Wavelet Trees: from Theory to Practice

Roberto Grossi
Dipartimento di Informatica
Università di Pisa, Italy
grossi@di.unipi.it

Jeffrey Scott Vitter Bojian Xu
Department of Electrical Engineering & Computer Science
The University of Kansas, USA
{*jsv, bojianxu*}*@ku.edu*

***Abstract*—The *wavelet tree* data structure is a space-efficient technique for rank and select queries that generalizes from binary characters to an arbitrary multicharacter alphabet. It has become a key tool in modern full-text indexing and data compression because of its capabilities in compressing, indexing, and searching. We present a comparative study of its practical performance regarding a wide range of options on the dimensions of different coding schemes and tree shapes. Our results are both theoretical and experimental: (1) We show that the run-length δ coding size of wavelet trees achieves the 0-order empirical entropy size of the original string with leading constant 1, when the string's 0-order empirical entropy is asymptotically less than the logarithm of the alphabet size. This result complements the previous works that are dedicated to analyzing run-length γ-encoded wavelet trees. It also reveals the scenarios when run-length δ encoding becomes practical. (2) We introduce a full generic package of wavelet trees for a wide range of options on the dimensions of coding schemes and tree shapes. Our experimental study reveals the practical performance of the various modifications.**

***Keywords*-wavelet tree; full-text indexing; pattern matching; data compression**

I. Introduction

The field of compressed full-text indexing [1] involves the design of indexes that support fast full-text pattern matching using limited amount of space. In particular, the goal is to have an index whose size is roughly equal to the size of the text in compressed format, with search performance comparable to the one achieved by uncompressed methods such as suffix trees and suffix arrays. Some compressed data structures are in addition *self-indexing* in that the data structure encodes the original text and can quickly recreate any portion of it in a random access manner, and thus the original text can be discarded and replaced by only the index.

Many of the compressed indexing techniques developed in the last decade made use of *wavelet trees* [2], used in conjunction with the text's Ψ decomposition [3] or the Burrows-Wheeler transform (BWT) [4], [5]. Conceptually, the wavelet tree in [2] can be most simply described as a full (and often balanced) binary tree (Figure 1). The root node is a bit array of the same length of the input text and partitions the text's alphabet in two sets, where a 0-bit indicates the corresponding text character is in the left set and a 1-bit indicates that the text character is in the right set. Such bit array representation happens recursively at each internal node, where the text at the internal node is of the characters dispatched from the parent node with their order in the text preserved. Each leaf node represents a distinct character in the alphabet of the input text. Wavelet trees cleverly decompose the text into a hierarchy of bit arrays without introducing any redundancy, so that the total size of the raw bit arrays in the wavelet tree is exactly the same as the size (in bits) of the input text, regardless of the shape of the wavelet tree. If the bit arrays at each node are 0th-order entropy compressed, the resulting space usage of the wavelet tree is equal to the 0th-order entropy-compressed size of the input text, again regardless of the shape of the tree [2], [6].

Definition 1. *Given a text T of size n from alphabet $\Sigma = \{1, 2, \ldots, \sigma\}$, the 0th-order empirical entropy of T is $H_0(T) = \sum_{i \in \Sigma} (n_i/n) \log(n/n_i)$[1], where n_i is the number of occurrences of character i in T.*

Let $\mathcal{B}^i$ $(i = 1, 2, \ldots, t = \sigma - 1)$ denote the raw bit arrays in the level-order traversal of the wavelet tree of T, and $|\mathcal{B}^i|$ denote the number of bits contained in $\mathcal{B}^i$.

Fact 2 ([2])**.** $nH_0(T) = \sum_{i=1}^{t} |\mathcal{B}^i| H_0(\mathcal{B}^i)$.

Wavelet trees support several useful queries, such as member (reporting the character at a given text position), rank (reporting the number of occurrences of a given character in a given prefix of the text), and select (reporting the position of a given occurrence of a given character), generalizing dictionaries from the binary alphabet [7], [8] to arbitrary multicharacter alphabets; the running time per query is $O(\log \sigma)$, provided constant-time responses to member, rank, and select queries on the bit arrays and a (nearly) balanced wavelet tree. When leaves are arranged in some canonical order such as the alphabetical order, wavelet trees support 2-d 4-sided range queries, which report the positions of the characters in a given range of the alphabet and in a given range of text locations. If the alphabet size is within a polylog factor of the text size,

The work of the first author was partially supported by Italian project PRIN MAINSTREAM of MIUR, and that of the last two authors was supported in part by National Science Foundation grant CCF-1017623.

[1]In this paper, we use log to denote $\log_2$.

all the aforementioned queries can be answered in constant time [9]. A single wavelet tree over the Burrows-Wheeler transform of a text with proper 0th-order entropy coding for the bit arrays produces a compressed full-text index, whose size is the same as the high-order entropy-compressed size of the text [10], [6], [11]. Wavelet trees also work on external memory by increasing the branching factor at the tree nodes to correspond to page sizes [12], [13]. Due to its capabilities in compressing, indexing, and searching, wavelet tree has become a key tool in modern full-text compressed indexing [2], [5], [14], [10], [15], [11], [9], [16] and data compression [6], [10], [15], [11], [17]. Besides the aforementioned set of results, the recent work [6] anticipated the open problem of analyzing the compression performance of the wavelet trees using run-length (or gap) δ encoding and studying the influence of the tree shape on the compression performance. In this paper, we partially complement this need by the following results:

- We study the scenarios where run-length δ encoding becomes better in wavelet tree compression. We show that if the alphabet size is bounded by a constant and the 0-order empirical entropy of the string is asymptotically less than the logarithm of the alphabet size, the run-length δ-encoded wavelet trees achieve the 0-order empirical entropy compression of the string with leading constant 1, whereas under the same condition the run-length γ coding achieves the same bound but with the best known leading constant 2. We recall that for arbitrary strings, the run-length γ encoded wavelet trees are more space saving than the run-length δ encoded wavelet trees, as was empirically observed in [15].
- We introduce a generic template-based package library of wavelet trees with a wide range of options on the coding schemes and tree shapes (summing up to 33 options). Our experimental study not only validates the above theoretical result on the compression performance of the run-length δ-encoded wavelet tree, but more importantly, it reveals the practical performance of all the relevant options, providing users a guide in choosing the appropriate type of wavelet trees based on the theme of their data. Although the implementation of some of the options are indeed available in the context of compressed full-text indexes such as those in the Pizza&Chili Corpus[2], as far as we know, this is the first package to provide a large set of options living in the same independent library.

