Pattern Searching via Indexing Strategies

Siddharth Malhotra
The University of Melbourne
smalhotra1@student.unimelb.edu.au

ABSTRACT

In this report, we discuss several indexing methodologies; such as Inverted indexing: In-memory, Merge-based inversion, hybrid approach for maintaining inverted indexes; Next-word indexing including its construction and compaction techniques; Suffix Array and BWT for optimising large scale pattern search. Further, we perform a critical analysis of these techniques.

KEYWORDS

Pattern Searching; Information Retrieval; Indexing

1 INTRODUCTION

Pattern search problem: Given that text $T[0 \ldots n-1]$ exists over some alphabet of size $\sigma = |\Sigma|$ and a pattern $P[0 \ldots m-1]$, locate the occurrences of P in T. *grep* is one of the simplest forms of document retrieval and allows wildcard marching through regular expressions. However, it may not be ideal for large documents, ranked retrieval and flexible matching operations.

Design of a search engine should include:

- Query resolution effectiveness
- Query term proximity
- Fast resolution of queries
- Minimal usage of hardware resources
- Scalability in terms of data
- Accommodate changes within documents
- Features such as Boolean restriction and phrase querying

Indexing strategies are data structures that map terms to the documents containing them for faster query retrieval. Inverse document frequency and term frequency statistics may be used to measure the how important a term is to a document.

2 INDEXING TECHNIQUES

2.1 Inverted Index Data Structure

`Zobel and Moffat [7]` define inverted lists as a collection which consists of a sequence of pairs of document numbers and in-document frequencies $(d, f_{d,t})$ pair, potentially augmented by word positions. Indexed terms may be stored sequentially or as a sequence of blocks that are linked. Some of the implications here would be: Accessing a sequence of blocks located randomly on disk would impose significant costs and the inverted list would be as many times the size. Second, if the invested listed is less than a kilobyte, there would be constraints on the feasible block size. Third, index update should be able to manage variable length fragments. Compression representation may allow large gains of performance.

Techniques for construction of index based on inverted lists:

- **In-Memory Inversion:** First collect term frequency information for the inverted list. Then, it places pointers into correct positions. The advantage of this approach is that no memory is wasted due to the random access capabilities of the main memory. This technique is only viable when main memory is 10-20% greater than size of index and vocabulary that are to be made.
- **Merge-Based Inversion:** Documents are read and indexed in memory until a fixed capacity is reached. Each inverted list needs to be represented in a structure that can grow as a dynamically resizable array about the term which is encountered. When memory is full, the index is flushed to disk as a single run with the inverted lists in the run stored in lexicographic order to facilitate subsequent merging.

Hybrid Approach to Index Maintenance:. Büttcher and Clarke [3] introduced a novel index maintenance stratergy, in which long lists are modified in-place while, short lists are maintained using merge-based update method. This would help leverage the best of the both strategies and give an optimal performance when compared to either in-place or merge-based alone.

- **In-Place + Immediate Merge:** Merged on-disk index is created with in-memory data and existing on-disk index. For terms encountered from long list during merge, postings are appended to a file containing postings for the term instead of new index.Short list follow the standard immediate merge strategy.
- **In-Place + Logarithmic Merge:** In logarithmic merge, a series of indexes is maintained, each twice as large as the previous one. Smallest A_0 are kept in memory and larger ones $(I_0, I_1, \ldots)$ on-disk. If A_0 gets too big $(>n)$, it is written to disk as I_0 or merged with A_0 (if I_0 already exists) as A_1.

2.2 Phrase querying/phrase completion using *NextWord* Indexing

Considering users combine querying with browsing along with techniques such as assisted query refinement Agosti and Smeaton [1], *next-word* indexing is proposed by Williams et al. [6] . Next-word index can, for each distinct word, identify all possible successor words and list locations at which they exist in the text database. B-Tree structure is used for storing such vocabularies and nextwords are sorted and stored contiguously. Further front-coding is used;

4 acne 7 acolyte 5 acorn 6 acorns 8 acoustic,

can be replaced by,

4,0 acne 5,2 olyte 3,3 rn 1,5 s 5,3 ustic,

in which the first number is the length of the stored string and the second is the number of characters shared with the previous string.

COMP90044-Research Methods, University of Melbourne, 2017
2017. 123-4567-24-567/08/06…$15.00
DOI: 10.1145/000000.000000

Type	Inverted Index	Nextword Index	Suffix-Based
Indexing Cost	Moderate	Higher	High
Query-based queries	Fast	Faster	Very Fast
Word-based queries	Moderate	Fast	Very Fast
Character-based patterns	Slow	Slow	Very fast
Space requirement	Moderate	Higher	High
Dynamic	Yes	Yes	No

Table 1: Performance comparison of Indexing Strategies

Locations of next word indexes are sorted and compressed with techniques used for standard inverted indexes. Storage of location lists in a concatenated contiguous manner implies that information concerning index terms can be found in only two disk accesses. Contiguous storage of variable length greatly reduces cost when compared with separately-stored blocks.

Compaction Techniques for *NextWord Indexes.* Bahle et al. [2] techniques for compression reduce space requirements and query response times. They define offset as follows, document d is a sequence of word occurrences $w_1 ... w_n$. Each word w_i is labelled with an offset o_i.Offsets are positive integers. Based on the fact that relative word positions are required for phrase indexes.

