A Worst Case Analysis of the LZ2 Compression Algorithm with Bounded Size Dictionaries

Sergio De Agostino

Computer Science Department
Sapienza University of Rome
Via Salaria 113, 00198 Rome, Italy
deagostino@di.uniroma1.it

Abstract. We make a worst case analysis of practical implementations of LZ2 compression, where the work space remains constant with the increase of the data size and the optimal solution must work with the same on-line decoder. The memory bound implies an off-line standard polynomial time optimal solution with huge multiplicative constants and we show that an on-line approach approximates with a large factor, leaving the design of an effective and more efficient off-line coding as an open problem in this context.

Keywords: factorization, dictionary, optimality, approximation.

1 Introduction

Sheinwald, Lempel and Ziv [17] proved that the power of off-line coding is not useful if we want on-line decodable files, as far as asymptotical results are concerned. Such result extends the asymptotical optimality results of Lempel and Ziv for ergodic sources in a non-constructive way, where the on-line reading of the data from left to right works with a sublinearly bounded buffer length. In the finite case, De Agostino and Storer [8] introduced the notion of on-line decodable Ziv-Lempel (LZ2) optimal coding and proved its NP-completeness. Moreover, a sublogarithmic factor approximation algorithm cannot be realized on-line and the greedy LZ2 compression algorithm is an $O(n^{\frac{1}{4}})$ approximation with binary worst case examples, where n is the string length [7,8]. Therefore, for finite strings, one could afford the extra-cost of off-line coding in exchange of some gain in compression of data stored on a read-only memory. Considering this point, we make in this paper a worst case analysis of practical implementations of LZ2 compression, where the work space remains constant with the increase of the data size. The memory bound implies a standard polynomial time optimal solution with huge multiplicative constants and we show that the on-line approach still approximates with a large factor, leaving the design of an effective and more efficient off-line coding as an open problem in this context. In other words, the trade off between effectiveness (good compression ratio) and efficiency (practical running time) must be improved. This depends on the improvement of the LZ2 string factorization process [20] while on-line greedy LZ1 factorization [15] is already structurally optimal and the only possible improvements are at the coding level as we will discuss in the next sections.

Such need for further theoretical analysis of the power of off-line encoding that must produce on-line decodable files is motivated, as previously mentioned, by applications to read-only memories. Indeed, the design of an on-line decodable off-line

Sergio De Agostino: A Worst Case Analysis of the LZ2 Compression Algorithm with Bounded Size Dictionaries, pp. 107–113.
Proceedings of PSC 2023, Jan Holub and Jan Žďárek (Eds.), ISBN 978-80-01-07206-6 © Czech Technical University in Prague, Czech Republic

coding approximating the optimal solution does not need a real time or even slower algorithm to be considered practical since compression must be performed only once. Therefore, we can accept a time cost that could not be permitted when on-line computation is required as for decoding or on-the fly coding. Typically, LZ2 compression is more efficient but less effective than LZ1 but optimal or nearly optimal LZ2 compression might be more effective than LZ1 on several specific types of data.

In Section 2, Lempel-Ziv compression is described in relation to the LZ1 and LZ2 string factorization methods in the unbounded and bounded memory cases. Then, the on-line versus off-line computation and the greedy versus optimal solution issues are discussed for the unbounded memory case. Such issues are discussed and analyzed for the bounded memory case in Section 3 after giving the polynomial time optimal algorithm. Conclusion and future work are given in Section 4.

2 LZ2 Compression with Bounded Size Dictionaries

Lempel-Ziv compression is a dictionary-based technique [14,15,20], using a string factorization process where the factors of the string are substituted by *pointers* to copies stored in a *dictionary*, which are called *targets*.

