Preface

Data volumes are exploding as organizations and users collect and store increasing amounts of information for their own use and for sharing it with others. To cope with these large datasets, software developers typically take advantage of faster and faster I/O-subsystems and multicore processors, and/or they exploit the virtual memory to make the caching and delivering of data requested by their algorithms simple and effective whenever their working set is small. Sometimes, they gain an additional speed-up by reducing the storage usage of their algorithms because this impacts favorably on the number of machines/disks required for a given computation, and on the amount of data that is transferredcached to/in the faster memory levels closer to the CPUs. However it is well-known, both in algorithm and software engineering communities, that the principled exploitation of all these issues via a proper arrangement of data and a properly structured algorithmic computation can abundantly surpass the best expected technology advancements and the help coming from operating systems or heuristics. As a result, data compression and indexing nowadays play a key role in the design of modern algorithms for applications that manage massive datasets. Their effective combination is, however, not easy because of three main reasons:

- each memory level (cache, DRAM, mechanical disk, SSD,...) has its own cost, capacity, latency, and bandwidth, and thus accessing data in the memory levels closer to the CPU is orders of magnitude faster than accessing data at the last levels. Therefore, space-efficient algorithms and data structures should be space and I/O-conscious and thus deploy (temporal and spatial) locality of reference as a key principle in their design.
- compressed space typically comes at a cost—namely, compression and/or decompression time—so that compression should be plugged into algorithms and data structures without impairing the efficiency of their operations.
- data compression and indexing seem “opposite approaches” because the former aims at removing data redundancies, whereas the latter introduces extra-data in the index to support faster operations.

It is, thus, not surprising that, until recently, algorithm and software designers were faced with a dilemma: achieve either efficient compression at the cost of slow operations over the compressed data, or vice versa.
This dichotomy was successfully addressed starting from the year 2000
[Ferragina and Manzini (2005)], due to various scientific achievements which
showed how to relate Information Theory and String-Matching concepts, in a way
that index regularities that show up when data is compressible are discovered and
exploited to reduce index occupancy without impairing query efficiency (see the
surveys [Navarro and Mäkinen (2007); Ferragina et al. (2008a)] and references
therein). The net result has been the design of compressed data structures for
indexing data (aka compressed indexes, or compressed and searchable data for-
mats) that take space close to the kth order entropy of the input data, and support
the powerful substring queries and the extraction of arbitrary portions of data in
time close to the one required by (optimal) uncompressed indexes. Given this latter
feature, these data structures are sometime called self-indexes.

Originally designed for raw data (i.e., strings), compressed indexes have been
recently extended to deal with other data types, such as sequences (see e.g.,
[Ferragina et al. (2007); Ferragina and Venturini (2007b); Pătrașcu (2008); Grossi
et al. (2013)]), dictionaries (see e.g., [Ferragina and Venturini (2010, 2013); Hon
et al. (2008)]), unlabeled trees (see e.g., [Benoit et al. (2005); Jansson et al. (2007);
Farzan et al. (2009); Sadakane and Navarro (2010)]), labeled trees (see e.g.,
[Ferragina et al. (2009b); Ferragina and Rao (2008)]), graphs (see e.g., [Claude and
Navarro (2010)]), binary relations and permutations (see e.g., [Barbay and Navarro
(2009); Barbay et al. (2010)]), and many others. The consequence of this
impressive flow of results is that, nowadays, it is known how to index almost any
(complex) data-type in compressed space and support various kinds of queries fast
over it. From a theoretical point of view, and as far as the RAM model is con-
cerned, the above dichotomy may be considered successfully addressed.

The thesis of Rossano Venturini provides significant contributions to several of
those issues, by achieving deep results in four main scenarios which combine in
different ways compression and indexing goals, looking at them from the theo-
retical and the engineering perspective. Here I summarize the main achievements
and refer the reader for more details to the cited Chapters and the numerous
publications that spurred from this research.

