Selective Weighted Adaptive Coding

Yoav Gross[1], Shmuel T. Klein[2], Elina Opalinsky[1], and Dana Shapira[1]

[1] Dept. of Computer Science, Ariel University, Ariel 40700, Israel
yodgimmel@gmail.com, elinao@ariel.ac.il, shapird@g.ariel.ac.il
[2] Dept. of Computer Science, Bar Ilan University, Ramat Gan 52900, Israel
tomi@cs.biu.ac.il

Abstract. Motivated by improving the processing time of the weighted compressor introduced recently, while only marginally affecting its compression performance, we propose a selective-weighted method that restricts the model based positions and/or the times the model gets updated. That is, the adaptive statistical weight oriented method combines the ability to assign higher priority to symbols that are closer to those being encoded, while extending the model to the probability distribution of symbols that are not necessarily located in consecutive positions. Several variants for selecting the representative positions for determining the model are suggested, and all are empirically shown to fulfill both objectives for small skips.

1 Introduction

The most noticeable advantage of *adaptive* compression techniques is their ability to adjust the encoding to the local changes in the input file. Static methods, on the other hand, such as Huffman coding [9] or Elias' universal codes [3], use the same set of codewords throughout the entire file, and the main way to adapt the encoding to the given input is via a preprocessing stage.

Statistical dynamic methods, like dynamic Huffman [14] or adaptive arithmetic coding [15], attempt to adjust to the local probability distribution by collecting statistics of the already processed portion of the file. Traditional statistical adaptive compression algorithms encode the next to be processed character according to the current model, and then update the model by incrementing the frequency of the currently read symbol. These algorithms are usually heuristics that are based on the following two strategies:

- all positions in the input file are treated equally, that is, with no consideration for their distance from the currently processed symbol;
- the distribution of the elements in *consecutive positions* in a "sliding window", just preceding the current processed character, is used to estimate the distribution of the elements still to come.

In this paper we suggest to use different policies to determine the varying probability distributions, first, by giving higher priority to symbols that are closer to those being encoded, second, by using the frequency counts of symbols that are not necessarily located in consecutive positions. The goal of our first policy is to improve the compression efficiency, while the goal of the second is to enhance the processing times while only marginally affecting the compression performance. We refer to this new method as selective-weighted.

Unlike the uniform treatment of symbols in different locations of the file, a new approach is taken in the *weighted* dynamic compression method suggested in [5], which assigns higher priority to closer to be encoded symbols by means of an increasing

Yoav Gross, Shmuel T. Klein, Elina Opalinsky, Dana Shapira: Selective Weighted Adaptive Coding, pp. 97–106.
Proceedings of PSC 2023, Jan Holub and Jan Žďárek (Eds.), ISBN 978-80-01-07206-6

weight function. The weighted method is especially suited for the encoding of files with locally skewed distributions. We distinguish between two weighted schemes, a *forward weighted* coding [7], and a *backward weighted* coding [5]. The weights assigned to the positions by the former scheme are generated by a non increasing function f, and the weight for each symbol σ is the sum of the values of f over all the positions where σ occurs in the portion of the input file that is still to be encoded. The latter is, in fact, a heuristic defined similarly to the forward-weighted distribution, but it is calculated over positions that have already been processed. As a result, in the forward approach, there is a need to transmit the exact character frequencies to the decoder, for example in a header, while for the backward approach, these frequencies can be learned incrementally and need not to be given. Empirical results have shown that backward weighted techniques can improve beyond the lower bound given by the entropy for static encoding.

An unweighted forward-looking dynamic algorithm has been suggested in [11]; it uses the true distribution of the characters within the remaining portion of the file by decrementing the frequency counts, rather than incrementing the frequency count of the current processed element as is done in the standard, backward looking, adaptive coding techniques. A bidirectional method, which combines both traditional and forward-looking methods, has then been proposed in [6] and the frequencies of the elements are transmitted progressively, whenever a new symbol is encountered, rather than as a bulk, in a header of the file. The combination of the weighted algorithm with PPM has been studied in [1].