2.1 LZ Factorizations

Given an alphabet A and a string S in A^*, the LZ1 factorization of S is $S = f_1 f_2 \cdots f_i \cdots f_k$ where f_i is the shortest substring which does not occur previously in the proper prefix $f_1 f_2 \cdots f_i$ for $1 \leq i \leq k$ [15]. LZ2 is easier to implement but less effective. The standard LZ2 greedy factorization of a string S, on which practical implementations of LZ2 compression are based, is $S = f_1 f_2 \cdots f_i \cdots f_m$ where each factor f_i is the longest match with the concatenation of a previous factor and the next character [19,20]. f_i is encoded by a pointer q_i whose target is such concatenation. Regardless of memory issues, LZ2 compression can be implemented in real time by storing the dictionary of targets with a trie data structure. When the string length goes to infinity, also the dictionary size does.

2.2 Bounded Size Dictionaries

In practical implementations the dictionary size is bounded by a constant and the pointers have equal size [18]. Let d be the cardinality of the fixed size dictionary (alphabet characters are always dictionary elements). With the most naive approach, there is a first phase of the factorization process where the dictionary is filled up and "frozen". Afterwards, the factorization continues in a non-adaptive way using the factors of the frozen dictionary. In other words, the factorization of a string S is $S = f_1 f_2 \cdots f_i \cdots f_k$ where f_i is the longest match with the concatenation of a previous factor f_j and the next character, where $j \leq d - \alpha$ and α is the alphabet size. The shortcoming of this heuristic is that after processing the string for a while the dictionary often becomes obsolete. Therefore, after the dictionary is filled up, the compression ratio is monitored. When the ratio deteriorates, a better heuristic deletes all the elements from the dictionary but the alphabet characters and restarts new adaptive and non-adaptive phases. Let $S = f_1 f_2 \cdots f_j \cdots f_i \cdots f_k$ be the factorization of the input string S computed by the LZ2 compression algorithm using such heuristic to bound the dictionary. Let j be the highest index less than i where a restarting

operation happens. Then, f_j is an alphabet character and f_i is the longest match with the concatenation of a previous factor f_h, with $h \geq j$, and the next character (1 and $m+1$ are considered restarting positions by default). This LZ2 compression heuristic with constant work space is called LZC [3] (a variation of LZW compression where the dictionary is restarted with no monitoring) and it is used by the Unix command Compress since it has a good compression effectiveness and it is easy to implement.

2.3 Optimal Factorizations

In the unbounded case, the pointer encoding the factor f_i has a size increasing with the index i. This means that the lower is the number of factors for a string of a given length, the better is the compression. The factorizations described in the previous subsections are produced by greedy algorithms. The question is whether the greedy approach is always optimal, that is, if we relax the assumption that each factor is the longest match, can we do better than greedy? The answer is negative with suffix dictionaries (every suffix of a dictionary element is a dictionary element) as for LZ1 compression. On the other hand, the greedy approach is not always optimal for LZ2 compression. We define $S = f_1 \cdots f_k$ a *feasible* LZ2 factorization if each factor f_i is equal to the concatenation of f_j and the next character, for some $j < i$. A feasible LZ2 factorization with the smallest number of factors is an *optimal* LZ2 factorization. Obviously, the coding of every feasible LZ2 factorization works with the same decoder of the standard greedy LZ2 coding.

As mentioned in the introduction, the optimal approach is NP-complete [8] and the greedy algorithm approximates with an $O(n^{\frac{1}{4}})$ multiplicative factor the optimal solution [7]. Moreover, a sublogarithmic factor approximation algorithm cannot be realized on-line [8]. Although these results can be viewed to be in some sense negative, they serve to motivate the need for further theoretical analysis of the power of off-line encoding that must produce on-line decodable files, as pointed out in [8]. The design of a practical on-line decodable off-line approximation algorithm has important applications to read-only memories. So, we produce further theoretical analysis for the bounded case in the next section.

3 Bounded Memory Greedy versus Optimal Analysis

The memory bound implies a standard polynomial time optimal solution with huge multiplicative constants and we show that an on-line approach still approximates with a large factor. We show the polynomial time algorithm in the first subsection. The second subsection discusses the on-line versus off-line computation issue, while the greedy versus optimal analysis is given in the third subsection.

