COL761-Assignment 1

Rajat Bhardwaj - 2020CS50436
Rishita Agrawal - 2020CS50439
Manshi Sagar - 2020CS50429

January 2024

1 Optimizations in FP Tree construction and Pattern Mining

We used the basic FP Tree implementation from some GitHub repositories and did a few optimisations as follows:

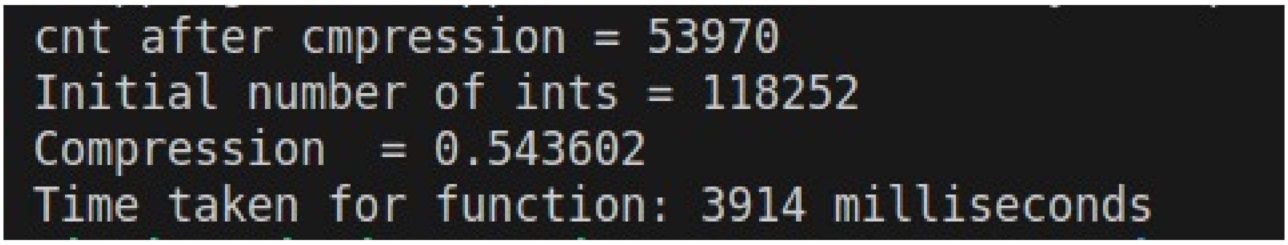
1. We discard singleton patterns, as they won't help in compression.
2. We have applied a filter to remove pattern such as AB : 40 + s, ABC : 40 are both in the frequent patterns, we only consider ABC : 40 and remove AB : 40 and compare s with the support to check if AB : s is frequent or not. Since after ABC : 40 is substituted, AB : 40 pattern is useless.

2 Compression Algorithm

```
1. Starting support = 0.9*numTransactions
2. Mine frequentPatterns for this support value
3. if $\sum$ frequentPattern.size()*frequentPattern.frequency() < bound:
       support <= 0.6*support , go to step 2
4. Sort frequentPatterns according to frequentPattern.frequency()*frequentPattern.size()
5. For every pattern $\in$ frequentPatterns:
       if pattern does not exist in compressMap:
           assign a key to pattern and add it in compressMap
6. For every transaction $\in$ transactions:
       For every pattern $\in$ frequentPatterns:
           if pattern $\in$ transaction:
               substitute pattern for its key
7. support <= 0.7*support, if support > 0.001*numTransactions: go to step 2
```

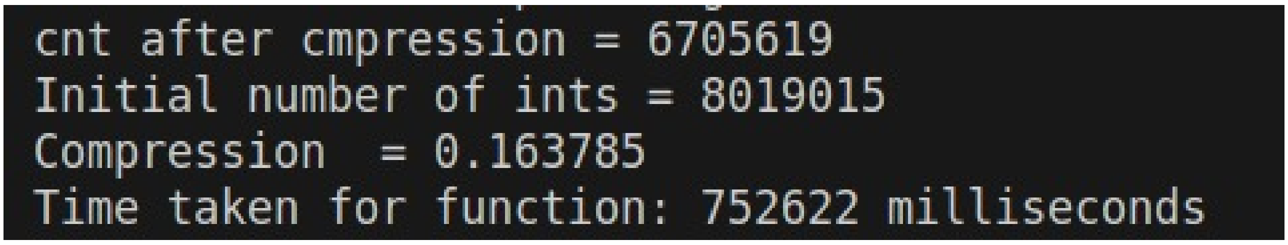
2.1 Optimization Techniques

1. **Mining and compressing in decreasing order of supports**
   In order to maximise compression, we compress those patterns with maximum frequency the earliest, ie, we first mine patterns using a high support value, compress transactions using these patterns and then decrement support by a factor of 0.7.
2. **Setting lower bound**
   After we mine patterns, we compare $\sum$frequentPattern[i].size()*frequentPatterns.frequency() with a lower bound and compress only if the quantity exceeds the *lowerBound*. This quantity tentatively represents the number of replacements that will be done if we compress using these patterns, so, we only compress in we expect good number of replacements, ie, good compression.
3. **Setting upper bound**
   If we get a large number of patterns, we first sort them according to their size and frequency (in Step 4) and take only the best *upperBound* number of patterns.



```
cnt after cmpression = 53970
Initial number of ints = 118252
Compression  = 0.543602
Time taken for function: 3914 milliseconds
```

Figure 1: Result of D_small.dat on local machine



```
cnt after cmpression = 6705619
Initial number of ints = 8019015
Compression  = 0.163785
Time taken for function: 752622 milliseconds
```

Figure 2: Result of D_medium.dat on local machine

3 Results

.	D_small.dat	D_medium.dat	D_medium2.dat	D_large.dat
Compression	54.63 %	Row 2, Cell 3	40.5%	8%
Time	3914 ms	Row 3, Cell 3	1212 sec	1 hr approx.

- D_small.dat file shoes maximum compression in minimum time. This is because support has a factor of the number of transactions, which is small in this file, so, we are able to mine patterns even with single digit frequency
- We observed more compression in D_medium2.dat as compared to D_medium.dat, also due to the same reason that D_medium2.dat has less number of transactions.
- We observed that despite having more size, D_medium.dat was able to mine more patterns for a smaller support value as compared to D_medium2.dat
- Compression and time consumption increases with more number of transactions
- HPC is not working at the time of submission, so we are not attaching screenshots of D_medium2.dat and D_large.dat. We are writing the results based on previous runs on HPC.

4 Contribution

MEMBER 1 :
Name - RAJAT BHARDWAJ
Entry number - 2020CS50436
CONTRIBUTION - 33.33 %

MEMBER 2 :
Name - RISHITA AGRAWAL
Entry number - 2020CS50439
CONTRIBUTION - 33.33 %

MEMBER 3 :
Nams - MANSHI SAGAR
Entry number - 2020CS50429
Contribution - 33.33 %