Compressed Data Structures with Relevance

(Invited Keynote)

Jeffrey Scott Vitter
The University of Kansas
Lawrence, KS 66047 USA
jsv@ku.edu

ABSTRACT

We describe recent breakthroughs in the field of compressed data structures, in which the data structure is stored in a compressed representation that still allows fast answers to queries. We focus in particular on compressed data structures to support the important application of pattern matching on massive document collections. Given an arbitrary query pattern in textual form, the job of the data structure is to report all the locations where the pattern appears. Another variant is to report all the documents that contain at least one instance of the pattern. We are particularly interested in reporting only the most relevant documents, using a variety of notions of relevance. We discuss recently developed techniques that support fast search in these contexts as well as under additional positional and temporal constraints.

Categories and Subject Descriptors

E.4 [**Data**]: Coding and Information Theory—*data compaction and compression*; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*pattern matching*; H.3.1 [**Information Storage and Retrieval**]: Content Analysis and Indexing—*indexing methods*; H.3.2 [**Information Storage and Retrieval**]: Information Storage—*file organization*; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*search process.*

General Terms

Algorithms, Design, Performance, Theory.

Keywords

Compressed data structure, compression, data compression, entropy, external memory, index, pattern matching, search.

1. Introduction

The field of compressed data structures began over 20 years ago [9] with the goal of achieving fast query time while using a compressed representation. The challenge is to construct a compressed representation of the input data — substantially smaller than the original input size, ideally about the size of the input data in entropy-compressed format — so that it can be queried directly without need for decompression, yet still provide the same functionality and speed as traditional data structures.

In this invited presentation, we focus on the ubiquitous problem of searching for a query pattern in a massive document collection. Web search engines such as Google support word-based search using the well-known inverted index: For each word in the document collection, the inverted index maintains a list of locations where the word appears. The search for a particular word involves a simple ordered search of the inverted index. However, in many applications, we are interested in searching for a query pattern that is not in word form — such as a partial word, multiword phrase, or more generally an arbitrary sequence of characters. In fact, in some document collections, there may be no natural notion of "word" at all; examples include collections of DNA sequences, protein data, music scores, Asian languages, and other heterogeneous sources.

In all those cases, we must use an alternative to inverted indexes. Suffix trees and suffix arrays provide the desired functionality, but for massive document collections they use an unacceptably large amount of space: A document collection containing n characters from an alphabet Σ occupies $n \log |\Sigma|$ bits.[1] However, a suffix tree or suffix array built on that document collection requires $O(n \log n)$ bits, which in practice is typically a factor of 4–15 times larger. And compared with the size of the documents in *compressed* format, the space blow-up is even more severe.

Theoretical breakthroughs just over a dozen years ago led to the development of a new generation of space-efficient search indexes. The *compressed suffix array* [5, 6, 14, 15, 4, 16] and the *FM-index* [1, 2, 3, 10] achieve high-order entropy space compression, and their query time is proportional to the query pattern size (in units of computer words) plus the product of the output size and $\operatorname{polylog} n$. Moreover, these new indexes are *self-indexing* in that they can reconstruct any part of the original document collection in a random-access manner, and thus the document collection becomes redundant and can be discarded. The net effect is that the document collection can be completely replaced by a much smaller index structure.

The output of all locations where a query pattern occurs may be unmanageably large, and it is often more helpful to report only the documents that contain the query pattern

[1] We assume for simplicity that $|\Sigma|$ is a power of 2.

CIKM'12, October 29–November 2, 2012, 2012 Maui, HI, USA.
ACM 978-1-4503-1156-4/12/10.

rather than all the matching locations within those documents. It is even more meaningful to report only the *most relevant* documents that contain the query pattern. We can define relevance in various ways — such as by the frequency of the pattern in the document, or perhaps by how close the pattern is to other instances, or as in Google by the static notion of the document's PageRank.

The *document indexing* problem, for a query pattern and query parameter k, is to find the top-k most relevant documents containing the pattern; in the absolute version, we find those documents that have a relevance value of at least k. We discuss recent advances [11, 7, 13, 17] in the standard RAM model as well as in the external memory model [19], when the data collection and the index are too massive to fit into the internal memory of a computer and must instead be stored externally, such as on disk. We give algorithmic paradigms in these models for efficient indexing, including dealing with approximations, and illustrate some biological applications.

