CMPUT:692 Assignment 2: Report

Nawshad Farruque

Experimental Setup:

For data, I have written a program that takes plain text and converts it to inverted index or term posting list. New posting list can be created by pasting blob of texts in RealData.txt file and then running the program. The output posting list is saved at data.txt file. Also, I have used the data that Sanket shared and mixed a large portion of that data with my own data to create DataFile1.txt for experiments. I have implemented both TopK [1] and Naive Document at a time (DAAT) algorithms for calculating cost savings posted in the following tables. Here average savings is equivalent to number of times document have gone through full evaluation process for both the algorithms, thats why the name “Full Evaluations Savings” has been used instead in the tables. Full evaluation happen much less in TopK, which already indicates that run time would be low when working on huge data set. I used Naive DAAT as a baseline algorithm for experimental evaluations of savings. To play with skip pointer and also upper bound I have implemented the options to specify them in the command line.

Experimental Results :

Number of queries tried: 40. Also list of terms obtained from from DataFile1.txt. This data file has 294 unique documents and 381 unique terms.

The query terms of varied length are built from serially chunking the term list in equal length of sublists from DataFile1.txt. For experimenting I wrote my own analysis program that evaluates both algorithms varying different parameters.

Average Full Evaluation Saving(%) Formula = ((no of times full evaluation process invoked in DAAT - no of times full evaluation process invoked in TopK)/ no of times full evaluation process invoked in DAAT)*100, further divided by the number of queries tried for average cost savings.

Average Computation Time Savings(%) Formula = ((computation time in DAAT - computation time in TopK)/ computation time in DAAT)*100, further divided by the number of queries tried for average computation time savings.

Table 1: Average Full Evaluations and Computation Time Savings based on Varied Query Length with K = 6

Query Length	Average Full Evaluation Saving(%)	Average Computation Time Savings (%)
3	33.5	62.2
5	41	78.52
7	48.8	81.09
9	55.3	84.25

Table 2: Average Full Evaluations Savings based on Varied K and Query Length

Value of k	Query length=2(Average Full Evaluation Savings %)	Query length=4(Average Full Evaluation Savings %)	Query length=6(Average Full Evaluation Savings %)
2	72.59	78.29	81.22
4	39.2	53.33	56.92
6	25.98	39.24	44.6

Table 3: Average Computation Time Savings based on Varied K and Query Length

Value of k	Query length=2(Average Computation Time Savings)	Query length=4(Average Computation Time Savings %)	Query lengt6=4(Average Computation Time Savings %)
2	75.7	72.48	81.28
4	71.4	78.8	81.44
6	67	75	73.72

Table 4: Average Full Evaluations Savings based on posting Length for k=6 and query length = 5

Average posting size = 5	Average posting size = 6	Average posting size = 7	Average posting size = 8
48	50	68	50

Table 5: Affect of Upper Bound Factor in Savings for k = 5 Query Length = 5 and Same Query:
query example: roll commission fast exce uncertainti

Upper Bound Factor	Full Evaluations Savings (%)
1	48
4	8
6	8

Discussion:

- In TopK algorithm, with the increase of query length the amount of documents that goes for full evaluation does not grow like Naive DAAT, thats the reason we see there is savings gain(both in time and evaluation) with increased length of queries.
- We can have the term upper bound multiplied by some constant. With the increasing of that constant value the cost savings decreases (see Table:4) and vice versa, this is also mentioned in [1]. In the program please use -c flag followed by an integer (6th argument) for setting up that constant value. This happens because the more big is the C the more documents are considered for full evaluation thus cost saving is less.
- We can move the pointers by some amount of skips. This changing in skip eventually contributes to less computation but with false negative results. In the program please use -s flag followed by the number of skips you want to make(4th argument).

How to run the program:

Inside the TopKQuery folder issue the following command: java -jar dist/TopKQuery.jar <file name> -k <K> -s<No. of Skips> -c <UB Constant>
Alternativley you can run java -classpath "dist/TopKQuery.jar:dist/lib/*:" topkquery.TopKQuery <file name> -k <K> -s<No. of Skips> -c <UB Constant>
You have to atleast mention the file name that is the first argument to run the program. If you want to use 6th argument you must ensure to provide all preciding arguments too.

Example: java -jar dist/TopKQuery.jar data.txt -k 5, will return topk documents and their scores. Then provide the terms as: t1 t2 t3 t4 etc and press enter for output.

Reference:

[1] Andrei Z. Broder, David Carmel, Michael Herscovici, Aya Soffer, and Jason Zien. 2003. Efficient query evaluation using a two-level retrieval process. In Proceedings of the twelfth international conference on Information and knowledge management (CIKM '03). ACM, New York, NY, USA, 426-434. DOI=http://dx.doi.org/10.1145/956863.956944