3.1 The Polynomial Time Algorithm

A *feasible* d-LZC factorization $S = f_1 \cdots f_k$ is such that the number of different concatenations of a factor with the next character between f_h and f_t (f_t is not counted) is less or equal than d decreased by the alphabet size, with h and t two consecutive positions where the restarting operation happens (no restarting between h and t), and each factor f_i with $h < i < t$ is equal to $f_j c$, where c is the first character of f_{j+1} and $h \leq j < i$ (1 and $k+1$ are considered restarting positions by default, meaning that

frozen dictionaries relate to special cases of feasible factorizations). We define *optimal* the feasible d-LZC factorization with the smallest number of factors. The *greedy* d-LZC factorization is the one described in the previous section (the Unix command Compress works with $d = 2^{16}$). We assume, as it happens in practical implementations as well, that the coding inserts a special character when the restarting operation happens. This avoids monitoring of the compression ratio in the standard applications and makes the coding of every feasible factorization decodable (the decrement by one unit of the dictionary size caused by the special character is obviously irrelevant for the compression effectiveness).

A practical algorithm to compute the optimal solution is not known. In order to have a polynomial time optimal algorithm, the bound to the dictionary size should be sublogarithmic. However, the number of all the possible dictionaries induced by a feasible d-LZC factorization of any input string is constant since d and the alphabet cardinality are constant. Given an input string, pair each of these dictionaries with each position of such string. Link the pair (p, D), where p is a position of the string and D is one of the dictionaries, to the pair $(p + \ell + 1, D')$ if ℓ is the length of a dictionary element matching the string in p and D' is the updating of D corresponding to the choice of such dictionary element as factor of a feasible d-LZC factorization. If the match ends the string, the pair links to a special node $\overline{v}$. Also, link the pair (p, D) to (p, A) where A is the dictionary comprising only the alphabet characters, in order to have the possibility of picking p as restarting position. The optimal d-LZC factorization and the sequence of pointers is given by the shortest path from $(1, A)$ to $\overline{v}$. Such polynomial time algorithm is, obviously, unpractical since the number of nodes (pairs) is linear in the string length but the number of dictionaries is a huge multiplicative constant.

3.2 On-line versus Off-line Computation

A trivial upper bound to the approximation multiplicative factor of a feasible factorization with respect to the optimal one is the maximum factor length of the optimal solution, that is, the height of the trie storing the dictionary. Such upper bound is $\Theta(d)$ in the worst case, where d is the dictionary size ($O(d)$ follows from the feasibility of the factorization and $\Omega(d)$ from the factorization of the unary string). In practice, a dictionary comprises thousands of elements and even a logarithmic approximation multiplicative factor is too large. In [8], it is shown in the unbounded case that a sublogarithmic approximation of the optimal LZ2 coding cannot be realized by means of an example binary string $X = pref(n)suff(n)$, where the prefix $pref(n)$ of length $\Omega(n)$ is such that, for any on-line algorithm applied on it, an appropriate suffix $suff(n)$ of length $\Omega(n)$ can be concatenated in order to fool the on-line strategy. From the definition of feasible d-LZC factorization, we know that a dictionary between two restarting positions might comprise less than d elements.

Definition 1. *We call Δ the highest number of elements a dictionary is composed of between two restarting positions in an optimal d-LZC factorization.*

We can adapt the example string X in [8] to the bounded case by considering d the number of different factors selected by the on-line strategy on X, so that it will be at least a logarithmic approximation of Δ. Moreover, we can append to X a sequence of characters where the dictionary performs very badly in order to have a string Y such that $|Y|$ is $\Theta(|X|)$ and LZC compression applied to an arbitrarily

long string $YYY\cdots Y$ has restarting positions on the first character of Y. Then, any on-line approach produces an $\Omega(\log(\Delta))$ approximation of the optimal LZC coding of Y^t for any positive integer t.