II. Run-length δ-Encoded Wavelet Trees

The γ code [18] of a positive integer i, denoted as $\gamma(i)$, uses $2\lfloor \log i \rfloor + 1$ bits: $\ell = \lfloor \log i \rfloor + 1$ bits to represent the unary of $\lfloor \log i \rfloor$, followed by anther $\lfloor \log i \rfloor$ bits to represent the binary of i with the most significant bit removed. The

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

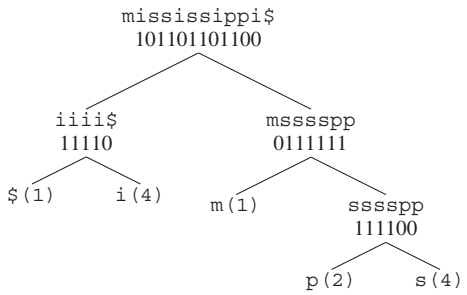


Figure 1. An example of a standard wavelet tree for text mississippi$ with uncompressed raw bit arrays. The text at each internal node is split into two subtexts by (nearly) evenly dividing the local alphabet into two. The texts shown at internal nodes are only for illustration purpose and are not actually stored. The numbers at leaf nodes represent the number of occurrences of the corresponding character in the original text.

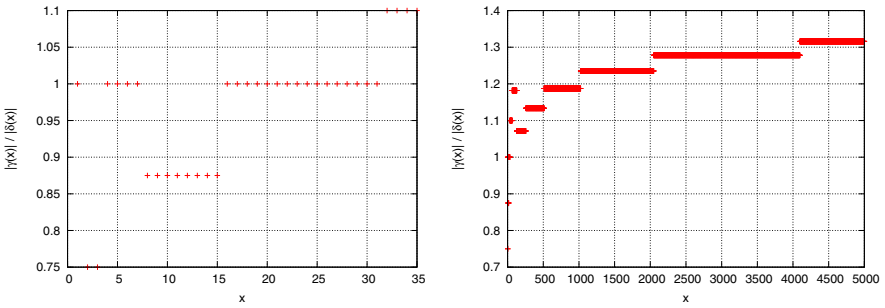
δ code [18] of a positive integer i, denoted as $\delta(i)$, uses $\lfloor \log i \rfloor + 2\lfloor \log(\lfloor \log i \rfloor + 1) \rfloor + 1$ bits: $2\lfloor \log \ell \rfloor + 1 = 2\lfloor \log(\lfloor \log i \rfloor + 1) \rfloor + 1$ bits to represent the γ code of the length $\lfloor \log i \rfloor + 1$ of the binary representation of i, followed by another $\lfloor \log i \rfloor$ bits to represent the binary of i with its most significant bit removed. Both γ and δ codes are prefix-free and can be uniquely decoded. Let $B = B[1 \dots n]$ be a binary string of size n. String B can be viewed as a sequence of maximal runs of identical bits $B = b_1^{\ell_1} b_2^{\ell_2} \cdots b_m^{\ell_m}$ for some $m \leq n$, where $b_i \neq b_{i+1}$ for $1 \leq i < m$. The run-length δ encoding of string B is the binary string $B_{rle,\delta} = b_1 \delta(\ell_1)\delta(\ell_2)\cdots\delta(\ell_m)$ where b_1 is necessary for decoding. The run-length δ encoded wavelet trees are the ones whose bit arrays are run-length δ encoded. The run-length γ-encoded wavelet trees can be similarly obtained by having all the wavelet tree bit arrays run-length γ encoded. Run-length δ encoding is less space efficient than run-length γ encoding in practice, because $\delta(i)$ consumes more bits than $\gamma(i)$ does when i is small (Figure 2), which is often the case in the wavelet tree compression for the real-world data [15]. In this section, we reveal the scenarios when run-length δ encoding becomes more space efficient than run-length γ encoding in wavelet tree compression. We want to note that all the proofs in this section are considered for the asymptotic case, when the text size n goes to infinite.

A. Binary string run-length δ encoding—achieving H_0

Let t be the number of the occurrences of the least frequent bit in B. Let $m' \leq m$ be the number of maximal runs in B that have size larger than 1. We use the following results from [6]:

Lemma 3 ([6]). *(1)* $nH_0(B) \geq t \log(n/t) + t$*; (2)* $m \leq 2t + 1$*; (3)* $m' \leq t + 1$.

Run-length γ encoding is suboptimal in compressing bit arrays, namely, $|B_{rle,\gamma}| \leq 4nH_0(B)+4$ (Lemma 3.3 in [6]), which we can further improve to be $|B_{rle,\gamma}| \leq 2nH_0(B) + 2\log n + 2$ (Lemma 10 in the Appendix). In this section, we



(a) Over a narrow range of x (b) Over a wide range of x

Figure 2. $|\delta(x)| < |\gamma(x)|$ for $x > 31$; $|\delta(x)| = |\gamma(x)|$ for $x \in \{1, 4\text{–}7, 16\text{–}31\}$; $|\delta(x)| > |\gamma(x)|$ for $x \in \{2\text{–}3, 8\text{–}15\}$.

show that although run-length δ encoding is also suboptimal for arbitrary bit arrays, it can reach the optimality when $H_0(B)$ is small enough. We first show the following result.

Lemma 4. $|B_{rle,\delta}| \leq 3nH_0(B)+2\log(\log n+1)+\log n+2$.

Proof: By definition,

$$\begin{aligned}
|B_{rle,\delta}| &= 1+\sum_{i=1}^{m}|\delta(\ell_i)| \\
&= 1+\sum_{i=1}^{m}(\lfloor\log\ell_i\rfloor+2\lfloor\log(\lfloor\log\ell_i\rfloor+1)\rfloor+1) \\
&\leq 1+\sum_{i=1}^{m}(\log\ell_i+2\log(\log\ell_i+1)+1) \\
&= \sum_{\ell_i>1}(\log\ell_i+2\log(\log\ell_i+1))+m+1 \\
&\leq m'\log\frac{n}{m'}+2m'\log\left(\log\frac{n}{m'}+1\right)+m+1 \\
&\leq (t+1)\log\frac{n}{t+1}+2(t+1)\log\left(\log\frac{n}{t+1}+1\right) \\
&\quad +2t+2
\end{aligned}$$

The second inequality is due to Jensen's inequality. The last inequality is due to the fact that $m \leq 2t+1$ and $m' \leq t+1$ (Lemma 3) and $x\log(n/x)+2x\log(\log(n/x)+1)$ increases over $1 \leq x \leq t+1 \leq \lfloor n/2\rfloor+1$. We proceed in evaluating our main inequality as

$$\begin{aligned}
&(t+1)\log\frac{n}{t+1}+2(t+1)\log\left(\log\frac{n}{t+1}+1\right) \\
&\quad +2t+2 \\
\leq\ &(t+1)\log\frac{n}{t}+2(t+1)\log\left(\log\frac{n}{t}+1\right)+2t+2 \\
=\ &\left(t\log\frac{n}{t}+t\right)+\left(2t\log\left(\log\frac{n}{t}+1\right)+t\right) \\
&\quad +2\log\left(\log\frac{n}{t}+1\right)+\log\frac{n}{t}+2 \\
\leq\ &nH_0(B)+2nH_0(B)+2\log(\log n+1)+\log n+2 \\
=\ &3nH_0(B)+2\log(\log n+1)+\log n+2
\end{aligned}$$