In summary, inverted indexes have well-known strengths: They are relatively easy to implement using small amounts of memory space. They can be readily adapted for efficient implementation in external memory. They allow dynamic updates and distributed operations. And they are easily tuned for top-k applications. However, they are inherently word-based and cannot readily handle searches for arbitrary patterns. Nor are they self-indexing. An ongoing challenge in the field is to simultaneously achieve the many advantages inverted indexes offer as well as the important features of self-indexing and the ability to search for general patterns. In this presentation, we discuss several exciting developments toward answering that challenge.

For more details, the reader is referred to surveys on indexing for massive string data [12, 8] as well as to recent papers with several co-authors, which can be found on the author's web page [18].

2. Acknowledgments

Support was provided in part by National Science Foundation grants CCF–0621457 and CCF–1017623. Thanks go to Wing-Kai Hon, Rahul Shah, and Cheng Sheng for helpful comments.

3. References

[1] P. Ferragina and G. Manzini. Opportunistic data structures with applications. In *Proc. IEEE Symp. on Foundations of Computer Science*, volume 41, pages 390–398, November 2000.

[2] P. Ferragina and G. Manzini. Indexing compressed texts. *Journal of the ACM*, 52(4):552–581, 2005.

[3] L. Foschini, R. Grossi, A. Gupta, and J. S. Vitter. When indexing equals compression: Experiments on suffix arrays and trees. *ACM Transactions on Algorithms*, 2(4):611–639, 2006. Conference versions in *SODA 2004* and *DCC 2004*.

[4] R. Grossi, A. Gupta, and J. S. Vitter. High-order entropy-compressed text indexes. In *Proc. ACM-SIAM Symp. on Discrete Algorithms*, January 2003.

[5] R. Grossi and J. S. Vitter. Compressed suffix arrays and suffix trees with applications to text indexing and string matching. In *Proc. ACM Symp. on Theory of Computing*, volume 32, pages 397–406, May 2000.

[6] R. Grossi and J. S. Vitter. Compressed suffix arrays and suffix trees with applications to text indexing and string matching. *SIAM Journal on Computing*, 35(32):378–407, 2005.

[7] W.-K. Hon, R. Shah, and J. S. Vitter. Space-efficient framework for top-k string retrieval problems. In *Proc. IEEE Symp. on Foundations of Computer Science*, Atlanta, October 2009.

[8] W.-K. Hon, R. Shah, and J. S. Vitter. Compression, indexing, and retrieval for massive string data. In *Proc. Symp. on Combinatorial Pattern Matching*, volume 6129 of *LNCS*, pages 260–274. Springer, June 2010. Invited keynote paper.

[9] G. Jacobson. Succinct static data structures. Technical Report CMU-CS-89-112, Dept. of Computer Science, Carnegie-Mellon University, January 1989.

[10] V. Mäkinen and G. Navarro. Implicit compression boosting with applications to self-indexing. In *Proc. Intl. Symp. on String Processing Information Retrieval*, volume 4726 of *LNCS*, pages 229–241. Springer, October 2007.

[11] S. Muthukrishnan. Efficient Algorithms for Document Retrieval Problems. In *Proc. ACM-SIAM Symp. on Discrete Algorithms*, pages 657–666, 2002.

[12] G. Navarro and V. Mäkinen. Compressed full-text indexes. *ACM Computing Surveys*, 39(1):article 2, 2007.

[13] G. Navarro and Y. Nekrich. Top-k document retrieval in optimal time and linear space. In *Proc. ACM-SIAM Symp. on Discrete Algorithms*, pages 1066–1077, January 2012.

[14] K. Sadakane. Compressed text databases with efficient query algorithms based on the compressed suffix array. In *Proc. Intl. Symp. on Algorithms and Computation*, volume 1969 of *LNCS*, pages 410–421. Springer, December 2000.

[15] K. Sadakane. New text indexing functionalities of the compressed suffix arrays. *Journal of Algorithms*, 48(2):294–313, 2003.

[16] K. Sadakane. Compressed suffix trees with full functionality. *Theory of Computing Systems*, 41(4):589–607, 2007.

[17] R. Shah, C. Sheng, S. V. Thankachan, and J. S. Vitter. On optimal top-k string retrieval, July 2012. arXiv:1207.2632v1 [cs.DS].

[18] J. S. Vitter. Online publication library, `http://www.ittc.ku.edu/~jsv/Papers/catalog/`.

[19] J. S. Vitter. *Algorithms and Data Structures for External Memory*. Foundations and Trends in Theoretical Computer Science. Now Publishers, Hanover, MA, 2008.