At this level of support FP-bonsai is virtually insensitive to constraint selectivity whereas there is a dramatic drop in mining time for FGC at just over 50% selectivity. This sharp drop in time can be explained by the Table 2 that shows that the number of FP-tree nodes halves at this selectivity value and that the number of 3-frequent constraint satisfied itemsets drops to zero at this point, which means that FGC terminates almost immediately when executing phases 2 and 3 of its mining process.

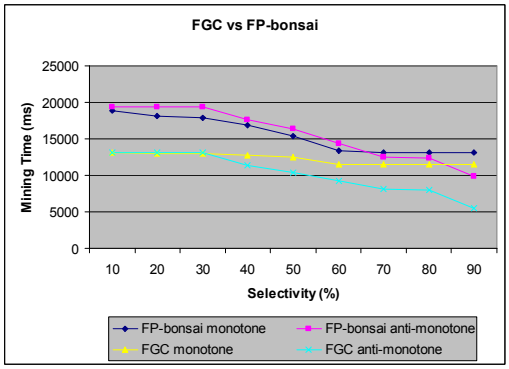

Figure 7 (a): mining at low support level (0.1%)

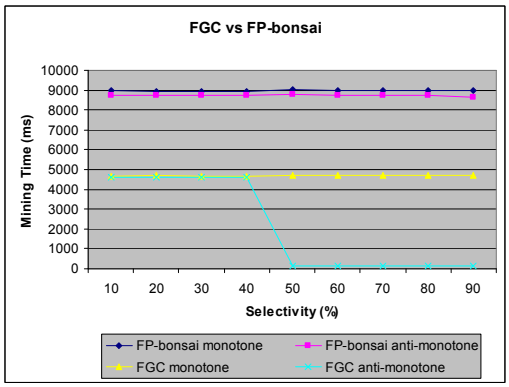

Figure 7(b): mining at high support level (10%)

Table 2 Variation in the number of FP-tree nodes

Constraint Selectivity	Number of FP-tree nodes	Number of 3-frequent itemsets
10%	10	25
20%	10	25
30%	10	25
40%	10	25
50%	5	0
60%	4	0
70%	2	0
80%	1	0
90%	1	0

5.2. Sensitivity of FGC on Constraint Type

Having established the superiority of FGC over FP-bonsai in the previous section we now turn our attention to examining its behavior with respect to the different constraint types. For each constraint type (e.g. monotone), we varied its selectivity in the range 10% to 90% while keeping the selectivity of the other constraint type (anti-monotone) fixed at 0%. This enabled us to measure the effect of each constraint type on mining time separately. In this group of experiments we used the synthetic datasets (D2, D3 and D4), in addition to the real-life dataset, D1.

As mentioned previously the two constraint types each have their own role to play and we were thus interested in isolating the effects of each constraint type on performance. Figures 8a to 8d illustrate the performance of FGC over the 4 different datasets that we tested.

The graphs show that both types of constraints are influential in reducing mining time across the selectivity range. This is true across all datasets tested. However, with D3, the smallest of the datasets, the reduction in mining time across the selectivity range was much smaller in proportion to the other three datasets. This is to be expected as the average transaction size is only 2 and a very large proportion of the frequent itemsets would be discovered during the pre-processing phase, which identifies the 2-frequent itemsets.

Another consistent trend is that both types of constraints perform equally well at low selectivity, but as the degree of selectivity increases the anti-monotone constraint starts to exert more influence over mining time. At higher degrees of selectivity, the anti-monotone constraint enables more aggressive pruning of branches of the SOTrieIT structure, resulting in smaller FP-trees that need to be built prior to mining in phases 2 and 3, unlike with the monotone constraint which does not contribute to such a reduction.

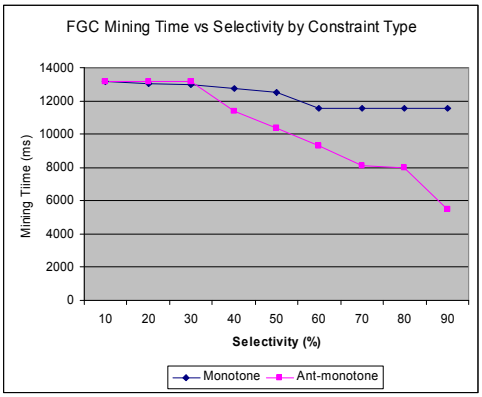

Figure 8a: FGC on dataset D1

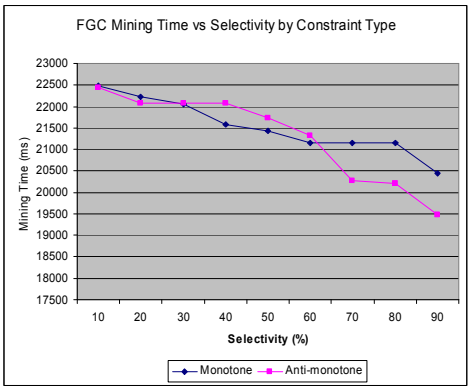

Figure 8b: FGC on dataset D2

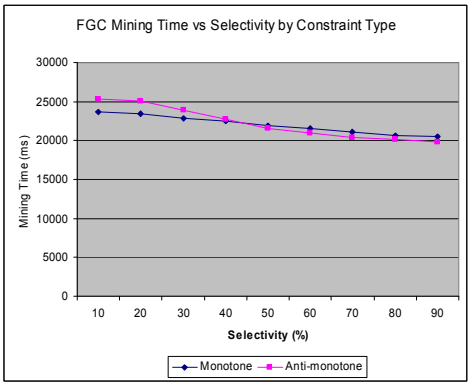

Figure 8c: FGC on dataset D3

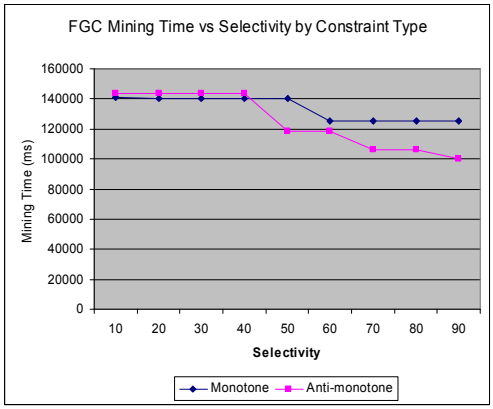

Figure 8d: FGC on dataset D4

6. Conclusion and future work

In this paper we have discussed an approach for efficiently embedding constraints into the mining process and evaluated its performance on both synthetic and real-

