CS6220 HW4: Comparison of Frequent Pattern Mining Algorithms

Introduction

The two FIM algorithms I have chosen to test/benchmark are:

1. Bodon's APRIORI implementation from FIMI03 [1] . The algorithm is implemented in C++ and requires that you pass in a minimum support threshold and has the option for passing in a minimum confidence threshold.
2. Borgelt's FIM implementation from FIMI03 [2]. The algorithm is implemented in C++

I decided to compare two different implementations of the APRIORI algorithm as it is the most commonly used FIM algorithm and has the potential for different implementations due to the liberal nature of the algorithm parameters (support and confidence threshold, choice of data structures including digraphs and tries, etc.)

The two benchmark datasets we chose were from the FIMI dataset repository [3]. They are:

1. **T10I4D100K**: A popular benchmark for FIM. The dataset consists of 100,000 synthetic examples generated by IBM Quest.
2. **Kosarak**: This is a dataset provided by Ferenc Bodon which consists of clickstream data of a Hungarian online news portal.

Experiments and Observations

All my experiments were run on a Macbook Pro with the following configurations:

Component	**Configuration**
Processor	2.6GHz 6-core Intel Core i7, Turbo Boost up to 4.3GHz, with 9MB shared L3 cache
RAM	16GB DDR4
Storage	256GB SSD
Graphics	Radeon Pro 560X with 4GB of GDDR5 memory and automatic graphics switching Intel UHD Graphics 630

Most of my experiments involve manipulating the *minimum support threshold* of the FIM algorithm implementation. The *minimum support threshold* is based on the *support* of an itemset, which is the number of transactions that contain that itemset. We say an itemset is *frequent* if the support of that itemset is greater than the minimum support threshold [1].

1. **Measuring the Running Time of the FIMI03 and FIMI04 implementations on the T10I4D100K dataset:**

The table below summarizes the running times of each FIM algorithm implementation at various minimum support thresholds:

Minimum Support Threshold	Bodon Running Time	Bodon Max Size Itemset	Borgelt Running Time	Borgelt Max Size Itemset
0.05	0.671s	3	4.146s	3
0.03	0.775s	3	9.992s	3
0.02	0.851s	3	28.048s	3
0.01	0.989s	4	170.23s	4

2. **Measuring the Running Time of the FIMI03 and FIMI04 implementations on the Kosarak dataset:**

The table below summarizes the running times of each FIM algorithm implementation at various minimum support thresholds:

Minimum Support Threshold	Bodon Running Time	Bodon Max Size Itemset	Borgelt Implementation	Borgelt Max Size Itemset
0.05	5.772s	5	49.738s	5
0.03	5.781s	5	61.144s	5
0.02	5.947s	6	129.70s	6
0.01	6.349s	6	284.17s	6

From the table, it is clear that *the running time of both FIM algorithms increases as we decrease the minimum support threshold.* This is pretty self-explanatory: the minimum support threshold intuitively denotes how strict the algorithm is with each itemset's support value. A larger threshold means that the algorithm is more liberal with what is considered an acceptable support value for a *frequent* itemset. Lower thresholds are more conservative with what is included in the *frequent* itemset. Also, the Bodon and Borgelt implementations produce drastically different results or the same algorithm: the Borgelt has a much larger max item set size, which means that the implementation treats the support thresholds differently when compared to the Bodon implementation.

The Borgelt implementation also takes significantly longer as we decrease the support threshold. This suggests that the Bodon implementation is much more efficient in terms of finding itemsets of different sizes for a specified support threshold. This could be due to the following reasons:

1. The Bodon implementation makes excellent use of hashing techniques when it comes to retrieving leaf values (as each of them represent an itemset), while the Borgelt implementation manually looks up leaf nodes without a hash table of any sorts.
2. The Borgelt implementation relies heavily on transaction representation and transaction filtering. While these techniques attempt to reduce the running time, they add a heavy memory overhead to the algorithm.

Overall, the Bodon implementation makes excellent use of some of the properties encountered in the FIM problem. The Borgelt implementation also provides a competent implementation but fails to exploit said properties. The APRIORI algorithm seems to be quite scalable to finding itemsets of different sizes for large datasets. It provides an excellent framework with support and confidence threshold that allows the algorithm to discriminate frequent itemsets from the rest.

References

1. Ferenc Bodon. A fast APRIORI implementation. Informatics Laboratory, Computer and Automation Research Institute, Hungarian Academy of Sciences.
2. Christian Borgelt, Efficient Implementations of Apriori and Eclat.
3. Frequent Itemset Mining Datasets Repository. http://fimi.uantwerpen.be/data/