Journal of Discrete Algorithms 34 (2015) 54–61

Contents lists available at ScienceDirect

Journal of Discrete Algorithms

www.elsevier.com/locate/jda

The greedy approach to dictionary-based static text compression on a distributed system

Sergio De Agostino

Computer Science Department, Sapienza University of Rome, Via Salaria 113, 00198 Rome, Italy

ARTICLE INFO

Article history:
Available online 27 May 2015

Keywords:
Distributed algorithms
Scalability
Robustness
Static lossless compression

ABSTRACT

The greedy approach to dictionary-based static text compression can be executed by a finite-state machine. When it is applied in parallel to different blocks of data independently, there is no lack of robustness even on standard large scale distributed systems with input files of arbitrary size. Beyond standard large scale, a negative effect on the compression effectiveness is caused by the very small size of the data blocks. A robust approach for extreme distributed systems is presented in this paper, where this problem is fixed by overlapping adjacent blocks and preprocessing the neighborhoods of the boundaries.

1. Introduction

Static data compression implies the knowledge of the input type. With text, dictionary-based techniques are particularly efficient and employ string factorization. The dictionary comprises typical factors plus the alphabet characters in order to guarantee feasible factorizations for every string. Factors in the input string are substituted by pointers to dictionary copies and such pointers could be either variable or fixed length codewords. The optimal factorization is the one providing the best compression, that is, the one minimizing the sum of the codeword lengths. Efficient sequential algorithms for computing optimal solutions were provided by means of dynamic programming techniques [32] or by reducing the problem to the one of finding a shortest path in a directed acyclic graph [29]. From the point of view of sequential computing, such algorithms have the limitation of using an off-line approach. However, decompression is still on-line and a very fast and simple real time decoder outputs the original string with no loss of information. Therefore, optimal solutions are practically acceptable for read-only memory files where compression is executed only once. Differently, simpler versions of dictionary-based static techniques were proposed which achieve nearly optimal compression in practice (that is, less than ten percent loss). An important simplification is to use a fixed length code for the pointers, so that the optimal decodable compression for this coding scheme is obtained by minimizing the number of factors. Such a variable to fixed length approach is robust since the dictionary factors are typical patterns of the input specifically considered. The problem of minimizing the number of factors gains a relevant computational advantage by assuming that the dictionary is *prefix-closed* (*suffix-closed*), that is, all the prefixes (suffixes) of a dictionary element are dictionary elements [4,7,20]. The left to right greedy approach is optimal only with suffix-closed dictionaries. An optimal factorization with prefix-closed dictionaries can be computed on-line by using a semi-greedy procedure [7,20]. On the other hand, prefix-closed dictionaries are easier to build by standard adaptive heuristics [2,31]. These heuristics are based on an "incremental" string factorization procedure [24,34]. The most popular for prefix-closed dictionaries is the one presented in [33]. However, the prefix and suffix properties force the dictionary to include many useless elements which increase the pointer size and slightly reduce the compression effectiveness. A more

E-mail address: deagostino@di.uniroma1.it.

http://dx.doi.org/10.1016/j.jda.2015.05.001
1570-8667/© 2015 Elsevier B.V. All rights reserved.

natural dictionary with no prefix and no suffix property is the one built by the heuristic in [27] or by means of separator characters as, for example, space, new line and punctuation characters with natural language.

Theoretical work was done, mostly in the nineties, to design efficient parallel algorithms on a random access parallel machine (PRAM) for dictionary-based static text compression [3,8–10,16,21,22,28,30]. Although the PRAM model is out of fashion today, shared memory parallel machines offer a good computational model for a first approach to parallelization. When we address the practical goal of designing distributed algorithms we have to consider two types of complexity, the interprocessor communication and the input–output mechanism. While the input/output issue is inherent to any parallel algorithm and has standard solutions, the communication cost of the computational phase after the distribution of the data among the processors and before the output of the final result is obviously algorithm-dependent. So, we need to limit the interprocessor communication and involve more local computation to design a practical algorithm. The simplest model for this phase is, of course, a simple array of processors with no interconnections and, therefore, no communication cost. Parallel decompression is, obviously, possible on this model [10]. With parallel compression, the main issue is the one concerning scalability and robustness. Traditionally, the scale of a system is considered large when the number of nodes has the order of magnitude of a thousand. Modern distributed systems may nowadays consist of hundreds of thousands of nodes, pushing scalability well beyond traditional scenarios (extreme distributed systems).

