A1 – COL761 - Transactional Data Compression

Team Details:

Team Name: Data_Voyagers

Github link: https://github.com/rajasekhar108/data_voyagers

Members:

Name	Entry Number	Contribution
Bogam Sai Prabhath	2023AIB2079	1/3
Mikshu Bhatt	2023AIB2067	1/3
Nallavadla Rajasekhar Reddy	2023AIB2066	1/3

1) FPTree growth will be run from the file <*fpattern.cpp*> and frequent patterns will be generated and saved in the <*frq.txt*> *file*
2) The Frequent patterns generated in the above will be used and mapped using Alpha-special characters for each frequent pattern used for mapping.
3) The frequent pattern mappings and the replaced frequent itemsets in the original transactional data will be stored in the respective file entered by the user.
4) After decompressing the file using <*interface.sh*> the decompressed file will be generated by using the data from the output file generated in the step 3.

Abbreviations:

tl : Top length of the frequent pattern generated

wl : weighted average length of the frequent patterns generated

l : length of the frequent pattern currently under consideration

Algorithm used:

FP-Tree: The Frequent Pattern (FP) tree algorithm generates all the frequent itemsets or patterns without generating candidates using a threshold supports of 0.8%,12% for medium and large files which were found out to be optimal on trial-and-error basis. The supports for small and lower medium files were calculated based on the median length of the patterns. The FP tree code used here has been taken from this source[1]. The FP tree generates frequent patterns without generating candidates and scans the entire database only once. Then Conditional FP trees will be generated in each iteration. The frequent patterns generated will vary depending on the dataset used. So, for a optimum solution for varying datasets an algorithm for the support is generated which is sorting all the frequent patterns and taking the median frequent pattern's frequency and dividing it by number of transaction. This will help algorithm in removing items with frequency less than support counts.

Compression: The Frequent patterns generated from the *fpattern.cpp* are used and few of them will be mapped by using "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ`~!@#$%^&*()_-+={}[];':,./<>?". The alpha (small & Capital) & special characters are taken from this string. The length of the string used while mapping depends on number of the total frequent patterns used after generation and the length = ($\log_{82}$ n)+1, where n denotes the

number of frequent patterns used for mapping and 82 denotes the number of characters present in the string above. The length is calculated in the function *int lg()*. These strings are generated randomly using string *generateRandomString ()* method used in the code.

Only a few of the frequent patterns generated are used. The frequent patterns filtered for mapping are based on subset checking and also filtered based on the length of frequent patterns generated. The filtering of frequent patterns based on the length of the frequent patterns follows the following procedure:

- The lengths and the frequencies (number of frequent patterns of each length) of the generated frequent patterns are then calculated and weighted average length of the frequent patterns is calculated.
- The frequent patterns of length < weighted average are discarded and will not be considered for replacement in the transaction database file.
- In the frequent patterns of length > weighted average the number of frequent patterns selected for replacement is selected on the criteria that (i) all (100%) of the top length (tl) frequent patterns will be used , (ii) only 10% of the frequent patterns of length = weighted average length (wl) will be used, (iii) In between tl and wl for each decreasing length only 100-((tl-l)*(90/(tl-wl)))% of frequent patterns generated for that length will be used, where l denotes the length of the frequent pattern currently in use.

As the number of frequent patterns available for use will be high for small and medium transactional files after this, there will be a need for further more filtering of frequent itemsets. This is done using subset checking of frequent patterns. It implies that all the frequent itemsets/patterns till k-1 length will be checked with frequent itemsets/patterns of length k, if the former are found to be a subset of later then they will be pruned out and will not be used for mapping. This process continues till the least length of frequent patterns is reached. After following the above 2 steps of frequent patterns or maps selection criteria, the string codes and their respective frequent patterns will be stored in the map. Only these mapped frequent patterns along with their codes will be used for further steps in the compression process.

These strings/codes are then run through the entire transactional database and the string or code will be replaced in the main transaction database if a particular pattern is found in a transaction. This is done in the method *string encodeeachtxn ().* The unused frequent patterns will be deleted from the map.

The mapped frequent patterns and the replaced transactions along with the string codes are all stored in the respective file entered by the user. The frequent patterns map and the encoded data both will be stored in the same file and they are differentiated by using unique string name “1mapping_starts1”.

Reconstruction of Data: Compressed Transaction database which is available in the file as entered in the Compression process step by the user will be scanned and frequent patterns map and compressed data are differentiated using the above unique string “1mapping_starts1”. If there exists a string code/(s) in any transaction in the compressed data then those codes will be replaced by the respective frequent pattern which was stored in the map. This is done using the functions *void create_mapping()* and *void decode()*. The reconstructed data will be then stored in the file as entered by the user after running the <*interface.sh*> file.

Final Report or Observation: The Process of Compression and Reconstruction of data used in the above algorithm is completely lossless. The time taken by the algorithm depends on the size of data and also the type of data, for varying supports we get varying compression ratios and varying run times for different size and type of data. For optimal solution to compress all types of data optimal approach for support and filtering frequent patterns are used.

Sources:

[1]. https://github.com/shubhamguptaiitd/frequent_pattern_mining/blob/master/fptree.cpp