- **Compound Indicators** Size of the index could be reduced with alternative offset and indicator schemes. Distinct occurrences of terms are explored. W-distinct occurrences if for 1 to n the offset z_i=c for the cth occurrence of word w_i in d. An indicator i for pair p . s at position k in document d is compound if $i = z_k$.$z_{k+1}$is a pair of integers. to evaluate the query each indicator b'= z_k .$z_{k+1} \in B_L$ where b = z_{k-1} .$z_k \in B_L$, add b'to $B_{t'}$.
- **Offsets based on repeated words** For phrase query evaluation, offsets that allow differentiation between repeat occurrences of the same word are sufficient. For each occurrence of word, the corresponding count is recorded. P-distinct offsets offer savings in storage requirements when compared to ordinal offsets. They are sufficient to allow unambiguous left-to-right evaluation of queries.

2.3 Large-Scale Pattern using Suffix Array

If performed sequentially the time complexity of pattern search is $O(n + m)$. For k occurrences of P in T, the suffix array index takes $O(m + \log n + k)$ time and $O(n \log n)$ bits of space in addition to T.

Suffix arrays (lexicographic ordering of the n suffixes) serve as an effective method for in-memory pattern searching and counting substrings. However, to analyse large amounts of data, for instance, DNA sequencing; data structures requiring less space are utilised. Wavelet tree (FM-index) of the Burrows-Wheeler sequence is one such approach allowing backward search technique.

The BWT may be constructed either directly or by first constructing the suffix array and then deriving the BWT in linear time from it. Drawback is the construction of Suffix Array requires at least 5n bytes of main memory. Some analysis require Longest Common Prefix-array. LCP-array is defined by $LCP[1] = -1$, $LCP[n+1] = -1$, and $LCP[i]$=—lcp(SSA[i-1], SSA[i])— for $2 \leq i \leq n$, where $lcp(u, v)$ for strings u and v. They all first construct the suffix array and then obtain the LCP-array in linear time from it. Drawback are similar to BWT.

FM-Index, based on BWT allows pattern search in $O(m \log \sigma)$ time and space proportional to original text. However, it could lead to random m disk access across memory when T is too large for the main memory. Sinha et al. [5] introduced the LOF-SA which merges suffix and LCP arrays in an interleaving manner and extracts necessary sequences from text to fill in the fringe characters.

Gog et al. [4] introduced an efficient mechanism for exploiting whole block reductions and describe a condensed BWT mechanism for storing and searching the string labels of a pruned suffix tree. Two major changes made are made, introduction to improvements to LOF-SA and use of condensed BWT structure.

Reducible Blocks

- Suffix pointers were eliminated for block reduction.
- Non-singleton irreducible blocks were placed on-disk.
- LCP's were stored in differences to their relative parent.
- Using bit-blind trees LOF SA's fringe characters were removed.

Condensed BWT

- Reversing the strings stored in index will allow backward search to match the forward prefixes.
- Pruned Suffix Tree takes $O((B+\sigma)\log n)$ space and identifies leaf in $O(m \log \sigma)$ time

3 COMPARISON AND REVIEW

An overview of the performance is shown in **Table 1**.

Inverted Index. Major disadvantage of in-memory strategy is index construction costs perhaps two-thirds of the total time. Also, the entire list has to be relocated if there is not enough space for new updates. Thus, non-sequential disk updates maybe possible. Merge-based update pose the problem of entire file being written/read whenever any on-disk inverted file is updated, even for a small posting. Hence, this approach becomes problematic when dealing with large lists, where they have to be copied several times. Substantial performance improvement happen using the hybrid approach as it avoids unnecessary disk transfers during merging of the unchanged portions of posting lists.

Nextword index Index. Substantially faster than a regular structure for resolving typical two or three word phrase queries. Monodirectional phrase browsing in large collections would be supported by nextword indexes. The drawback is that there is not much of

a significant size improvement when compared to conventional techniques.

On-Disk Suffix Arrays. The two-level suffix array requires less disk space using disk blocks that are based on prefixes allowing reductions between blocks. The condensed BWT is comprehensive, *existence* and *count* queries can be resolved without disk access. The drawback is the construction of suffix array; however, if generated from a central service, the two-level structure would be ideal for querying on low-cost devices.

REFERENCES

[1] M. Agosti and A. F. Smeaton. Information retrieval and hypertext. *Kluwer Academic Publishers, Dordrecht, Netherlands.*, 1996.
[2] D. Bahle, H. E. Williams, and J. Zobel. Compaction techniques for nextword indexes. *String Processing and Information Retrieval*, pages 33–45, 2001.
[3] S. Büttcher and C. L. A. Clarke. A hybrid approach to index maintenance in dynamic text retrieval systems,, london, uk, april 10-12, 2006, proceedings. *Proc. ECIR*, pages 229–240.
[4] S. Gog, A. Moffat, J. S. Culpepper, A. Turpin, and A. Wirth. Compressing integers for fast file access. *IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING*, 26(8), 2014.
[5] R. Sinha, S. J. Puglisi, A. Moffat, and A. Turpin. Improving suffix array locality for fast pattern matching on disk. *ACM SIGMOD*, pages 661–672, 2008.
[6] H. E. Williams, J. Zobel, and P. Anderson. What's next? index structures for efficient phrase querying. *Australasian Database Conference*, pages 141–152, 1999.
[7] J. Zobel and A. Moffat. Inverted files for text search engines. *ACM Comp. Surv.*, 38(2), 2006.