The idea of the backward dynamic methods is based on the assumption that the more data is collected in the already processed portion of the input file, the better can one predict the corresponding probability distribution. However, this assumption is not necessarily true, and the distribution over a *subset* of the elements of the file can at times serve as a good approximation for further encodings [10] as shown in our following experiment.

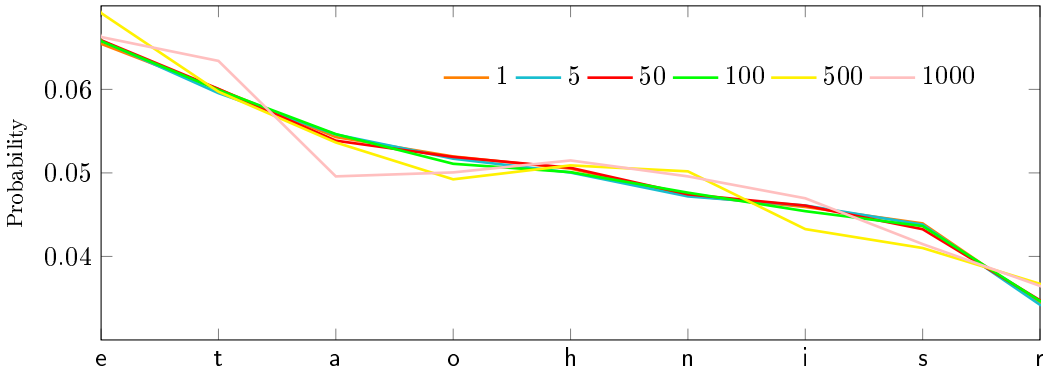

Figure 1. The probabilities of some of the most frequent letters of the file english, in positions with skips with various skip sizes.

We considered the file english, taken from the Pizza & Chili corpus[1], and sorted its letters in non decreasing order of their frequencies. The most frequent letters in this file are, in order, `\b`, `e`, `t`, `a`, `o`, `h`, `n`, `i`, `s`, `r`, `d`, ..., where `\b` stands for blank. Figure 1 depicts on the y-axis the probability of occurrence of the letters that appear on the x-axis, given in this order. The probabilities are based on occurrences of all characters

[1] http://pizzachili.dcc.uchile.cl/

(gap size 1), or on selected subsets of positions, choosing the characters with gap sizes of 5, 50, 100, 500 and 1000 characters. As can be seen, there are obviously fluctuations, but the general forms of the distribution graphs remain similar, even for quite sparse subsets of the inspected positions within the file. The impact on the corresponding codewords lengths in a Huffman or other encoding will even be smaller.

Basing the encoding of the current character on non-consecutive positions in the already processed portion of the file might be straightforward for homogeneous files, but one must carefully avoid some extreme cases. For example, a skip size equal to the fixed length of a line in a text document, or a fixed length field in a database, could result in a completely biased alphabet. We therefore also suggest varying skip values.

Our paper is constructed as follows. In Section 2 we briefly recall the details of the backward weighted compression scheme and propose several selection algorithms. Section 3 presents our experimental results.

2 The Selective Weighted Variants

We concentrate on the *backward weighted* variant, a special case of weighted coding that considers all the positions that have already been processed. Let $T = x_1 \cdots x_n$ be an input file of size n over an alphabet Σ, and assume we have already processed the prefix of T up to position $i-1$ of T, and are about to encode x_i. Given is a function g, $g : [1,n] \longrightarrow \mathbb{R}^+$, which assigns a *non-negative* real number to each position $i \in [1,n]$ within T. A weight $W(g,\sigma,i)$ based on g is defined for each symbol $\sigma \in \Sigma$ and every position i, $i \in [1,n]$, as the sum of the values of the function g for all positions in the prefix $[1,i-1]$ at which σ occurs. Formally,

$$W(g,\sigma,i) \;=\; \sum_{\{j \;\mid\; 1 \le j \le i-1 \;\wedge\; x_j = \sigma\}} g(j).$$

The classic non-weighted adaptive backward compression algorithms, e.g., adaptive arithmetic coding and the one-pass methods based on Huffman coding of the FGK algorithm by Faller [4], Gallager [8] and Knuth [12] and the enhanced algorithm by Vitter [14], are the special case in which g is the constant function $g = \mathbb{1} \equiv g(i) = 1$ for all i.