The last inequality is due to Lemma 3 and the fact that $\log x+1 \leq x$ for any $x \geq 2$. ■

Lemma 5. *If $H_0(B) = o(1)$, then $|B_{rle,\delta}| \leq nH_0(B) + o(nH_0(B)) + 2\log(\log n+1)+\log n+2$.*

Proof: We first show that $H_0(B) = o(1)$ implies $t = o(n)$. Suppose $t = \Omega(n)$ by contradiction: there must exist a positive constant $C \leq 1/2$, such that when n becomes large enough, we have $Cn \leq t \leq n/2$, which forces

$$H_0(B) = \frac{t}{n}\log\frac{n}{t}+\frac{n-t}{n}\log\frac{n}{n-t} \geq C+\frac{1}{2}\log\frac{1}{1-C} > C$$

which is a positive constant, a contradiction to $H_0(B) = o(1)$, so we have $t = o(n)$.

Now we bound $|B_{rle,\delta}|$. Following the result in the proof for Lemma 4, we have

$$\begin{aligned}
|B_{rle,\delta}| &\leq \left(t\log\frac{n}{t}+t\right)+\left(2t\log\left(\log\frac{n}{t}+1\right)+t\right) \\
&\quad +2\log\left(\log\frac{n}{t}+1\right)+\log\frac{n}{t}+2 \\
&\leq nH_0(B)+o(nH_0(B))+2\log(\log n+1) \\
&\quad +\log n+2
\end{aligned}$$

The last inequality is due to Lemma 3 and the fact $2t\log(\log(n/t)+1)+t = o(t\log(n/t)+t)$, because t is positive and $2\log(\log(n/t)+1) = o(\log(n/t))$ since n/t goes to infinite when n goes to infinite due to $t = o(n)$. ■

B. General string compression—achieving H_0

Let $T_{rle,\delta}$ denote the wavelet tree over the text T with the bit arrays run-length δ encoded. Let $|T_{rle,\delta}|$ denote the total number of bits in the run-length δ encoded bit arrays in $T_{rle,\delta}$. We can similarly define $T_{rle,\gamma}$ and $|T_{rle,\gamma}|$ using the run-length γ coding. By combining Lemma 10 in the Appendix and Fact 2, it is trivial to show that $|T_{rle,\gamma}| \leq 2nH_0(T)+(2\log n+2)(\sigma-1)$ for any text T. As for $T_{rle,\delta}$, we can similarly obtain $|T_{rle,\delta}| \leq 3nH_0(T)+(2\log(\log n+1)+\log n+2)(\sigma-1)$ for any text T, by combining Lemma 4 and Fact 2. In this section, we want to further show that if σ is constant and $H_0(T)$ is asymptotically less than $\log\sigma$,

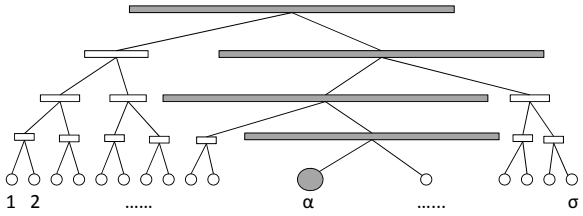


Figure 3. A wavelet tree of T whose σ is constant and $H_0(T) = o(\log \sigma)$

$|T_{rle,\delta}|$ achieves $nH_0(T)$ with leading constant 1, plus some lower order terms.

Lemma 6. *If σ is a constant and $H_0(T) = o(\log \sigma)$, then there must exist some α, $1 \le \alpha \le \sigma$, such that $\sum_{i \in \Sigma, i \neq \alpha} n_i = f(n)$, where $f(n)$ is some function of n and $f(n) = o(n)$.*

Proof: We first show by contradiction that there cannot be more than one character from Σ whose frequencies in T are asymptotically close to n. Suppose we have $k > 1$ characters whose frequencies in T are $\Omega(n)$. Without loss of the generality, we assume that these characters are $\{1, 2, \ldots, k\}$. That is, when n is large enough, for each $i \in \{1, 2, \ldots, k\}$, there exist some constants C_i and D_i, such that $0 < C_i < 1$, $0 < D_i < 1$, and $C_i\, n \le n_i \le D_i\, n$ (clearly, $0 < \sum_i C_i \le \sum_i D_i \le 1$). Then,

$$H_0(T) \ge \sum_{1 \le i \le k} \frac{n_i}{n} \log \frac{n}{n_i} \ge \sum_{1 \le i \le k} C_i \log \frac{1}{D_i}$$

which is a positive constant, a contradiction to $H_0(T) = o(\log \sigma)$.

On the other hand, by the pigeon principle there must exist a character $\alpha \in \Sigma$, whose frequency is $n_\alpha = \Omega(n)$: indeed $n_\alpha \ge n/\sigma$ and σ is a constant by hypothesis. As previously proved, α is the only character with this property: we must have $n_i = o(n)$ for all $i \in \Sigma \setminus \{\alpha\}$. Letting $f(n) = \sum_{i \in \Sigma, i \neq \alpha} n_i$, observe that $n_\alpha = n - f(n)$. It suffices to observe that $f(n) = o(n)$ since each $n_i = o(n)$ for $i \neq \alpha$ and there are $\sigma - 1 = O(1)$ of these terms in the summation for $f(n)$, thus proving the claimed bound. ■

Figure 3 visualizes what Lemma 6 claims. Character α dominates the string T. Let A^i, $1 \le i \le \sigma - 2$ ($i = \sigma - 2$ can happen in a very skewed wavelet tree), denote the wavelet tree raw bit arrays that do not involve character α (represented by those short white rectangles in Figure 3). Let B^i, $1 \le i \le \sigma - 1$ ($i = \sigma - 1$ can happen in a very skewed wavelet tree), denote the wavelet tree raw bit arrays that involve character α (represented by those long gray rectangles in Figure 3). Let $|A^i_{rle,\delta}|$ and $|B^i_{rle,\delta}|$ denote the run-length δ coding size of A^i and B^i respectively. We now prove our main result, for which the following lemma is used.

