Information Systems 56 (2016) 73–108

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/infosys

Practical compressed string dictionaries ☆

Miguel A. Martínez-Prieto [a,*,1], Nieves Brisaboa [b], Rodrigo Cánovas [c], Francisco Claude [d,3], Gonzalo Navarro [e,2]

[a] DataWeb Research, Department of Computer Science, University of Valladolid, Spain
[b] Database Laboratory, University of A Coruña, Spain
[c] NICTA Victoria Research Laboratory, Department of Computing and Information Systems (CIS), The Univerity of Melbourne, Australia
[d] Escuela de Informática y Telecomunicaciones, Universidad Diego Portales, Chile
[e] CeBiB — Center of Biotechnology and Bioengineering, Department of Computer Science, University of Chile, Chile

ARTICLE INFO

Article history:
Received 9 May 2014
Received in revised form
28 July 2015
Accepted 18 August 2015
Recommended by Ralf Schenkel
Available online 21 September 2015

Keywords:
Compressed string dictionaries
Text processing
Text databases
Compressed data structures

ABSTRACT

The need to store and query a set of strings – a *string dictionary* – arises in many kinds of applications. While classically these string dictionaries have accounted for a small share of the total space budget (e.g., in Natural Language Processing or when indexing text collections), recent applications in Web engines, Semantic Web (RDF) graphs, Bioinformatics, and many others handle very large string dictionaries, whose size is a significant fraction of the whole data. In these cases, string dictionary management is a scalability issue by itself. This paper focuses on the problem of managing large static string dictionaries in compressed main memory space. We revisit classical solutions for string dictionaries like hashing, tries, and front-coding, and improve them by using compression techniques. We also introduce some novel string dictionary representations built on top of recent advances in succinct data structures and full-text indexes. All these structures are empirically compared on a heterogeneous testbed formed by real-world string dictionaries. We show that the compressed representations may use as little as 5% of the original dictionary size, while supporting lookup operations within a few microseconds. These numbers outperform the state-of-the-art space/time tradeoffs in many cases. Furthermore, we enhance some representations to provide prefix- and substring-based searches, which also perform competitively. The results show that compressed string dictionaries are a useful building block for various data-intensive applications in different domains.

1. Introduction

A *string dictionary* is a data structure that maintains a set of strings. It arises in classical scenarios like Natural Language (NL) processing, where finding the *lexicon* of a text corpus is the first step in analyzing it [56]. They also arise as a component of *inverted indexes*, when indexing NL text collections [79,19,6]. In both cases, the dictionary comprises all the different words used in the text collection. The dictionary implements a bijective function that maps strings to identifiers (IDs, generally integer values) and back. Thus, a string dictionary must provide, at least,

☆ A preliminary version of this paper appeared in *Proceedings of 10th International Symposium on Experimental Algorithms* (*SEA*), 2011, pp. 136–147.

* Corresponding author.
E-mail addresses: migumar2@infor.uva.es (M.A. Martínez-Prieto), brisaboa@udc.es (N. Brisaboa), rcanovas@student.unimelb.edu.au (R. Cánovas), fclaude@recoded.cl (F. Claude), gnavarro@dcc.uchile.cl (G. Navarro).

[1] Funded by the Funded by the Spanish Ministry of Economy and Competitiveness: TIN2013-46238-C4-3-R, and ICT COST Action KEYSTONE (IC1302).
[2] Funded with basal funds FB0001, Conicyt, Chile.
[3] Funded in part by Fondecyt Iniciación 11130104.

http://dx.doi.org/10.1016/j.is.2015.08.008
0306-4379/

two complementary operations: (i) `string-to-ID` *locates* the ID for a given string, and (ii) `ID-to-string` *extracts* the string identified by a given ID.

String dictionaries are a simple and effective tool for managing string data in a wide range of applications. Using dictionaries enables replacing (long, variable-length) strings by simple numbers (their IDs), which are more compact to represent and easier and more efficient to handle. A compact dictionary providing efficient mapping between strings and IDs saves storage space, processing and transmission costs, in data-intensive applications. The growing volume of the datasets, however, has led to increasingly large dictionaries, whose management is becoming a scalability issue by itself. Their size is of particular importance to attain the optimal performance under restrictions of main memory.