A simple weighted adaptive coding introduced and named b-2 in [7], divides all the frequencies by 2 at the end of every block of k characters, for some given parameter k, so that the occurrences of characters at the beginning of T contribute to W less than those closer to the current position. Furthermore, all positions within the same block contribute equally to W, and their weights are twice as large as those assigned to the indices in the preceding block. Therefore, the corresponding function g, denoted by $g_{\text{b-2}}$, maintains the equality $g_{\text{b-2}}(i+k) = 2g_{\text{b-2}}(i)$, for each pair of indices i and $i+k$.

Another family of weighted coding schemes, named b-w, is based on the function $g_{\text{b-w}}(i) = (\sqrt[k]{2})^{i-1}$ for $i \ge 1$, for a given parameter k. As for b-2, the function $g_{\text{b-w}}$ still provides a fixed ratio of 2 between blocks but with rather smoother differences at the block borders.

Table 1 compares the classic backward coding, denoted by b-adp, with b-w, on the running example $T = x_1 \cdots x_{12} =$ dbcabcbcaaaa over the alphabet $\Sigma =$ {a, b, c, d}. The table presents for each method the following information:

– $g(i)$: the value of g for position i, of the specific method;

- $W(g, x_i, i)$: the specific weight of the character $\sigma = x_i$ up to (and not including) the column i of the table; that is, the sum of W for those indices $j < i$ at which the character σ occurs, including the initial 1 values.
- $CW[1, i]$: the cumulative W weights for all the characters $\sigma \in \Sigma$ up to (and not including) the column i;
- p_i: the ratio of $W(g, x_i, i)$ and $CW[1, i]$;
- IC: the *Information content*, $-\log p_i$ bits, for each position i.

For b-adp, the values of $g(i)$ are just 1 for every i, whereas for b-w they are $(\sqrt{2})^{i-1}$, taking $k = 2$. The sum of the IC values is a lower bound on, and can be used as an estimate of, the storage requirement by the corresponding method, since it can be closely approximated by arithmetic coding. This sum is 25.588 bit and 24.447 bit for b-adp and b-w, respectively.

	i	1	2	3	4	5	6	7	8	9	10	11	12
	T	d	b	c	a	b	c	b	c	a	a	a	a
b-adp	$g(i)$	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
	W	1.000	1.000	1.000	1.000	2.000	2.000	3.000	3.000	2.000	3.000	4.000	5.000
	CW	4.000	5.000	6.000	7.000	8.000	9.000	10.000	11.000	12.000	13.000	14.000	15.000
	p_i	0.250	0.200	0.167	0.143	0.250	0.222	0.300	0.273	0.167	0.231	0.286	0.333
	IC	2.000	2.322	2.585	2.807	2.000	2.170	1.737	1.874	2.585	2.115	1.807	1.585
b-w	$g(i)$	1.000	1.414	2.000	2.828	4.000	5.657	8.000	11.314	16.000	22.627	32.000	45.255
	W	1.000	1.000	1.000	1.000	2.414	3.000	6.414	8.657	3.828	19.828	42.456	74.456
	CW	4.000	5.000	6.414	8.414	11.243	15.243	20.899	28.899	40.213	56.213	78.841	110.841
	p_i	0.250	0.200	0.156	0.119	0.215	0.197	0.307	0.300	0.095	0.353	0.539	0.672
	IC	2.000	2.322	2.681	3.073	2.219	2.345	1.704	1.739	3.393	1.503	0.893	0.574

Table 1. *Classic* b-adp *algorithm vs.* b-w *for the example* $T = x_1 \cdots x_{12} =$ dbcabcbcaaaa.