Lemma 7. *For any bit string B of size n, $|B_{rle,\delta}| \le 3n$*

Proof: For any bit-run of size $\ell_i > 0$, we have the fact $\delta(\ell_i) \le 2\ell_i$. That is, the δ code of a bit-run uses no more than double of the number of bits in the run. Since $B_{rle,\delta}$ also has a leading bit for decoding, we get $|B_{rle,\delta}| \le 2n + 1 \le 3n$. ■

Theorem 8. *If σ is a constant and $H_0(T) = o(\log \sigma)$, then $|T_{rle,\delta}| \le nH_0(T) + o(nH_0(T)) + (2\log(\log n + 1) + \log n + 2)(\sigma - 1)$*

Proof: We first give a lower bound of $nH_0(T)$. By Lemma 6, we have $n_\alpha = n - f(n)$ with $f(n) = o(n)$, thus

$$nH_0(T) \ge f(n) \log \frac{n}{f(n)} + n_\alpha \log \frac{n}{n_\alpha} = \omega(f(n)) \qquad (1)$$

The right side of inequality (1) is the 0-order empirical entropy size of a text, which is obtained by converting all the characters in T except α into an identical character other than α. The entropy size of this new binary text must be no more than the entropy size of T.

For those wavelet tree bit arrays that do not involve α, we have $\sum_i |A^i_{rle,\delta}| < 3 \sum_i |A^i| < 3f(n)(\sigma - 2)$, because each $|A^i_{rle,\delta}| \le 3|A^i|$ (Lemma 7) and the sum of all $|A^i|$ on each wavelet tree level is bounded by $f(n)$ and we have no more than $\sigma - 2$ levels where A_i's can occur. Because of inequality (1) and σ being a constant, we get $\sum_i |A^i_{rle,\delta}| < 3f(n)(\sigma/2 + 1) = o(nH_0(T))$.

For those bit arrays that involve α, because α dominates the string at each node of B^i, each B^i has the property that $t^i = o(n^i)$, where t^i is the number of the least frequent bit in B^i and n^i is the size of B^i. Therefore, $|B^i_{rle,\delta}| \le n^i H_0(B^i) + o(n^i H_0(B^i)) + 2\log(\log n^i + 1) + \log n^i + 2$ (Lemma 5). By adding up all the $|B^i_{rle,\delta}|$, we get $\sum_i |B^i_{rle,\delta}| \le nH_0(T) + o(nH_0(T)) + (2\log(\log n + 1) + \log n + 2)(\sigma - 1)$, because $\sum_i n^i H_0(B^i) \le nH_0(T)$ (Fact 2) and we have no more than $(\sigma - 1)$ number of B^i bit arrays.

Finally, $|T_{rle,\delta}| = \sum_i |A^i_{rle,\delta}| + \sum_i |B^i_{rle,\delta}| \le nH_0(T) + o(nH_0(T)) + (2\log(\log n + 1) + \log n + 2)(\sigma - 1)$. ■

Corollary 9. *If σ is a constant, $H_0(T) = o(\log \sigma)$, and $nH_0(T) = \omega(\log n)$, then $|T_{rle,\delta}| \le (1 + o(1))nH_0(T) + O(\sigma)$.*

Proof: Immediately from the result in Theorem 8 and the hypothesis $nH_0(T) = \omega(\log n)$. ■

Comment: Theorem 3.1 in [19] claims that if A is a non-singular compressor [3], then $|A(T)| \le \lambda nH_0^*(T) + g_\sigma$ holds only for $\lambda \ge 2$. Here: (1) $|A(T)|$ is the number of bits in the output of compressing T using A; (2) $H_0^*(T)$ is the modified 0-order empirical entropy [20] of T, which essentially defines $nH_0^*(T) = \log n + 1$ if T is a text of n identical characters; otherwise $nH_0^*(T) = nH_0(T)$; and (3) g_σ is a function only of the alphabet size σ. However,

[3] A is a non-singular compressor, if $A(T_1) \neq A(T_2)$ given $T_1 \neq T_2$

our Theorem 8 and Corollary 9 indicate that we can do better in terms of the above lower bound in [19], except few cases. That is, (1) $\lambda < 2$ can be achieved, when $nH_0^*(T) = \omega(\log n)$ (or equivalently, $nH_0(T) = \omega(\log n)$); (2) for the few special cases where $nH_0^*(T) = \Theta(\log n)$ (or equivalently, $nH_0(T) = \Theta(\log n)$ or $nH_0(T) = o(\log n)$), λ can be larger than 2. It is nevertheless worth noting that when employed to encode an optimal partition of the BWT, it may happen several times that $nH_0 = O(\log n)$, so the motivation for Theorem 3.1 in [19] is realistic in that context. In our context, we however look at the general case in which a 0th-order compressor is needed.

III. A Manifold Software Tool for Experimenting on Wavelet Trees

We wanted to experiment with the properties discussed in Section II. In that context, we realized that a more general tool could be useful also for other experiments. As a result, we implemented a generic template-based software package library that provides several incarnations of the wavelet trees using C++.[4] In this way, we hope that we can contribute to the current status of the wavelet tree implementations. Indeed, some of them are available in the context of compressed full-text indexes in the Pizza&Chili Corpus and can be extracted from that software distributions. Other implementations using normal and Huffman wavelet trees with RRR [8] coding have been studied by Navarro et al. [21], [22], from which we reuse the practical implementation of the RRR structure as a component in the RRR compressed wavelet trees. Along with the above implementations, we added a lot more options than those previously known. As far as we know, this is the first package to provide a large set of options living under the same roof.

We employed our tool to present the experiments on low-entropy texts required in Section II, as well as the experiments on normal texts to evaluate a wide range of options shown in Table I. All the options in Table I are supported while only those with bold selection marks will be discussed in the experimental study: the others are apparently less efficient or unnecessary, but there could be other applications using them in a different context that might favor them. So we leave the final choice to the end user, depending on the kind of application she has in mind.

A. Software tool

We give a brief description for the available options shown in Table I. We first give a choice for three kinds of shapes. `Normal:` a balanced shape of the wavelet tree (see an example in Figure 1), where the local alphabet at each node is divided (nearly) into two halves, while the alphabetic order among leaf nodes is preserved, so that the height is the logarithm of the alphabet size. `AWWT:` alphabetic weight-balanced wavelet tree (aka Hu-Tucker-shaped wavelet tree [23]), where the frequencies of the alphabet symbols are considered when dividing each node into its children having similar weights. `Huffman:` the standard Huffman-shaped wavelet tree [22].

