CS470 Homework 2

Matthew Chau

February 20, 2020

Collaboration Statement

For this assignment, I have not consulted or asked for help from anyone for the completion of this assignment.

Runtime Data

T10I4D100K.txt

Dataset	*T10I4D100K.txt*					
Minimum Support	500	750	1000	1250	1500	1750
Total Lines of Data	100000					
Ratio	0.005	0.0075	0.01	0.0125	0.015	0.0175
Runtime (s)	12.26	0473852	6.18	5.72	3.53	3.14
Number of Frequent Sets	1073	561	385	310	237	186

retail.txt

Dataset	*retail.txt*					
Minimum Support	500	750	1000	1250	1500	1750
Total Lines of Data	88162					
Ratio	0.0057	0.0085	0.0113	0.0142	0.0170	0.0198
Runtime (s)	5.65	4.12	3.41	2.94	2.72	2.56
Number of Frequent Sets	468	219	135	91	68	56

Above are the data of the execution of the Apriori Algorithm on two different datasets: T10I4D100K.txt, which consists of 100,000 different transactions, and retail.txt, which consists of 88162 different transactions. The minimum support are hold the same to compare the different runtime.

From Figure 1 below, we notice that both curves have downward slope with inward curves, and the execution time for retail.txt is much shorter than T10I4D100K.txt, around half of the runtime when the minsup ratio is around 0.005. This also indicates that the runtime tend to have a positive relationship with the number of transactions in a file.

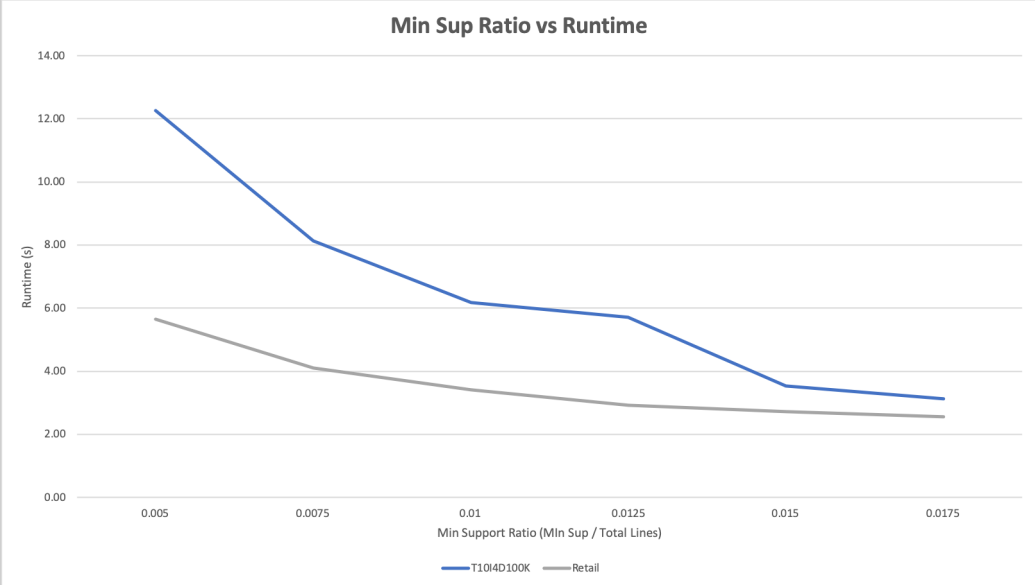

Figure 1: Figure 1

Optimization Methods

When designing this algorithm, I have used several optimizing methods to boost the running speed. There are two data containers: subset_mapping, a dictionary that maps rows from the data set to the set of all length-k candidate frequent subsets; freq_count, a dictionary that maps each frequent set to its number of occurrence in the data set.

First of all, the data set is entered into subset_mapping, a dict in python with average cost of O(1) in query, insert, and remove. The benefit of using a dynamic data set is to delete redundant rows that cannot generate new candidates, so that the iteration of the data set would be much quicker as the size of the data decreases.

Secondly, I use a temporary freq_count_local to store a set of subsets of length k at kth iteration to avoid iterating all subsets from length 1 to k. Then the frequency count in freq_count_local is added to the global freq_count. This reduces O(size of dataset) operation to O(number of length k frequent subsets), which makes it more efficient.

Experience and lesson learned

While doing this assignment, I have realized how efficiency of algorithms is, and have gained much experience on manipulating with large data sets. At first, my algorithm runs tremendously slow - around 35 minutes with incorrect results. After pruning and some optimization, it is reduced to around 18 minutes; and later on 5 minutes. With more and more optimization, as lots of loops being deleted, redundant data being removed, better data structures being used, it's finally maintained at around 11 seconds, and I believe it can still be further optimized.

Also, when debugging for mistakes, I have learned to divide the problem into small sub-problems. For example, I have reduced the problem to 1. Generate length 1 candidate subsets, 2. Generate length 2 candidate subsets, and so on. After generating all candidates, I generalized and added a while loop that loops until no more candidates can be generated.

Making the data set smaller is also a good way to check the accuracy of the algorithm in a much shorter time. To debug, I have manually added a couple rows of data so that I could do human calculations on the data set, while comparing to the results of the algorithm. Printing between different sections is also a good way to find out where the error is located exactly. It helped me find a lot of errors in my code, and gave me a clear idea of what the variables look like at a particular moment.