Motivated by trying to enhance the processing times, even at the price of possibly reduced compression efficiency, we suggest a new encoding scheme based on a periodic selection process, which is controlled by a skip-function f. In a first stage we consider only the special case in which the skip-function is a constant c, and the model gets updated every $f(s) = c$ characters. We distinguish between two different strategies.

(a) The complete-selective algorithm uses the entire input file to compute the probability distributions, as usually done in adaptive methods, but updates the model only every $f(s)$ characters.
(b) The subset-selective algorithm encodes the entire input file T based on the probability distributions of characters appearing at positions selected according to $f(s)$. That is, it only uses a sub-sequence of the input file to determine the model for the encoding of the entire file.

Algorithm 1 brings the formal descriptions of the encoding procedures for both COMPLETE and SUBSET. The only difference is the addition of zeroing the $g(i)$ function in the last lines for the latter. Thereby, the model is updated at steps indexed by $f(s)$, where s is the number of updates so far. While the COMPLETE variant remembers all the changes from the last update, the SUBSET variant skips over the non-selected values. Decoding is just the reverse process.

Table 2 and 3 continue our running example with $s = 3$, the first for COMPLETE and the second for SUBSET. For the first, the sum of the IC values is 24.564 for b-adp

Algorithm 1: COMPLETE (SUBSET) SELECTIVE

COMPLETE (SUBSET)-SELECTIVE $(T = x_1 \cdots x_n,\ g,\ f)$

```
1 s ← 0; last ← 0 ; Initialize the model according to the uniform distribution on Σ
2 for i ← 1 to n do
3     encode x_i according to the current model
4     if i − last = f(s) then
5         update the model according to the distribution of the characters in Σ, given by the
            probabilities {W(g, σ, i+1)/CW[1, i+1]}_{σ∈Σ}
6         s ← s + 1
7         last ← i
8     else
9         g(i) ← 0
```

i		1	2	3	4	5	6	7	8	9	10	11	12
T		d	b	c	a	b	c	b	c	a	a	a	a
b-adp	$g(i)$	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
	W	1.000	1.000	1.000	1.000	2.000	2.000	3.000	3.000	2.000	3.000	3.000	3.000
	CW	4.000	4.000	4.000	7.000	7.000	7.000	10.000	10.000	10.000	13.000	13.000	13.000
	p_i	0.250	0.250	0.250	0.143	0.286	0.286	0.300	0.300	0.200	0.231	0.231	0.231
	IC	2.000	2.000	2.000	2.807	1.807	1.807	1.737	1.737	2.322	2.115	2.115	2.115
b-w	$g(i)$	1.000	1.414	2.000	2.828	4.000	5.657	8.000	11.314	16.000	22.627	32.000	45.255
	W	1.000	1.000	1.000	1.000	2.414	3.000	6.414	8.657	3.828	19.828	19.828	19.828
	CW	4.000	4.000	4.000	8.414	8.414	8.414	20.899	20.899	20.899	56.213	56.213	56.213
	p_i	0.250	0.250	0.250	0.119	0.287	0.357	0.307	0.414	0.183	0.353	0.353	0.353
	IC	2.000	2.000	2.000	3.073	1.801	1.488	1.704	1.272	2.449	1.503	1.503	1.503

Table 2. Complete Selective *with* $s = 3$ *for* b-adp *and* b-w *on the running example* $T = x_1 \cdots x_{12} =$ dbcabcbcaaaa.

and 22.296 for b-w. For the second, $g(i)$ is assigned 0 at every position that is not a multiple of 3, which yields a sum of IC values of 23.558 for b-adp and 21.792 for b-w. These examples show that there are special cases for which even the compression efficiency may improve.

