Alex Reibman
CS 378

Results

This implementation of the Apriori-DHP algorithm is by far one of the fastest results possible. Using T10I4D100K.dat and a minimum support count of 500, the program can be completed in just less than 18 seconds. At higher support counts, there are even faster results. From word of ear, this is one of the fastest frequent itemset miners in the class.

Algorithmic optimizations used

The central idea I used was an idea called Direct Hashing and Pruning, DHP. DHP is a variation of the Apriori algorithm, but it implements a hashing method to filter unnecessary itemsets before generating the next candidate itemsets. This implementation is dramatically faster than the traditional Apriori method. However, I took it a step forward. DHP is problematic because it can result in false positives and make the process slower. To avoid this, I used a variation of DHP called Perfect Hashing and Pruning, PHP.

PHP is more efficient than DHP because it reduces the amount of data scanned. It also reduces the number of transactions scanned by pruning the database for every run through. PHP works by creating a hash table of each item in the database. Because it's a hashmap, we can easily add to it. The hashmap has the item id as the key and the count of times it is found in the database as its value. Every time I add an item I increment its value.

This generates all of the frequent 1 sized itemsets. I prune all items that are less than the support count. For 2 sized itemsets, I have to find the combinations of each with a combination generating algorithm. With these combinations, I store all of the items found within the combinations and run through the database again. If any items are left in the database that are not found in the collection of items used to form the combinations, I delete the items from the database. I repeat this until there's nothing left in the database.

For the most part, I relied on 3 primary data structures: ArrayLists, Hashmaps, and ConcurrentSkipListMaps. ArrayLists were used for storing small data that only needed iterations. Hashmaps were used to store data as well as ConcurrentSkipListMaps. ConcurrentSkipListMaps were probably the fastest data structure to use because when modifying the database (formatted with ConcurrentSkipListMaps), Add and Remove only take O(logn) time.

For this implementation, I took inspiration from S. Ozel and H. Guvenir. My implementation is not exactly like theirs (mainly because I had a very difficult time understanding their paper), but it gave me a lot of good ideas on how to implement the algorithm.

The process and lessons

This was a very daunting process. Considering the fact that this used 2 of my late passes plus is 2 days late, I can’t say this was easy. If I had to do this again, I would have started alone instead of as a group mainly because I learned that I can do a really good job interpreting papers in an implementable way.

The most important thing I took away from this project is the magic of hashmaps. These things are fast. Really fast. Additionally, I now know how to use ConncurrentSkipListMaps, which only have a O(logn) Add/removal/get time. This is ridiculously fast.