We then give 11 options for choosing the coding scheme for the bit array at each wavelet tree node, where the attached directories for searching capability can be omitted in several cases. `None:` raw bit arrays are stored, augmented by structures of $o(n)$ extra bits to support the $O(1)$ time member, rank, and select queries. As suggested in [22], we use the technique based on one-level superblocks and lookup table-based popcount operations for the $o(n)$ searching structures. `RRR:` the practical implementation of the RRR structure [21] is used to compress each bit array. Huffman shaped-wavelet tree combined with RRR coding was claimed the best in [21], so our experimental study did include the comparison with the best prior work. `RLE+`γ`:` Each bit array is compressed using the run-length γ encoding [15], [6]. We also use one-level superblock-based technique over the runs for the searching ability. `RLE+`δ`:` Each bit array is compressed using the run-length δ encoding: note that this is an implementation of what was discussed in Section II. We also use one-level superblock based technique over the runs for the searching ability. `SC:` Each bit array is compressed using the small-integer t-subset encoding [24]. `AC:` Each bit array is compressed using pure 0th-order arithmetic coding [25]. `LP/`$*$`:` Only the bit arrays at nodes whose local alphabet size is no more than three are compressed using any of the aforementioned coding schemes, while other bit arrays are not compressed. This option is proposed because empirical observations show that, in the inherently compressed wavelet trees such as the AWWT, those nodes close to the leaves are likely to have sparse bit arrays, which can be highly compressed.

B. Experimental setup

We used the following real-world biological and textual data and their *low-entropy* variations to test the efficiency of the several wavelet trees. The main goal of using the low-entropy data is to validate our analysis in Section II that run-length δ encoding is more space efficient than run-length γ encoding when the entropy of the data is small.

1) `Genome`: The whole human genome sequences of around 2.8G bases from NCBI[5]. We removed all the masked '`N`' characters, so that the sequence only contain characters from {`ACGT`}.
2) `Protein`: Protein data of around 1.1G characters from the Pizza&Chili Corpus.

[4] Download: `http://www.ittc.ku.edu/~bojianxu/publications`

[5] `ftp://ftp.ncbi.nlm.nih.gov/genomes/H_sapiens/Assembled_chromosomes`

	None	RRR	RLE+γ	RLE+δ	SC	AC	LP/RRR	LP/(RLE+γ)	LP/(RLE+δ)	LP/SC	LP/AC
Normal	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
AWWT	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Huffman	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Table I
THE 33 SUPPORTED WAVELET TREE OPTIONS ON THE TREE SHAPES AND CODING SCHEMES.

3) `English`: English text of around 2G characters from the Wikipedia dump on 2010–07–30 [6].
4) `Genome-`X, `Protein-`X, and `English-`X: The symbol X implies that the average length of the maximal runs of identical characters in the sequence is $X-1$. The low-entropy data is obtained by randomly replacing each character in the original real-world data with an identical character from the alphabet. For example, `Genome-10` is obtained by replacing each character in `Genome` with an identical character from the alphabet $\{$`ACGT`$\}$ with probability $9/10 = 0.9$; `Genome-20` is similarly obtained but with probability $19/20 = 0.95$. Therefore, a larger value for X leads to a sequence of a lower entropy. In our experiments, we set $X \in \{10, 20, 40, 80, 160\}$.

For our computing platform, we employed `g++ 4.4.1` with `-O9` option to build the executables of all the source code in our experiments. The experiments were conducted on a Dell Vostro 430 with a 2.8GHz four-core Intel@CoreTM i7-860 chip with 8MB L3 Cache, but no parallelism was used and only one core is used. The machine runs 64-bit Ubuntu 9.10 operating system and has 8GB internal memory.

C. Experimental study

We focused on the measurements of the wavelet tree size (the size of the bit arrays and their search structures), construction time, and query (member, rank, and select) performance. We did not specifically run experiments for the 2-d 4-sided range queries as they are reduced to the member, rank, and select queries.

Summary. The overall statement from the experimental study can be summarized as the following: (1) run-length encoding-based wavelet trees are the best in space efficiency for BWT and lower-entropy data, but have the worst query performance. Run-length γ coding is more space saving in practice, whereas run-length δ coding becomes better when the entropy of the data is very small. (2) The compression ratio of a plain AWWT is comparable to the Huffman-shaped wavelet tree. It is also the case when they are both combined with RRR compression. Huffman-shaped wavelet tree is more space efficient than AWWT for the data of very low entropy. (3) Huffman wavelet tree and AWWT combined with RRR coding are the good trade-off between space efficiency and query performance. (4) The search performance of plain AWWT is one of the best in all types of wavelet trees and a bit slower when combined with RRR compression for a better compression ratio. AWWT is also the fastest in the construction time and the only type of wavelet trees that are inherently compressed and also support 2-d 4-sided range queries. We believe that using the wavelet tree in other contexts other than those discussed here, could give probably different results since they depend on the kind of distribution of the input data. This is the reason to leave so many options in our package library.

We give more details on the experimental results. We ran two sets of experiments, one for normal texts and the other for the BWT of the texts. For ease of display in the tables, we use the following numbers to represent the subset of options that we are going to discuss, where option 3 represents what we called $|T_{rle,\delta}|$ in Section II, and option 4 represents what we called $|T_{rle,\gamma}|$.

1:	Huffman + None	2:	Normal + None
3:	Normal + (RLE+δ)	4:	Normal + (RLE+γ)
5:	Normal + RRR	6:	AWWT + LP/(RLE+γ)
7:	AWWT + LP/RRR	8:	AWWT + None
9:	AWWT + (RLE + γ)	10:	AWWT + RRR
11:	Huffman + RRR	12:	Huffman + (RLE+γ)
13:	Huffman + LP/RRR	14:	Huffman + LP/(RLE+γ)

Wavelet tree size. Table II shows the size of the wavelet trees without searching capability. Exceptions are the RRR-compressed wavelet trees (options 5, 10, 11), still being searchable.

We first examine the compression size of the normal texts of the real-world data. The run-length γ coding (option 4) is more space efficient than the run-length δ coding (option 3) in practice, which is consistent with our description in Section II. Huffman (option 1) and AWWT+LP/RRR (option 7) achieve the best compression ratio, while AWWT+LP/RLE+γ (option 6) is also close to the best. This result is consistent with our empirical observations that the tree nodes close to the leaf nodes in AWWT are likely to have sparse bit arrays, which can be effectively compressed using RRR or run-length encoding to reach a good compression ratio. Another observation is that an AWWT itself is already good at 0th-order entropy

[6] `http://download.wikimedia.org/enwiki/20100730/enwiki-20100730-pages-articles.xml.bz2`