The following experiment is based on defining the distance separating consecutive choices in the selective approach by a varying function. We have to balance between the following, opposing, requirements.

1. On the one hand, the weighted approach calls for giving priority to positions close to the one currently processed. This would imply that the selected elements should be denser at the end than at the beginning of the already treated prefix of the file.
2. On the other hand, there is a need to attain as soon as possible a critical mass of selected items, from which a reliable estimate of the true probability distribution may be derived. It is therefore at the beginning of the file that the selected items should be more frequent.

The first option is hard to implement, because the file is processed progressively. We therefore opt for the second one, and try to control the density of the selected items by choosing different parameters for the skip function.

Our next suggestion is a selective method which is tuned by the function g of the weights. The intuition is that the model should not be updated as long as no

	i	1	2	3	4	5	6	7	8	9	10	11	12
	T	d	b	c	a	b	c	b	c	a	a	a	a
b-adp	$g(i)$	0.000	0.000	1.000	0.000	0.000	1.000	0.000	0.000	1.000	0.000	0.000	1.000
	W	1.000	1.000	1.000	1.000	1.000	2.000	1.000	3.000	1.000	2.000	2.000	2.000
	CW	4.000	4.000	4.000	5.000	5.000	5.000	6.000	6.000	6.000	7.000	7.000	7.000
	p_i	0.250	0.250	0.250	0.200	0.200	0.400	0.167	0.500	0.167	0.286	0.286	0.286
	IC	2.000	2.000	2.000	2.322	2.322	1.322	2.585	1.000	2.585	1.807	1.807	1.807
b-w	$g(i)$	0.000	0.000	2.000	0.000	0.000	5.657	0.000	0.000	16.000	0.000	0.000	45.255
	W	1.000	1.000	1.000	1.000	1.000	3.000	1.000	8.657	1.000	17.000	17.000	17.000
	CW	4.000	4.000	4.000	6.000	6.000	6.000	11.657	11.657	11.657	27.657	27.657	27.657
	p_i	0.250	0.250	0.250	0.167	0.167	0.500	0.086	0.743	0.086	0.615	0.615	0.615
	IC	2.000	2.000	2.000	2.585	2.585	1.000	3.543	0.429	3.543	0.702	0.702	0.702

Table 3. Subset Selective *with* $s = 3$ *for* b-adp *and* b-w *on the running example* $T = x_1 \cdots x_{12} =$ dbcabcbcaaaa.

significant mass of weights has been accumulated that can modify it. We only update the model when the ratio of the sum of the weights since the last update, to the total sum of the weights in the model, crosses a certain threshold, as shown in Algorithm 3. The tuned algorithm provides a single method that assigns a decreasing number of updates for b-adp while at the same time it is almost equivalent to a selection with fixed intervals for b-w. The different behaviour of b-adp and b-w by the tuned selection can be seen in Figure 2, that plots the indices where the model updates have been performed for *threshold* = 1.

Algorithm 2: TUNED SELECTIVE

TUNED-SELECTIVE($T = x_1 \cdots x_n$, g, *threshold*)

1 $cum \leftarrow 0$
2 **for** $i \leftarrow 1$ **to** n **do**
3 encode x_i according to the current model
4 $cum \leftarrow cum + g(i)/\big(g(1) + \cdots + g(i-1)\big)$
5 **if** $cum \geq threshold$ **then**
6 update the model by the probability distribution, at position i, of the characters in Σ: $\big\{W(g, \sigma, i+1)/CW[1, i+1]\big\}_{\sigma \in \Sigma}$
7 $cum \leftarrow 0$

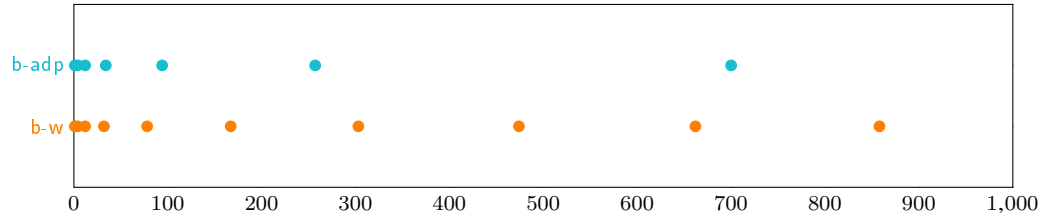

Figure 2. The different behaviour of b-adp and b-w by the tuned selection.

