Emre Can Kucukoglu
eckucukoglu@gmail.com
https://github.com/eckucukoglu/daat-query-processing

Document-at-a-time query processing with WAND optimization

i. Implementation Details

As an implementation language, Java is preferred. Since critical count values are known, fixed sized array is used for data structure. Even though hash can be better option because of its low cost search operations (actually another languages like python provides easy-to-use library for hash implementation), binary search operations (O(log n)), can be implemented since both postings list and dictionary are already sorted order.

Main classes are; Dictionary, Document, Posting, Query, QueryProcessor, Term. Moreover, min-heap implementation of Morten Silcowitz is used in Heap class. To not to mixed up file operations with others, Parser class also has the static, data file specific functions.

Because of problem's nature, processing has some bottlenecks. These are mainly caused by:

- File operations,
- Memory allocations,
- Run time,

Although effectiveness of these issues are not concerned while comparing document-at-a-time approch with WAND optimization, WAND's positive impact will be mentioned in conclusion part.

First bottleneck has occured in run time, because of the sequential search to find matching terms in dictionary for a given query. To pre-compute the postings index of a term, this sequential traverse is also used while matching terms. Essential bottleneck caused by file operations are occured while reading postings from postings binary file. Note that any term can have more than 500.000 postings. Thanks to Java garbage collector and pointer reference based implementation, even with huge inputs (dictionary: 3560655, documents: 2236050, queries:500, postings: 62292689) required memory can be allocated in 8GB RAM, Windows 8 OS computer.

While processing document-at-a-time approach, document indexer is computed with binary search operation in postings list. Furthermore, after deciding document id, again binary search operation is used in postings list to compute score.

For WAND optimization, for each term, its one-third of IDF value is taken as an upper-bound. For each document, if computed WAND threshold is more than score threshold, then compute this document's cosine similarity, add the heap, set score threshold with heap's k's document's score. Otherwise, skip this document.

Critical count values should be set in Parser class. These values are;

- number of terms in the dictionary,
- number of queries,
- number of documents
- data directory,
- document vector lengths data file name,
- query data file name,
- dictionary (term list) data file name,
- postings binary data file.

ii. Test Results

During document-at-a-time query processing for a given dataset (dictionary: 3560655, documents: 2236050, queries:500, postings: 62292689), average EvalPostingNo (the number of posting list elements that are used in the score computationevaluated postings) is 197393.044 (about 200k).

Since disjunctive semantics are assumed, this value can not change according to k value. (k is maximum heap size)

After WAND optimization, with heap size 2, average EvalPostingNo is 127517.29 (about 127k), average NonEvalPostingNo is 68532.148 (about 68k). With heap size 10, average EvalPostingNo is 160267.998 (about 160k), average NonEvalPostingNo is 36912.114 (about 37k).

iii. Conclusion

As seen in Figure 1 and Figure 2, with WAND optimization, almost **35%** and **19%** of document score calculations are omitted (respectively for k=2 and k=10). As a consequences of this improvements, while processing WAND optimization, memory usage decreased dramatically, i.e. 84%. Nothing changed in file operations, however implementation can be improved to reduce postings binary data file read operations. Even though, WAND implementation has threshold computation, run time of processing is also decreased.