(a) normal texts

		1	2	3	4	5	6	7	8	9	10	11	12	13	14
Real-world Data	Genome	**1.01**	1.01	1.26	1.09	1.20	1.04	**1.01**	1.01	1.09	1.20	1.20	1.10	1.10	1.05
	Protein	**1.01**	1.16	1.29	1.12	1.23	1.03	**1.01**	1.08	1.11	1.21	1.19	1.10	1.04	1.02
	English	**1.01**	1.50	1.22	1.08	1.31	1.01	**1.01**	1.04	1.05	1.16	1.16	1.07	1.01	1.00
Low-entropy Data	Genome-10	2.33	4.19	**1.41**	**1.40**	2.27	2.35	2.37	2.33	**1.31**	1.62	1.62	**1.31**	2.37	2.35
	Genome-20	3.84	7.27	**1.42**	**1.47**	3.41	3.86	3.88	3.84	**1.38**	2.15	2.15	**1.38**	3.88	3.86
	Genome-40	6.62	12.88	**1.43**	**1.53**	5.52	6.64	6.65	6.62	**1.45**	3.17	3.17	**1.45**	6.65	6.64
	Genome-80	11.72	23.12	**1.44**	**1.57**	9.42	11.74	11.75	11.72	**1.50**	5.09	5.09	**1.50**	11.75	11.74
	Genome-160	21.13	41.96	**1.44**	**1.61**	16.59	21.14	21.15	21.13	**1.55**	8.67	8.67	**1.55**	21.15	21.14
	Protein-10	1.63	4.72	**1.39**	**1.37**	2.42	2.62	2.63	2.64	**1.30**	1.70	1.38	**1.23**	1.65	1.64
	Protein-20	2.49	8.37	**1.40**	**1.43**	3.71	4.41	4.43	4.43	**1.37**	2.33	1.67	**1.29**	2.51	2.50
	Protein-40	4.14	15.09	**1.41**	**1.48**	6.13	7.76	7.77	7.77	**1.42**	3.53	2.25	**1.35**	4.15	4.15
	Protein-80	7.22	27.56	**1.42**	**1.52**	10.60	13.97	13.98	13.99	**1.47**	5.79	3.39	**1.40**	7.24	7.23
	Protein-160	12.99	50.74	**1.42**	**1.56**	18.88	25.55	25.56	25.56	**1.51**	10.01	5.54	**1.45**	13.01	13.00
	English-10	1.54	8.06	**1.36**	**1.29**	3.67	2.40	2.41	2.41	**1.25**	1.64	1.36	**1.21**	1.55	1.54
	English-20	2.31	14.62	**1.37**	**1.36**	6.02	4.01	4.01	4.02	**1.32**	2.21	1.61	**1.27**	2.32	2.31
	English-40	3.79	26.78	**1.38**	**1.41**	10.38	7.02	7.02	7.02	**1.37**	3.30	2.14	**1.32**	3.79	3.79
	English-80	6.58	49.41	**1.40**	**1.46**	18.63	12.65	12.65	12.65	**1.42**	5.37	3.17	**1.38**	6.59	6.59
	English-160	11.85	91.74	**1.41**	**1.50**	33.81	23.21	23.21	23.22	**1.47**	9.26	5.15	**1.42**	11.85	11.85

(b) BWT of texts

		1	2	3	4	5	6	7	8	9	10	11	12	13	14
Real-world Data	Genome	1.01	1.01	**1.08**	**0.94**	1.07	0.95	1.02	1.01	**0.94**	1.07	1.07	**0.95**	1.03	0.96
	Protein	1.01	1.16	**0.71**	**0.65**	0.93	0.93	1.00	1.08	**0.64**	0.91	0.88	**0.64**	0.97	0.91
	English	1.01	1.50	**0.32**	**0.29**	0.75	0.92	0.97	1.04	**0.29**	0.58	0.57	**0.29**	0.97	0.95
Low-entropy Data	Genome-10	2.33	4.19	**1.42**	**1.41**	2.27	2.35	2.37	2.33	**1.32**	1.63	1.63	**1.32**	2.37	2.35
	Genome-20	3.84	7.27	**1.43**	**1.48**	3.41	3.86	3.88	3.84	**1.39**	2.15	2.15	**1.39**	3.88	3.86
	Genome-40	6.62	12.88	**1.44**	**1.53**	5.53	6.64	6.65	6.62	**1.45**	3.17	3.17	**1.45**	6.65	6.64
	Genome-80	11.72	23.12	**1.45**	**1.58**	9.42	11.74	11.75	11.72	**1.50**	5.10	5.10	**1.50**	11.75	11.74
	Genome-160	21.13	41.96	**1.45**	**1.62**	16.59	21.14	21.15	21.13	**1.55**	8.67	8.67	**1.55**	21.15	21.14
	Protein-10	1.63	4.72	**1.41**	**1.38**	2.43	2.62	2.63	2.64	**1.31**	1.71	1.39	**1.24**	1.65	1.64
	Protein-20	2.49	8.37	**1.41**	**1.44**	3.72	4.42	4.43	4.43	**1.38**	2.33	1.68	**1.30**	2.51	2.51
	Protein-40	4.14	15.09	**1.42**	**1.49**	6.13	7.76	7.77	7.77	**1.43**	3.53	2.26	**1.36**	4.16	4.15
	Protein-80	7.22	27.56	**1.43**	**1.53**	10.61	13.98	13.99	13.99	**1.48**	5.79	3.39	**1.41**	7.24	7.23
	Protein-160	12.99	50.74	**1.43**	**1.56**	18.88	25.55	25.56	25.56	**1.52**	10.01	5.54	**1.45**	13.01	13.00
	English-10	1.54	8.06	**1.33**	**1.27**	3.66	2.40	2.41	2.41	**1.23**	1.62	1.33	**1.18**	1.54	1.54
	English-20	2.31	14.62	**1.37**	**1.36**	6.02	4.01	4.01	4.02	**1.32**	2.20	1.61	**1.26**	2.32	2.31
	English-40	3.79	26.78	**1.39**	**1.42**	10.38	7.02	7.02	7.02	**1.38**	3.30	2.14	**1.33**	3.79	3.79
	English-80	6.58	49.41	**1.41**	**1.47**	18.63	12.65	12.65	12.65	**1.43**	5.37	3.17	**1.38**	6.59	6.59
	English-160	11.85	91.74	**1.42**	**1.51**	33.81	23.21	23.21	23.22	**1.47**	9.27	5.15	**1.43**	11.85	11.85

Table II
THE SIZE OF WAVELET TREES WITHOUT SEARCHING CAPABILITY EXCEPT THOSE RRR-COMPRESSED (OPTIONS 5, 10, 11) STILL BEING SEARCHABLE. TREE SIZE IS EXPRESSED AS THE RATIO OF THE WAVELET TREE SIZE DIVIDED BY THE 0TH-ORDER EMPIRICAL ENTROPY SIZE OF THE TEXT.

compression (option 8).