While the tuned selective process is controlled solely by the weight function g, we might wish to modify the selection pace via a chosen parameter. Let $f(j)$ be a function describing the distance from the j-th selected location to the following one. We shall explore the functions $f(j) = j^{\alpha}$, rounded to the nearest integer, for various values of the parameter α. Choosing $\alpha = 1$ would imply a linear increase in the distance between consecutive selected points. If s locations are selected in the prefix of size i of the text, one gets that $\sum_{j=1}^{s} j$ must be bounded by i, so that $s \leq \sqrt{2i}$. For general α, the corresponding bound is

$$i \geq \sum_{j=1}^{s} j^{\alpha} \simeq \int^{s} x^{\alpha} dx \simeq \frac{1}{\alpha+1} s^{\alpha+1}$$

from which one can derive $s \leq \left\lceil (\alpha+1)i \right\rceil^{\frac{1}{\alpha+1}}$. Table 4 shows the first few selected indices for various values of α, where the line for $\alpha = 0$ has been grayed as this is the special case with no selection at all, that is, all the positions are chosen. The values for $\alpha = 1$ are emphasized, and the corresponding method appears in the experimental results as incremental. The column headed s shows the number of selected positions for a text of size $n = 1000$.

α	s	indices of selected points
0	1000	1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
0.25	300	1 2 3 4 5 7 9 11 13 15 17 19 21 23 25 27 29
0.5	131	1 2 4 6 8 10 13 16 19 22 25 28 32 36 40 44 48
0.75	71	1 3 5 8 11 15 19 24 29 35 41 47 54 61 69 77 85
1	**45**	**1 3 6 10 15 21 28 36 45 55 66 78 91 105 120 136 153**
1.25	31	1 3 7 13 20 29 40 53 69 87 107 129 154 181 211 243 278
1.5	23	1 4 9 17 28 43 62 85 112 144 180 222 269 321 379 443 513

Table 4. *Sample of selected indices for various values of* α.

3 Experimental Results

In order to evaluate the selective methods, we have considered several datasets downloaded from the Pizza & Chili Corpus and report our outcomes here on only two representative files.

- *english* – a concatenation of English texts from the Gutenberg Project;
- *dna* – a sequence of gene DNA sequences obtained from the Gutenberg Project.

As mentioned above, the weighted approach is especially suitable for the encoding of files with locally skewed distributions and is most effective when it is applied on files that have been pre-processed by the *Burrows-Wheeler Transform* (BWT) [2]. To improve the time complexity of this transformation via the use of a suffix array [13], BWT is applied in blocks. We therefore consider only a 4M prefix of the above files. The experiments were conducted on a machine running 64 bit Windows 10 with an Intel Core i5-8250 @ 1.60GHz processor, 6144K L3 cache, and 8GB of main memory.

The plots in Figure 3 summarize the experiments, with those on the left corresponding to english and those on the right to dna. They show the compression ratio

(size of the compressed file divided by the size of the original) and encoding and decoding times in seconds, as a function of the skip size $f(s)$ between consecutive selected positions. This skip size is constant for the basic complete and subset strategies, and represents the *average* interval size for those with varying distances, labelled tuned, α and incremental, which is the special case $\alpha = 1$. As a benchmark, we also added the values of gzip.