In [1] an approximation scheme of optimal compression with static prefix-closed dictionaries was presented for massively parallel architectures, using no interprocessor communication during the computational phase since it is applied in parallel to different blocks of data independently. The scheme is algorithmically related to the semi-greedy approach previously mentioned and implementable on extreme distributed systems because adjacent blocks overlap and the neighborhoods of the boundaries are preprocessed. However, with standard large scale the overlapping of the blocks and the preprocessing of the boundaries are not necessary to achieve nearly optimal compression in practice. Furthermore, the greedy approach to dictionary-based static text compression is nearly optimal on realistic data for any kind of dictionary even if the theoretical worst-case analysis shows that the multiplicative approximation factor with respect to optimal compression achieves the maximum length of a dictionary element [31]. If the dictionary is well-constructed by relaxing the prefix property, the loss of greedy compression can go down to one percent with respect to the optimal one. In this paper, we relax the prefix property of the dictionary and present two implementations of the greedy approach to static text compression with an arbitrary dictionary on a large scale and an extreme distributed system, respectively. Moreover, we present a finite-state machine implementation of greedy static dictionary-based compression with an arbitrary dictionary that can be relevant to achieve high speed with standard scale distributed systems. We wish to point out that scalability cannot be guaranteed with adaptive dictionary approaches to data compression, as the sliding window method [25] or the dynamic one [34]. Indeed, the size of the data blocks over the distributed memory of a parallel system must be at least a few hundreds kilobytes in both cases, that is, robustness is guaranteed with scalability only with very large files [3,12,13]. This is still true with improved variants employing either fixed-length codewords [26,6] or variable-length ones [5,17–19,23].

In Section 2 we describe the different approaches to dictionary-based static text compression. The previous work on parallel approximations of optimal compression with prefix-closed dictionaries is given in Section 3. Section 4 shows the finite-state machine and the two implementations of the greedy approach for arbitrary dictionaries. Experiments are discussed in Section 5. Conclusions and future work are given in Section 6.

2. Dictionary-based static text compression

As mentioned in the introduction, the dictionary comprises typical factors (including the alphabet characters) associated with fixed or variable length codewords. The optimal factorization is the one minimizing the sum of the codeword lengths and sequential algorithms for computing optimal solutions were provided by means of dynamic programming techniques [32] or by reducing the problem to the one of finding a shortest path in a directed acyclic graph [29]. When the codewords are fixed-length, with suffix-closed dictionaries we obtain optimality by means of a simple left to right greedy approach, that is, advancing with the on-line reading of the input string by selecting the longest matching factor with a dictionary element. Such a procedure can be computed in real time by storing the dictionary in a trie data structure. If the dictionary is prefix-closed, there is an optimal semi-greedy factorization which is computed by the procedure of Fig. 1 [7,20]. At each step, we select a factor such that the longest match in the next position with a dictionary element ends to the rightest. Since the dictionary is prefix-closed, the factorization is optimal. The algorithm can even be implemented in real time with a modified trie data structure [20].

The semi-greedy factorization can be generalized to any dictionary by considering only those positions, among the ones covered by the current factor, next to a prefix that is a dictionary element [7]. The generalized semi-greedy factorization procedure is not optimal while the greedy one is not optimal even when the dictionary is prefix-closed. The maximum length of a dictionary element is an obvious upper bound to the multiplicative approximation factor of any string factorization procedure with respect to the optimal solution. We show in the next theorems that this upper bound is tight for the greedy and semi-greedy procedures when the dictionary is arbitrary [31] and that such tightness is kept by the greedy procedure even if the dictionary is prefix-closed.

Theorem 2.1. *There exists an infinite family of dictionaries such that the greedy and semi-greedy procedures produce $\Theta(m)$ approximations of the optimal factorization of an input string in the worst case, where m is the maximum length of a dictionary element.*