As for the normal texts of the low-entropy data, the compression performance of each option decreases as the entropy of the data becomes smaller. This is reasonable because the overhead caused by various auxiliary data structures in the wavelet tree become relatively more significant. The run-length γ coding again is more space efficient than the run-length δ coding until the entropy of the data becomes very small (option 3 and 4 for the low-entropy data in Table I(a)), which is consistent with our analysis in Section II. Overall, only the run-length encoding-based wavelet trees (options 3, 4, 9, 12) can achieve closely to the entropy of the data, because the run-length encoding can effectively code the bit-runs in the bit arrays regardless of the length of the runs. However, the RRR coding (options 5, 10, 11) of a given-length bit array has fixed sizes for the superblock and block structures, introducing relatively higher overheads when the bit-runs become much longer than the block size, which is the case in the data of very low entropy; the Huffman wavelet tree and AWWT (options 1, 8), which have to use at least one bit for the most frequent character, also make the coding very inefficient when the frequent character dominates the text. Huffman and AWWT combined with LP/* coding (options 6, 7, 13, 14) are not efficient because the very long and sparse bit arrays that involve the most frequent character are far away from the leaf nodes, making them have no chance to be compressed in the LP/* coding, which leads to an overall less efficient compression.

For the BWT of the texts, all the above observations for the normal texts are valid except that the run-length encoding-based wavelet trees (options 3, 4, 9, 12) turn out to be the best for both the real-world data and the low-entropy data, because the run-length-based encoding can capture and effectively code the partitions of context and automatically achieves high-order compression. This automatic high-order compression becomes even more relevant in the real-world English text, because English text has rich semantic correlations between contexts.

For the wavelet trees that have search capabilities (Table III), similar considerations hold except that more options

(a) normal texts

		1	2	3	4	5	6	7	8	9	10	11	12	13	14
Real-world Data	Genome	**1.08**	1.08	1.35	1.18	1.20	**1.12**	**1.12**	1.08	1.18	1.20	1.20	1.19	**1.13**	**1.12**
	Protein	**1.07**	1.23	1.40	1.23	1.23	**1.10**	**1.11**	1.15	1.22	1.21	1.19	1.22	**1.09**	**1.10**
	English	**1.07**	1.60	1.32	1.17	1.31	**1.08**	**1.09**	1.11	1.15	1.16	1.16	1.17	**1.07**	**1.07**
Low-entropy Data	Genome-10	2.47	4.45	**1.48**	**1.47**	2.27	2.51	2.50	2.47	**1.39**	1.62	1.62	**1.39**	2.50	2.51
	Genome-20	4.08	7.73	**1.48**	**1.54**	3.41	4.11	4.10	4.08	**1.46**	2.15	2.15	**1.46**	4.10	4.11
	Genome-40	7.03	13.69	**1.50**	**1.59**	5.52	7.06	7.05	7.03	**1.52**	3.17	3.17	**1.52**	7.05	7.06
	Genome-80	12.46	24.57	**1.51**	**1.64**	9.42	12.48	12.47	12.46	**1.57**	5.09	5.09	**1.57**	12.47	12.48
	Genome-160	22.45	44.58	**1.50**	**1.67**	16.59	22.47	22.46	22.45	**1.61**	8.67	8.67	**1.61**	22.46	22.47
	Protein-10	1.74	5.02	**1.48**	**1.46**	2.42	2.79	2.79	2.81	**1.40**	1.70	1.38	**1.34**	1.74	1.75
	Protein-20	2.66	8.90	**1.49**	**1.51**	3.71	4.70	4.70	4.72	**1.46**	2.33	1.67	**1.40**	2.67	2.67
	Protein-40	4.41	16.05	**1.49**	**1.56**	6.13	8.26	8.25	8.27	**1.51**	3.53	2.25	**1.45**	4.42	4.42
	Protein-80	7.69	29.30	**1.50**	**1.60**	10.60	14.86	14.86	14.87	**1.55**	5.79	3.39	**1.49**	7.69	7.70
	Protein-160	13.82	53.92	**1.49**	**1.63**	18.88	27.16	27.16	27.17	**1.59**	10.01	5.54	**1.53**	13.82	13.83
	English-10	1.65	8.58	**1.45**	**1.39**	3.67	2.56	2.56	2.57	**1.35**	1.64	1.36	**1.31**	1.65	1.65
	English-20	2.47	15.55	**1.46**	**1.45**	6.02	4.27	4.27	4.28	**1.41**	2.21	1.61	**1.37**	2.47	2.47
	English-40	4.04	28.47	**1.47**	**1.50**	10.38	7.47	7.47	7.48	**1.46**	3.30	2.14	**1.42**	4.04	4.04
	English-80	7.01	52.51	**1.48**	**1.54**	18.63	13.45	13.45	13.46	**1.51**	5.37	3.17	**1.47**	7.01	7.01
	English-160	12.60	97.49	**1.49**	**1.58**	33.81	24.67	24.67	24.68	**1.55**	9.26	5.15	**1.51**	12.60	12.61

(b) BWT of texts

		1	2	3	4	5	6	7	8	9	10	11	12	13	14
Real-world Data	Genome	1.08	1.08	**1.16**	**1.02**	1.07	1.02	1.05	1.08	**1.02**	1.07	1.07	**1.02**	1.06	1.03
	Protein	1.07	1.23	**0.76**	**0.69**	0.93	0.99	1.05	1.15	**0.69**	0.91	0.88	**0.69**	1.02	0.97
	English	1.07	1.60	**0.34**	**0.31**	0.75	0.98	1.03	1.11	**0.31**	0.58	0.57	**0.31**	1.03	1.01
Low-entropy Data	Genome-10	2.47	4.45	**1.50**	**1.48**	2.27	2.51	2.50	2.47	**1.40**	1.63	1.63	**1.40**	2.50	2.51
	Genome-20	4.08	7.73	**1.49**	**1.55**	3.41	4.12	4.11	4.08	**1.47**	2.15	2.15	**1.47**	4.11	4.12
	Genome-40	7.03	13.69	**1.50**	**1.60**	5.53	7.07	7.05	7.03	**1.53**	3.17	3.17	**1.53**	7.05	7.07
	Genome-80	12.46	24.57	**1.51**	**1.64**	9.42	12.49	12.47	12.46	**1.58**	5.10	5.10	**1.58**	12.47	12.49
	Genome-160	22.45	44.58	**1.51**	**1.68**	16.59	22.48	22.46	22.45	**1.62**	8.67	8.67	**1.62**	22.46	22.48
	Protein-10	1.74	5.02	**1.50**	**1.47**	2.43	2.80	2.80	2.81	**1.42**	1.71	1.39	**1.35**	1.75	1.76
	Protein-20	2.66	8.90	**1.50**	**1.52**	3.72	4.71	4.71	4.72	**1.47**	2.33	1.68	**1.40**	2.67	2.68
	Protein-40	4.41	16.05	**1.51**	**1.57**	6.13	8.26	8.26	8.27	**1.52**	3.53	2.26	**1.46**	4.42	4.43
	Protein-80	7.69	29.30	**1.51**	**1.61**	10.61	14.86	14.86	14.87	**1.56**	5.79	3.39	**1.50**	7.70	7.70
	Protein-160	13.82	53.92	**1.50**	**1.64**	18.88	27.16	27.16	27.17	**1.60**	10.01	5.54	**1.54**	13.82	13.83
	English-10	1.65	8.58	**1.42**	**1.36**	3.66	2.56	2.56	2.57	**1.33**	1.62	1.33	**1.28**	1.65	1.65
	English-20	2.47	15.55	**1.46**	**1.45**	6.02	4.27	4.27	4.28	**1.41**	2.20	1.61	**1.36**	2.47	2.47
	English-40	4.04	28.47	**1.48**	**1.51**	10.38	7.47	7.47	7.48	**1.47**	3.30	2.14	**1.43**	4.04	4.04
	English-80	7.01	52.51	**1.49**	**1.55**	18.63	13.45	13.45	13.46	**1.52**	5.37	3.17	**1.48**	7.01	7.02
	English-160	12.60	97.49	**1.50**	**1.59**	33.81	24.67	24.67	24.68	**1.56**	9.27	5.15	**1.52**	12.60	12.61