This paper focuses on techniques to compress string dictionaries and the space/time tradeoffs they offer. We focus on *static* dictionaries, which do not change along the execution. These are appropriate in the many applications using dictionaries that either are static or are rebuilt only sparingly. We revisit traditional techniques for managing string dictionaries, and enhance them with data compression tools. We also design new structures that take advantage of more sophisticated compression methods, succinct data structures, and full-text indexes [62]. The resulting techniques enable large string dictionaries to be managed within compressed space in main memory. Different techniques excel on different application niches. The least space-consuming variants operate within microseconds while compressing the dictionary to as little as 5% of its original size.

The main contributions of this paper can be summarized as follows:

1. We present, as far as we know, the most exhaustive study to date of the space/time efficiency of compressed string dictionary representations. This is not only a survey of traditional techniques, but we also design novel variants based on combinations of existing techniques with more sophisticated compression methods and data structures.
2. We perform an exhaustive experimental tuning and comparison of all the variants we study, on a variety of real-world scenarios, providing a global picture of the current state of the art for string dictionaries. This results in clear recommendations on which structures to use depending on the application.
3. Most of the techniques outstanding in the space/time tradeoff turn out to be combinations we designed and engineered, between classical methods and more sophisticated compression techniques and data structures. These include combinations of binary search, hashing, and Front-Coding with grammar-based and optimized Hu-Tucker compression. In particular, uncovering the advantages of the use of grammar compression for string dictionaries is an important finding.
4. We create a C++ library, `libCSD` (*Compressed String Dictionaries*), implementing all the studied techniques. It is publicly available at https://github.com/migumar2/libCSD under GNU LGPL license.
5. We go beyond the basic `string-to-ID` and `ID-to-string` functionality and implement advanced searches for some of our techniques. These enable *prefix*-based searching for most methods (except Hash ones) and *substring* searches for the FM-Index and XBW dictionaries.

The paper is organized as follows. Section 2 provides a general view of string dictionaries. We start describing various real-world applications where large dictionaries must be efficiently handled, then define the notation used in the paper, and finally describe classical and modern techniques used to support string dictionaries, particularly in compressed space. Section 3 provides the minimal background in data compression necessary to understand the various families of compressed string dictionaries studied in this paper. Section 4 describes how we have applied those compression methods so that they perform efficiently for the dictionary operations. Sections 5–9 focus on each of the families of compressed string dictionaries. Section 10 provides a full experimental study of the performance of the described techniques on dictionaries coming from various real-world applications. The best performing variants are then compared with the state of the art. We find several niches in which the new techniques dominate the space/time tradeoffs of classical methods. Finally, Section 11 concludes and describes some future work directions.

2. String dictionaries

2.1. Applications

This section takes a short tour over various example applications where handling very large string dictionaries is a serious issue and compression could lead to considerable improvements.

NL Applications: It is the most classic application area of string dictionaries. Traditionally, the size of these dictionaries has not been a concern because classical NL collections were carefully polished to avoid typos and other errors. On those collections, Heaps [44] formulated an empirical law establishing that, in a text of length n, the dictionary grows sublinearly as $O(n^{\beta})$, for some $0<\beta<1$ depending on the type of text. β Value is usually in the range 0.4–0.6 [6], so the dictionary of a terabyte-size collection would occupy just a few megabytes and easily fit in any main memory. Heaps' law, however, does not model well the dictionaries used in other NL applications. The use of string dictionaries in Web search engines or in Machine Translation (MT) systems are two well-known examples:

- Web collections are much less “clean” than text collections whose content quality is carefully controlled. Dictionaries of Web crawls easily exceed the gigabytes, due to typos and unique identifiers that are taken as “words”, but also due to “regular words” from multiple languages. The *ClueWeb09* dataset[4] is a real example that comprises close to 200 million different words

[4] http://boston.lti.cs.cmu.edu/Data/clueweb09