**Lossless data compression.** The problem of lossless data compression consists of
compactly representing data in a format that allows their faithful recovery. A
recent challenging trend has been the one which asks to improve the performance
of a given compressor \( C \) by partitioning the input data in a way that compressing
each individual part by \( C \) is better than applying \( C \) to the whole input. Chapter 3
investigates this problem by taking as compressor \( C \) one whose compression
performance can be bounded in terms of the zero-th or the kth order empirical
entropy of the input data. Previous results offered poor or sub-optimal solutions
[Buchsbaum et al. (2003); Giancarlo and Sciorino (2003); Buchsbaum et al.
(2000, 2003); Ferragina et al. (2005a)]. In Chap. 3 it is described the first algorithm
which computes in \( O(n \log_{1+\varepsilon} n) \) time and \( O(n) \) space, a partition of the input data
whose compressed output is guaranteed to be no more than \( (1 + \varepsilon) \)-worse the
optimal one, where \( \varepsilon > 0 \) is fixed in advance and \( n \) is the input length.
In the next Chap. 4 is addressed the design of the compressor $C$, taking into consideration the very famous class of dictionary-based compressors introduced by Lempel and Ziv in the late 70s [Ziv and Lempel (1977)], currently at the core of many softwares like gzip, zip, pkzip, lzma2, LZ4, just to cite a few. This compression scheme squeezes an input text by replacing some of its substrings with (shorter) codewords which are actually pointers to phrases in a dictionary. Surprisingly enough, although many fundamental results were known about the speed and effectiveness of this compression process, there was no parsing scheme which guaranteed to achieve the minimum number of bits for a fixed class of codewords. Chapter 4 presents an algorithm which achieves bit-optimality in the compressed output-size of LZ77 by taking efficient/optimal time and optimal space. The experimental results show also that this theoretical achievement is effective in practice, by beating renowned LZ77-implementations such as lzma2 and LZ4 an interesting example of a win-win situation of a theoretical idea with a practical impact.

Random access to compressed data. This is a key problem in modern data-storage solutions, whenever data have to be compressed and still provided to their underlying applications. Classical compression algorithms fail in offering this facility, because they support only the decompression of the whole compressed file. Conversely, more and more applications nowadays need to fast access portions of those compressed data: think, e.g., to the collection of Web pages and the search engine that needs to access them at each user query in order to return pertinent snippets for the query results to be shown to the search engine users. Chapter 5 addresses this problem by presenting an elegant scheme that represents data into a compressed form, with provable space bounds in terms of its $k$th order entropy, still providing access to any of its substrings in optimal time. It is remarkable that this result improves known achievements via a very simple approach.

Engineering compressed indexes. The theory of compressed full-text indexes was mature enough to ask ourselves if and how this could be a breakthrough for compressed data storage. The Pizza and Chili’s effort described in Chap. 6, joint between the University of Pisa and the University of Chile, contributes in this direction by offering a set of tuned implementations of the most successful compressed text indexes, together with effective test-beds and scripts for their automatic validation and test. These indexes, all adhering a standardized API, were extensively tested on various datasets showing that they take roughly the same space used by traditional compressors (like gzip and bzip2), with the additional feature of being able to extract arbitrary portions of compressed data at the speed of 2–4 MB/sec, and to search for arbitrary substrings (hence, not necessarily words) at few µsec per occurrence. Compared to the approach described in the previous point, this adds to the compressed data-storage scheme the capability to search for arbitrary substrings in an efficient way. It is remarkable that the Pizza and Chili site is currently one of the most used software libraries in the Computational Biology community, as witnessed by the numerous papers and international projects which mention it.
Compressed indexes for string dictionaries. The final problem addressed in this thesis concerns with the compressed indexing of a large dictionary of strings having variable length, and supporting tolerant retrieval queries [Manning et al. (2008)] such as membership, prefix/suffix, and substring searches. As strings are getting longer and longer, and dictionaries of strings are getting larger and larger, it becomes crucial to devise implementations for the above primitive which are fast and work in compressed space. This is the topic of Chap. 7 that introduces the so-called compressed permuterm index to solve the problem above in efficient time and $k$th order entropy space. This result is so simple and elegant that it has been mentioned in the Manning-Raghavan-Schülze’s book [Manning et al. (2008)], and constitutes the core of a submitted US Patent owned by Yahoo!

Overall, I think that this Thesis is a nice balance of significant theoretical achievements and precious algorithm-engineering results that provide both a strong contribution to the theory of Data Structures and another witness, if it were further needed, of the fact mentioned at the beginning of my Preface that the proper “arrangement of data and structuring of computation” can introduce improvements which abundantly surpass the best expected technology advancements and the cute heuristics devised by software engineers!

Paolo Ferragina
University of Pisa

References


Compressed Data Structures for Strings
On Searching and Extracting Strings from Compressed Textual Data
Venturini, R.
2014, XIV, 118 p. 18 illus., Hardcover
ISBN: 978-94-6239-032-4
A product of Atlantis Press