life datasets. We have shown that by pre-processing the transaction database and exploiting constraints, significantly better results than the current state-of-art algorithm can be obtained. We also systematically studied the effects of each constraint type on performance.

One possible direction for future work would be to extend the approach to other types of constraints, such as succinct and convertible constraints. Another possibility would be to incorporate constrained closed frequent itemsets into the FGC algorithm. The concept of closed frequent itemsets have been shown to be very effective on its own [7] and it would be interesting to test its effect in conjunction with pre-processing and constraint exploitation.

7. References

[1] (2000). Annual Survey Results, National Association of Recording Merchandisers.

[2] Agrawal, R., T. Imielinski, et al. (1993). "Mining association rules between sets of items in large databases." SIGMOD Record (ACM Special Interest Group on Management of Data) **22**(2): 207-216.

[3] Agrawal, R., H. Mannila, et al. (1996). Fast Discovery of Association Rules. Advances in Knowledge Discovery and Data Mining. U. M. Fayyad, G. Piatetsky-Shapiro and R. Uthurusamy, MIT Press.

[4] Bonchi, F., F. Giannotti, et al. (2003). ExAMiner: optimized level-wise frequent pattern mining with monotone constraints. Third IEEE International Conference on Data Mining, 2003(ICDM 2003).

[5] Bonchi, F., F. Giannotti, et al. (2005). "ExAnte: a preprocessing method for frequent-pattern mining." Intelligent Systems, IEEE [see also IEEE Intelligent Systems and Their Applications] 20(3): 25-31.

[6] Bonchi, F. and B. Goethals (2004). FP-Bonsai: The Art of Growing and Pruning Small FP-Trees.

[7] Bonchi, F. and C. Lucchese (2004). On closed constrained frequent pattern mining. Proceedings of the Fourth IEEE International Conference on Data Mining, 2004. ICDM 2004.

[8] Boulicaut, J.-F. and B. Jeudy (2000). Using constraints during set mining: Should we prune or not? In Actes des Seizime Journes Bases de Donnes Avances BDA '00, Blois, France.

[9] Cristofer, L. (2001). ARMiner Project. Boston.

[10] Das, A., Ng, W.-K., & Woon, Y.-K. (2001). Rapid association rule mining. Proceedings of the tenth international conference on Information and knowledge management (pp. 474-481). Atlanta, Georgia, USA ACM Press.

[11] Han, J. and J. Pei (2000). "Mining frequent patterns by pattern-growth: methodology and implications." SIGKDD Explorations. **2**(2): 14-20.

[12] Han, J., J. Pei, et al. (2000). Mining frequent patterns without candidate generation. Proceedings of the 2000 ACM SIGMOD international conference on Management of data, Dallas, Texas, United States, ACM Press.

[13] Hipp, J., U. Güntzer, et al. (2000). "Algorithms for association rule mining — a general survey and comparison." SIGKDD Explor. Newsl. **2**(1): 58-64.

[14] Jia, L., R.-Q. Pei, et al. (2003). Using constraint technology to mine frequent datasets. International Conference on Machine Learning and Cybernetics, 2003.

[15] Jia, L., R. Pei, et al. (2003). Tough constraint-based frequent closed itemsets mining Proceedings of the 2003 ACM symposium on Applied computing Melbourne, Florida ACM Press**:** 416-420

[16] Kosters, W. A., W. Pijls, et al. (2003). Complexity Analysis of Depth First and FP-Growth Implementations of APRIORI.

[17] Lakshmanan, L. V. S., C. K.-S. Leung, et al. (2003). "Efficient dynamic mining of constrained frequent sets." ACM Trans. Database Syst. 28(4): 337-389.

[18] Lakshmanan, L. V. S., R. Ng, et al. (1999). Optimization of constrained frequent set queries with 2-variable constraints. Proceedings of the 1999 ACM SIGMOD international conference on Management of data. Philadelphia, Pennsylvania, United States, ACM Press**:** 157-168.

[19] Park, J. S., M.-S. Chen, et al. (1997). "Using a hash-based method with transaction trimming for mining association rules." IEEE Transactions on Knowledge and Data Engineering **9**(5): 813-825.

[20] Pei, J. (2002). Pattern-growth methods for Frequent pattern mining. School Of Computing Science. Burnaby, British Columbia, Canada, Simon Fraser University**:** 147.

[21] Pei, J., J. Han, et al. (2004). "Mining sequential patterns by pattern-growth: the PrefixSpan approach." Knowledge and Data Engineering, IEEE Transactions on 16(11): 1424-1440.

[22] Woon, Y.-K., W.-K. Ng, et al. (2001). Fast online dynamic association rule mining. Proceedings of the Second International Conference on Web Information Systems Engineering, 2001.

[23] Woon, Y.-K., W.-K. Ng, et al. (2004). "A support-ordered trie for fast frequent itemset discovery." Knowledge and Data Engineering, IEEE Transactions on 16(7): 875-879.