As can be seen, there is a slight increase in the size of the compressed file with growing $f(s)$, that is, when the selected positions become sparser, with almost no difference on the performance of the different methods keeping the full statistics on the dna input, and slowly diverging values on the english file. The loss of compression efficiency is more accentuated for the subset approach, which can be explained by the fact that it did not accumulate enough data to get reliable estimates. Note that for these examples, the compression is still better than that of gzip.

In parallel to the slight loss in compression efficiency, there is, with increasing skip size $f(s)$, a significant improvement in both encoding and decoding times, again with similar performance for all the methods, except that based on the selection of a subset, for which the gain in execution time is even stronger. None of these times are comparable with the performance of the highly optimized gzip.

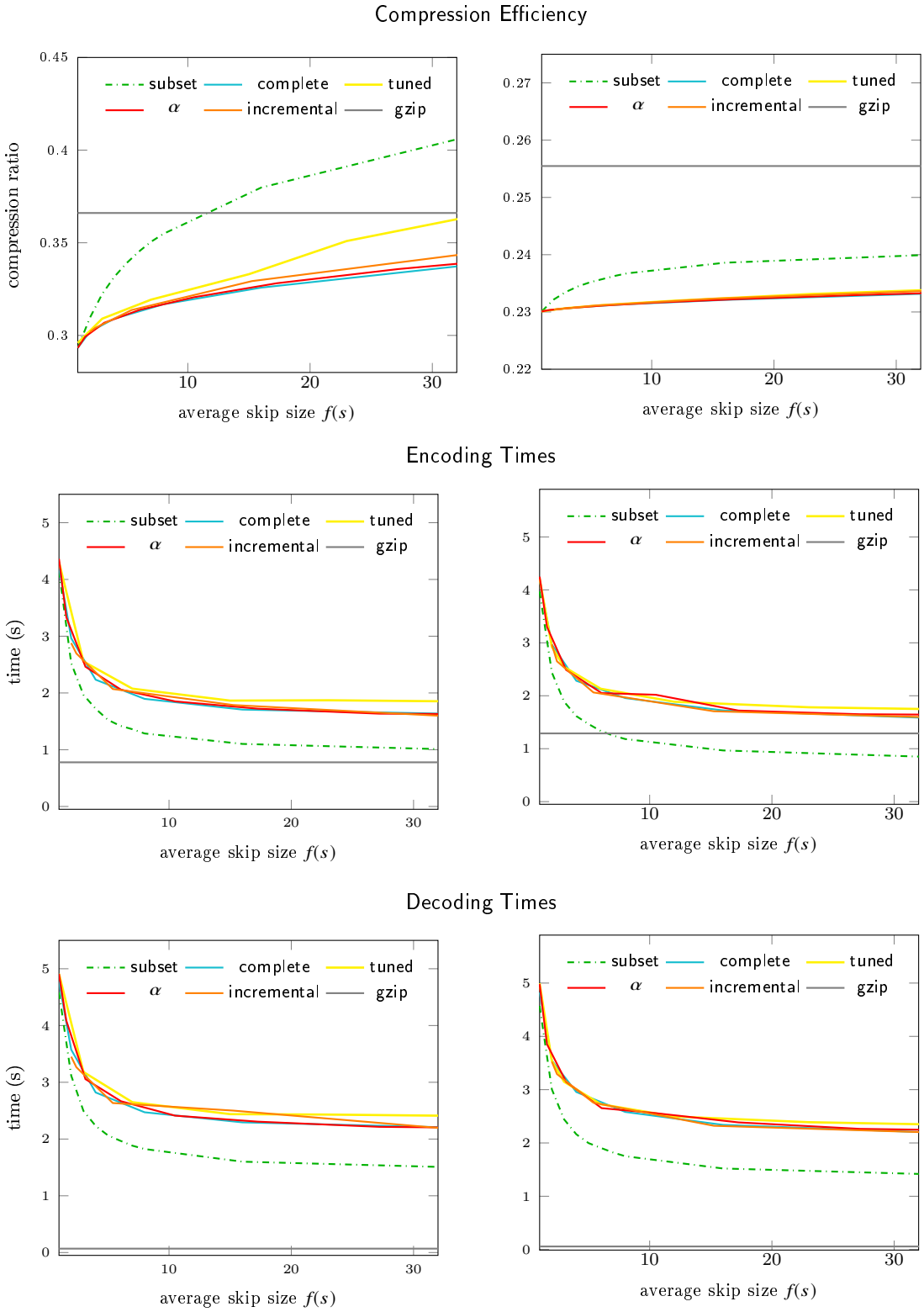

Figure 3: Experimental results for selective variants of b-w as a function of the average skip size $f(s)$ on the 4M *english* and *dna* BWT transformed files. The compression ratio and encoding and decoding times in seconds on the 4M *english* (left) and *dna* (right) BWT transformed files, as a function of the skip size $f(s)$ between consecutive selected positions. This skip size is constant for the basic complete and subset strategies, and represents the *average* interval size for those with varying distances, labelled tuned, α, incremental ($\alpha = 1$) and gzip.

4 Conclusion

We extended the recently introduced weighted adaptive compression paradigm to variants basing the model, on the basis of which the encoding is derived, on various selective approaches. Our empirical tests indicate that the time performance can be significantly improved by the selective methods, while only marginally affecting the compression.

References