iv. Figures

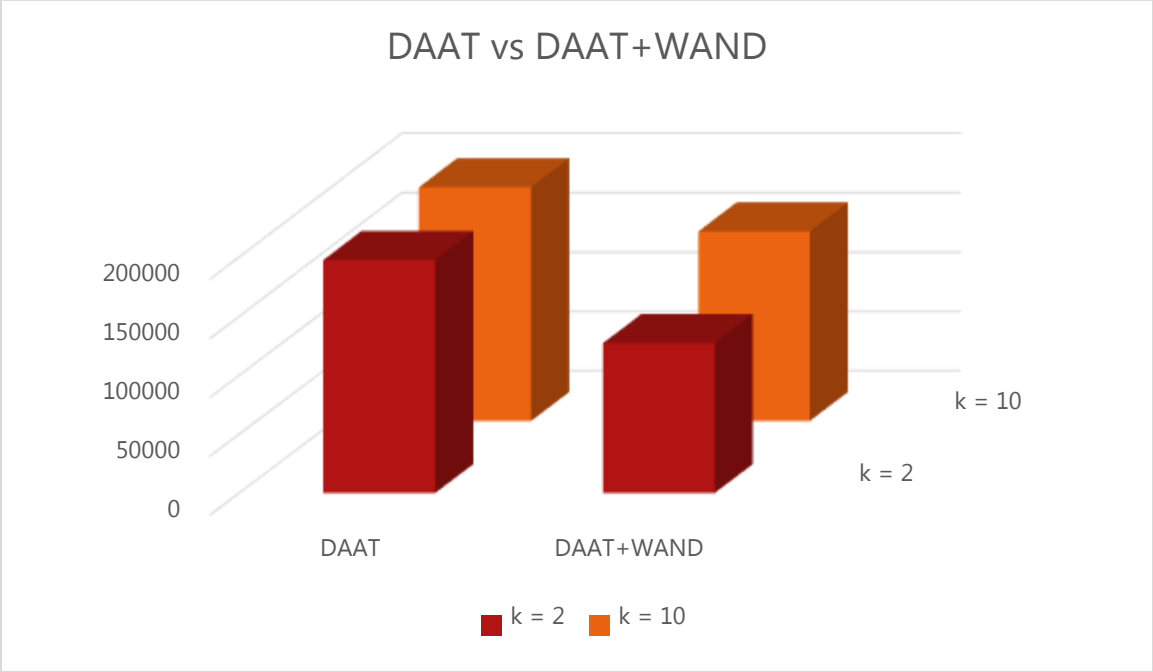

Figure 1: Comparison of average evalpostingno for simple DAAT and WAND optimization

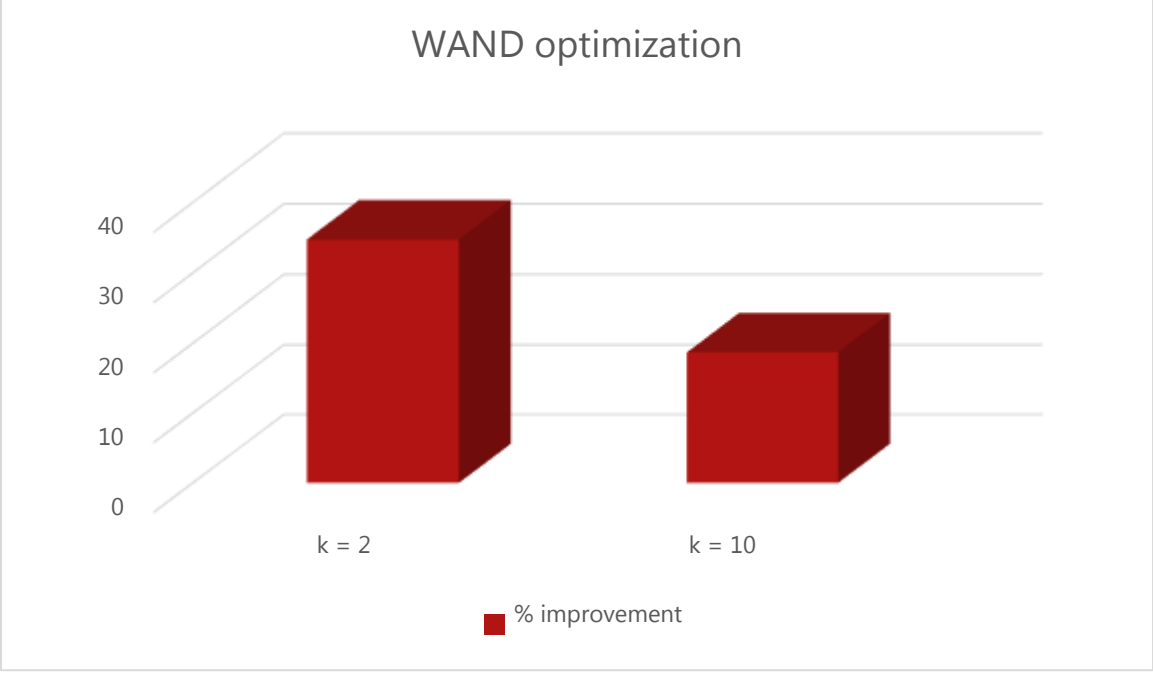

Figure 2: Percentage of improvement for WAND optimization