3.3 The Greedy versus Optimal Analysis

The proof of the following theorem employs techniques similar to the ones for the unbounded dictionary case of [7]. A preliminay version of this theorem was shown in [6] without giving worst case examples. Such examples are described in this subsection after the proof of the theorem (we suggest to study first the proofs of theorem 3.1 and theorem 3.2 in [7] for the unbounded cases).

Theorem 2. *The greedy d-LZC factorization is an $O(\sqrt{\Delta})$ approximation of the optimal one.*

Proof. Let S be the input string and let R be a substring of S given by the concatenation of the factors of the greedy d-LZC factorization between two consecutive positions where the restarting operation happens. Let T be the trie storing the set I of strings corresponding to factors of the optimal d-LZC factorization of S contained in R. Let Φ be the number of occurrences of all these factors in R. We call an element of the dictionary built by the greedy d-LZC factorization of S an *internal occurrence* if it corresponds to a substring of a factor of I in R. We denote with M_T the number of internal occurrences. The number of non-internal occurrences is less than $|I|$. Therefore, we can consider only the internal ones. For each factor $f \in I$, an internal occurrence corresponding to f is represented by a subpath of the path representing f in T. Let u be the endpoint at the lower level in T of this subpath (which, obviously, represents a prefix of f). Let $d(u)$ be the number of subpaths representing internal occurences with endpoint u and let $c(u)$ be the total sum of their lengths. Since the occurrences (internal or not) are different from each other between two consecutive positions where the restarting operation happens and two equal length subpaths with the same endpoint represent the same factor, we have $c(u) \geq d(u)(d(u)+1)/2$. Therefore

$$1/2 \sum_{u \in T} d(u)(d(u)+1) \leq \sum_{u \in T} c(u) \leq 2|S| \leq 2H_T\Phi$$

where H_T is the height of T and the multiplicative factor 2 is due to the fact that occurrences of dictionary elements may overlap. Since $M_T = \sum_{u \in T} d(u)$, we have

$$M_T^2 \leq |I| \sum_{u \in T} d(u)^2 \leq |I| \sum_{u \in T} d(u)(d(u)+1) \leq 4|T|H_T\Phi$$

where the first inequality follows from the fact that the arithmetic mean is less than the quadratic mean. Then

$$M_T \leq \sqrt{4|I|H_T\Phi} = \Phi\sqrt{\frac{4|I|H_T}{\Phi}} \leq 2\Phi\sqrt{H_T}$$

Since the trie height is $O(\Delta)$, the theorem statement follows. □

We need to adapt the worst case example for the unbounded dictionary case [7] in order to have one for the bounded case. To reformulate such result in our context,

if we let d be the number of factors of the LZ2 factorization of a string of length n in the unbounded case, there exists a binary string X of lenght n on which the optimal factorization working with the same decoder has a number of factors and produces a number of dictionary elements, which are both $\Theta(d^{2/3})$. As in the previous subsection, we can append to X a sequence of characters where the dictionary performs very badly in order to have a string Y such that $|Y|$ is $\Theta(|X|)$ and LZC compression applied to an arbitrarily long string as $YYY \cdots Y$ has restarting positions on the first character of Y. So, the optimal solution employs two thirds of the dictionary space on the input blocks up to a multiplicative contant. On the other hand, the greedy solution cost approximates the optimal solution cost with a multiplicative approximation factor which is, up to a multiplicative constant, greater than the square root of the actual dictionary size needed by the optimal approach. So, it follows from the proof of Theorem 1 that the square root of the actual dictionary size needed by the optimal approach is a tight bound to the approximation factor of the greedy approach, up to multiplicative constants.

