On the optimality of parsing in dynamic dictionary based data compression

Yossi Matias * Süleyman Cenk Ṣahinalp †

We consider the parsing method to be used in dynamic dictionary based data compression. We show that (1) the commonly used greedy parsing may result in far from optimal compression with respect to the dictionary in use; (2) a one-lookahead greedy parsing scheme obtains optimality with respect to any dictionary construction schemes that satisfy the prefix property; and (3) there is a data structure which enables efficient on-line implementation of this one-lookahead method.

Summary

The most common compression algorithms used in practice are the on-line dictionary schemes. These algorithms are based on maintaining a *dictionary* of strings that are called *phrases*. They *parse* the input string T incrementally, i.e., partition T to non-overlapping substrings that appear in the dictionary $\mathcal{D}$ so that longer prefixes of T are compressed in successive iterations. These substrings are represented in the compressed string T' by the respective dictionary *codewords*.

Most practical compression algorithms use *dynamic* dictionary construction schemes, introduced by Ziv and Lempel [ZL77, ZL78], in which the dictionary is initially empty and is constructed incrementally, in an on-line fashion: as the input is read, some of its substrings are chosen as dictionary phrases and added to the dictionary.

The most popular dictionary based compression algorithms are the LZ78 method [ZL78], its LZW variant [Wel84], and the LZ77 method [ZL77]. Most dynamic schemes, including the above, satisfy the *prefix property*: for any given phrase in the dictionary, all its prefixes are also phrases in the dictionary.

*Department of Computer Science, Tel-Aviv University, Israel, and Information Sciences Research Center, Bell Laboratories. e-mail: matias@math.tau.ac.il. Research supported in part by an Alon Fellowship, by the Israel Science Foundation founded by The Academy of Sciences and Humanities, and by the Israeli Ministry of Science.

†Department of Computer Science, University of Warwick, UK; and Center for BioInformatics, University of Pennsylvania, USA. e-mail: cenk@dcs.warwick.ac.uk. Research supported in part by ESPRIT LTR Project no. 20244 - ALCOM-IT, and NATO Research Grant CRG.972175.

We consider the parsing method that is to be used once the dictionary construction scheme is selected. In particular, given a dictionary construction scheme, we consider whether there is an efficient dynamic parsing method that achieves optimality with respect to this scheme on all input strings. We call a parsing method optimal with respect to a given dictionary construction scheme, if it obtains the smallest number of phrases possible on any input string. Under certain conditions (which apply LZ78, LZW, and other existing schemes) this phrase optimality translates into bit optimality.

For static dictionaries, in which the phrases are fixed before the parsing starts, the issue of parsing optimality has been resolved for prefix closed dictionaries. Greedy parsing is far from optimal: there are strings that can be parsed to m phrases using a given (static) dictionary, for which greedy parsing with the same dictionary obtains $\Omega(m^{3/2})$ phrases [GS85]. On the other hand, finding optimal parsing for prefix-closed dictionaries can be done in linear time.

We show that greedy parsing can be far from optimal for dynamic dictionary construction schemes, by considering the LZW scheme, and showing a gap similar to the static case. We also show that a one-lookahead scheme, denoted as *flexible parsing*, or $\mathcal{FP}$, obtains optimality with respect to any dictionary scheme satisfying the prefix property. We also introduce a data structure which implements $\mathcal{FP}$ for the LZW dictionary in amortized $O(1)$ time per character, and space proportional to the size of the compressed output, which are both optimal.

Below we overview some of our results. More details can be found in [MS97], where we also show that greedy parsing is optimal for dictionaries with the suffix property, and explore the use of k-lookahead schemes ($k > 1$) for parsing optimality in dictionaries without the prefix property. An experimental study of the $\mathcal{FP}$ scheme is given in [MRS98].

Non-optimality of Greedy Parsing.
A greedy parser selects the longest advancing prefix of the uncompressed portion of the text in every iteration. *There are input strings which can be parsed to some $O(\ell)$ phrases, and be represented by $O(\ell \log \ell)$ bits by using the LZW dictionary for which greedy parser ob-*

tains $\Omega(\ell^{3/2})$ phrases and outputs $\Omega(\ell^{3/2}\log \ell)$ bits.

We first construct a string T which uses an arbitrarily large dictionary $\Sigma = \{0, 1, \ldots, k, k+1, k+2, \ldots, k+\sqrt{k}\}$, where k is a prime number. Let R be the substring $1, \ldots, k$, and let R_i denote the concatenation of i copies of the string R. Let S be the substring $1, 2, 1, 2, 3, \ldots, 1, 2, \ldots, k$, and let T be the concatenation of $0, S, k+1, 0, R_1, 1, k+2, 0, R_2, 1, \ldots, k+\sqrt{k}, 0, R_{\sqrt{k-1}}, 1$.