1. R. M. Avrunin, S. T. Klein, and D. Shapira: *Combining forward compression with PPM.* SN Comput. Sci., 3(239) 2022.
2. M. Burrows and D. J. Wheeler: *A block-sorting lossless data compression algorithm*, Tech. Rep. 124, Digital Equipment Corporation, 1994.
3. P. Elias: *Universal codeword sets and representations of the integers.* IEEE Trans. Information Theory, 21(2) 1975, pp. 194–203.
4. N. Faller: *An adaptive system for data compression*, in Record of the 7-th Asilomar Conference on Circuits, Systems and Computers, 1973, pp. 593–597.
5. A. Fruchtman, Y. Gross, S. T. Klein, and D. Shapira: *Backward weighted coding*, in 31st Data Compression Conference, DCC 2021, Snowbird, UT, USA, March 23-26, 2021, IEEE, 2021, pp. 93–102.
6. A. Fruchtman, Y. Gross, S. T. Klein, and D. Shapira: *Bidirectional adaptive compression.* Discret. Appl. Math., 330 2023, pp. 40–50.
7. A. Fruchtman, Y. Gross, S. T. Klein, and D. Shapira: *Weighted Burrows-Wheeler compression.* SN Comput. Sci., 4(265) 2023.
8. R. Gallager: *Variations on a theme by Huffman.* IEEE Transactions on Information Theory, 24(6) 1978, pp. 668–674.
9. D. A. Huffman: *A method for the construction of minimum-redundancy codes.* Proceedings of the IRE, 40(9) 1952, pp. 1098–1101.
10. S. T. Klein, E. Opalinsky, and D. Shapira: *Selective dynamic compression*, in Proceedings of the Prague Stringology Conference, Czech Technical University in Prague, Czech Republic, 2019.
11. S. T. Klein, S. Saadia, and D. Shapira: *Forward looking Huffman coding.* Theory of Computing Systems, 2020, pp. 1–20.
12. D. E. Knuth: *Dynamic Huffman coding.* Journal of Algorithms, 6(2) 1985, pp. 163–180.
13. U. Manber and E. W. Myers: *Suffix arrays: A new method for on-line string searches.* SIAM J. Comput., 22(5) 1993, pp. 935–948.
14. J. S. Vitter: *Design and analysis of dynamic Huffman codes.* JACM, 34(4) 1987, pp. 825–845.
15. I. H. Witten, R. M. Neal, and J. G. Cleary: *Arithmetic coding for data compression.* Commun. ACM, 30(6) 1987, pp. 520–540.