4 Conclusion

The gap between on-line and off-line computation, shown in this paper, has its stringological reason in the structure of the dictionary which might not contain all the suffixes of its elements. Differently, with LZ1 coding dictionaries have this property and greedy factorizations are optimal. However, the coding is more expensive and on-line improved variants exist employing either fixed-length codewords [4,16] or variable-length ones [5,9,10,11,12,13]. Moreover, there are off-line approaches to improve LZ1 coding working with on-line decoders [1,2]. As previously pointed out, with LZ2 coding practical off-line string factorizations approximating on-line decodable optimal solutions could be more effective than LZ1 compression on several specific types of data.

References

1. A. Apostolico and S. Lonardi: *Compression of biological sequences by greedy off-line textual substitution*, in Proceedings IEEE Data Compression Conference, 2000, pp. 143–152.
2. A. Apostolico and S. Lonardi: *Off-line compression by greedy textual substitution*, in IEEE Proceedings, vol. 88, 2000, pp. 1733–1744.
3. T. C. Bell and I. H. Witten: *Text Compression*, Prentice Hall, 1990.
4. M. Crochemore, A. Langiu, and F. Mignosii: *Note on the greedy parsing optimality for dictionary-based text compression.* Theoretical Computer Science, 525 2014, pp. 55–59.
5. M. Crochemore, G. M., A. Langiu, F. Mignosi, and A. Restivo: *Dictionary symbolwise flexible parsing.* Journal of Discrete Algorithms, 14 2012, pp. 74–90.
6. S. DeAgostino: *Greedy versus optimal analysis of bounded size dictionary compression and on-the-fly distributed computing*, in Proceedings Prague Stringology Conference, 2020, pp. 74–83.
7. S. DeAgostino and R. Silvestri: *A worst case analysis of the lz2 compression algorithm.* Information and Computation, 139 1997, pp. 258–268.
8. S. DeAgostino and J. A. Storer: *On-line versus off-line computation in dynamic text compression.* Information Processing Letters, 59 1996, pp. 169–174.
9. A. Farrugia, P. Ferragina, A. Frangioni, and R. Venturini: *Bicriteria data compression*, in Proceedings SIAM-ACM Symposium on Discrete Algorithms (SODA 14), 2014, pp. 1582–1585.
10. P. Ferragina, I. Nitto, and R. Venturini: *On optimally partitioning a text to improve its compression.* Algorithmica, 61 2011, pp. 51–74.

11. P. Ferragina, I. Nitto, and R. Venturini: *On the bit-complexity of lempel-ziv compression.* SIAM Journal on Computing, 42 2013, pp. 1521–1541.
12. D. Kosolobov: *Relations between greedy and bit-optimal lz77 encodings*, in Proceedings Symposium on Theoretical Aspect of Computer Science, 2018, pp. 46:1–46:14.
13. A. Langiu: *On parsing optimality for dictionary-based text compression - the zip case.* Journal of Discrete Algorithms, 20 2013, pp. 65–70.
14. A. Lempel and J. Ziv: *On the complexity of finite sequences.* IEEE Transactions on Information Theory, 22 1976, pp. 75–81.
15. A. Lempel and J. Ziv: *A universal algorithm for sequential data compression.* IEEE Transactions on Information Theory, 23 1977, pp. 337–343.
16. Y. Matias and C. S. Sahinalp: *On the optimality of parsing in dynamic dictionary-based data compression*, in Proceedings SIAM-ACM Symposium on Discrete Algorithms (SODA 99), 1999, pp. 943–944.
17. D. Sheinwald, A. Lempel, and J. Ziv: *On encoding and decoding with two - way head machines.* Information and Computation, 116 1995, pp. 128–133.
18. J. A. Storer: *Data Compression: Methods and Theory*, Computer Science Press, 1988.
19. T. A. Welch: *A technique for high-performance data compression.* IEEE Computer, 17 1984, pp. 8–19.
20. J. Ziv and A. Lempel: *Compression of individual sequences via variable-rate coding.* IEEE Transactions on Information Theory, 24 1978, pp. 530–536.