The LZW dictionary scheme first processes the substring S, and inserts the substrings $(01), (12), (21), (123), (31), \ldots, (12\ldots k)$ in $\mathcal{D}$ with respective codewords $k+\sqrt{k}+1, k+\sqrt{k}+2, \ldots, 3k-2$. Then it processes the substrings $(k+i, 0, R_i, 1)$ for $i = 1, \ldots, \sqrt{k-1}$: for each such substring it first inserts in $\mathcal{D}$, $(1(k+i))$, then inserts $((k+i)01\ldots i+1)$, and because k is prime, then inserts all substrings of R_i of size $i+1$. Altogether there will be $k+2$ insertions to $\mathcal{D}$, and hence no more than $(3/2)\log k + O(1)$ bits are required to represent a codeword at any iteration.

For each substring inserted in $\mathcal{D}$, LZW outputs one codeword, hence the total number of codewords output by LZW for T is at least $k^{3/2}$. This implies that the total number of bits it outputs is at least $k^{3/2}\log k$. An optimal parser still obtains $2k-1$ phrases for S; however it obtains only one phrase for every occurrence of R in T. Hence the number of phrases it outputs for each R_i is no more than $i+2$, and the total number of phrases it outputs for T is no more than $3k$, and the total number of bits it outputs is no more than $(9/2)k\log k + O(k)$.

Optimality of $\mathcal{FP}$.
We now turn our attention to $\mathcal{FP}$. Rather than greedily parsing the longest advancing prefix of the uncompressed portion of the text, $\mathcal{FP}$ parses the prefix which results in the longest advancement in the next iteration.
For any dictionary construction scheme which builds a dictionary that satisfies the prefix property at all iterations, flexible parsing obtains the minimum number of phrases out of any input string T.

Let $\mathcal{P}_d$ be a dictionary construction scheme that builds a dictionary satisfying the prefix property at all iterations, and let $\mathcal{C}$ and $\mathcal{C}'$ be the compression algorithms that respectively use $\mathcal{FP}$, and any other parsing scheme together with $\mathcal{P}_d$. Our claim is that the number of codewords output by $\mathcal{C}$ on any given input T is at most that output by $\mathcal{C}'$.

Let the i^{th} phrase of T obtained by $\mathcal{C}$ end at position $E(i)$, and the i^{th} phrase obtained by $\mathcal{C}'$ end at position $E'(i)$. Similarly, let the longest phrase in $\mathcal{D}$ which starts at $E(i)+1$ end at position $L(i)$ and let the longest phrase which starts at $E'(i)+1$ end at position $L'(i)$. We show by induction on i that $L(i) \geq L'(i)$. Notice that $T[E'(i)+1 : L'(i)]$ is a phrase and $E'(i) \leq L'(i-1) \leq L(i-1)$. Hence, because $\mathcal{D}$ is a prefix dictionary, either (1) $T[E(i-1)+1 : E'(i)]$ is a phrase in $\mathcal{D}$, hence by definition of $\mathcal{FP}$, $L(i) \geq L'(i)$, or (2) $E'(i) \leq E(i-1)$, which completes the induction.

Efficient implementation of $\mathcal{FP}$.
There is a data structure which can implement the LZW dictionary construction with $\mathcal{FP}$ in amortized $O(1)$ time per character, using $O(|T'|)$ space.

The data structure maintains the trie, $\mathcal{T}$, of phrases as in the original LZW algorithm; in addition, it also maintains $\mathcal{T}^r$, the compressed trie of the *reverses* of all phrases inserted in the $\mathcal{T}$. (Given a string $S = s_1, s_2, \ldots, s_n$, its reverse S^r is the string $s_n, s_{n-1}, \ldots, s_2, s_1$.) For each node v in $\mathcal{T}$, there is a corresponding node v^r in $\mathcal{T}^r$, linked to v, which represents the reverse of the phrase represented by v. As in the case of the $\mathcal{T}$ alone, the insertion of a phrase S to this data structure takes $O(|S|)$ time. Given a dictionary phrase S, and the node n which represents S in $\mathcal{T}$, one can find out whether the substring obtained by concatenating S with any character a is in $\mathcal{D}$, by checking out if there is an edge from n with corresponding character a in $O(1)$ time. Similarly checking whether $S[2 : |S|]$ is in $\mathcal{D}$ requires $O(1)$ time time by going from n to n', the node representing reverse of S in $\mathcal{T}^r$, and checking if the parent of n' represents $S[2 : |S|]^r$. The total space needed is $O(|\mathcal{D}|) = O(T')$.

References

[GS85] M. E. Gonzales-Smith and J. A. Storer. Parallel algorithms for data compression. *Journal of the ACM*, 32(2):344–373, April 1985.

[MRS98] Y. Matias and N. Rajpoot and S. C. Sahinalp Implementation and experimental evaluation of flexible parsing for dynamic dictionary based data compression. *Workshop on Algorithmic Engineering*, 1998.

[MS97] Y. Matias and S. C. Sahinalp Optimal parsing for dictionary based data compression. Bell Laboratories technical report, June 1997.

[Wel84] T.A. Welch. A technique for high-performance data compression. *IEEE Computer*, pages 8–19, January 1984.

[ZL77] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, IT-23(3):337–343, May 1977.

[ZL78] J. Ziv and A. Lempel. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, IT-24(5):530–536, September 1978.