Table III
THE SIZE OF WAVELET TREES WITH SEARCHING CAPABILITY. TREE SIZE IS EXPRESSED AS THE RATIO OF THE WAVELET TREE SIZE DIVIDED BY THE 0TH-ORDER EMPIRICAL ENTROPY SIZE OF THE TEXT.

(1, 6, 7, 13, 14) can achieve comparable compression performance for the normal texts of the real-world data, while clearly the overall size of the wavelet tree is a bit larger than those without searching capabilities.

Wavelet tree construction time. The distinction between low-entropy and normal texts does not significantly affect the wavelet tree construction time, so we move the results regarding the low-entropy data into Table VIII in the Appendix. Table IV shows that pure AWWT (option 8) is the fastest for both normal texts and BWTs. Normal wavelet trees (option 2) and Huffman-shaped wavelet tree (option 1) are also fast. AWWT+LP/RRR (option 7) is comparable. The run-length encoding-based methods (options 3, 4, 9, 12, 14) take longer time than any other method as they need γ or δ encoding for each run. AWWT+LP/(RLE+γ) (option 6) is relatively faster than other run-length-based options because it only encodes the wavelet tree nodes that are close to leaf nodes and the construction of the pure AWWT is already the fastest. It does not come to surprise that run-length γ coding (option 4) is faster than run-length δ coding (option 3) since the latter needs more computation in the encoding and decoding of a same integer. On the average, the RRR-based wavelet trees (options 5, 10, 11) reach the median performance in construction. Overall, all types of wavelet trees take less construction time for BWTs than that for normal texts, because the longer bit-runs at the nodes of the wavelet trees for BWTs makes the encoding of the whole bit arrays faster.

Wavelet tree query time. Once again, the distinction between low-entropy and normal texts does not seem to play a significant role in the performance of member, rank, and select queries, so we move the results regarding the low-entropy data into Table IX–XI in the Appendix. Tables V–VII show that the Huffman wavelet tree, the normal wavelet tree, and the AWWT (options 1, 2, 8) reach the best performance. All RRR-compressed wavelet trees (options 5, 7, 10, 11, 13) are comparable in query performance with the best options, because they are all based on block structures and table lookups, while RRR-compressed wavelet trees are a bit slower to pay for the gain in space efficiency. All

(a) normal texts

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Genome	**67.91**	**59.90**	191.34	182.13	111.36	120.45	**84.83**	**59.88**	178.41	110.51	118.84	186.81	94.37	127.67
Protein	**60.07**	**55.02**	172.23	164.47	101.37	72.96	**61.54**	**51.87**	157.66	98.25	104.48	164.33	71.30	86.05
English	**147.27**	**152.04**	403.19	389.41	265.55	137.71	**123.47**	**112.48**	329.06	205.20	241.50	367.92	153.14	161.73

(b) BWT of texts

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Genome	**64.21**	**57.41**	172.72	166.00	105.08	109.14	**80.37**	**57.48**	164.30	104.65	111.71	170.05	88.84	116.68
Protein	**46.65**	**45.36**	120.16	118.79	83.60	57.11	**51.03**	**42.86**	113.16	79.82	81.66	115.70	55.19	63.59
English	**95.39**	**113.28**	209.30	211.15	200.09	89.12	**88.61**	**80.59**	162.16	143.24	155.79	174.89	99.13	99.28

Table IV
THE CONSTRUCTION TIME (SECOND) OF WAVELET TREES

(a) normal texts

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Genome	**0.46**	**0.46**	12.01	10.87	0.80	5.45	0.63	**0.46**	10.90	0.80	0.80	10.58	0.62	5.60
Protein	**0.94**	**1.08**	24.64	22.51	1.88	5.87	1.18	**1.01**	20.68	1.77	1.66	18.76	1.10	4.81
English	**1.26**	**1.94**	45.04	42.32	3.19	4.54	1.43	**1.32**	26.57	2.29	2.23	25.21	1.32	2.64

(b) BWT of texts

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Genome	**0.46**	**0.46**	12.39	11.42	0.80	5.86	0.61	**0.46**	11.47	0.79	0.78	11.42	0.61	5.79
Protein	**0.94**	**1.08**	31.63	30.79	1.75	7.86	1.15	**1.01**	28.43	1.65	1.54	25.80	1.07	6.34
English	**1.26**	**1.94**	52.97	54.87	2.71	5.88	1.39	**1.31**	35.64	1.88	1.82	34.14	1.29	3.27

Table V
THE TIME COST (SECOND) FOR 1,000,000 MEMBERSHIP QUERIES.

(a) normal texts

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Genome	**0.82**	**0.83**	18.66	16.93	1.37	8.51	1.09	**0.83**	16.94	1.36	1.37	16.51	1.08	8.78
Protein	**1.69**	**1.94**	38.30	35.04	3.19	9.21	2.07	**1.81**	32.20	3.02	2.86	29.34	1.97	7.71
English	**2.30**	**3.51**	69.71	65.60	5.43	7.34	2.55	**2.37**	41.41	3.95	3.88	39.49	2.40	4.47

(b) BWT of texts

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Genome	**0.82**	**0.83**	19.18	17.68	1.33	9.08	1.06	**0.83**	17.72	1.34	1.34	17.67	1.07	9.04
Protein	**1.68**	**1.94**	48.45	47.25	2.97	12.17	2.03	**1.81**	43.72	2.81	2.64	39.75	1.93	9.96
English	**2.26**	**3.43**	79.98	82.71	4.53	9.09	2.43	**2.31**	53.99	3.17	3.13	51.91	2.32	5.26

Table VI
THE TIME COST (SECOND) FOR 1,000,000 RANK QUERIES.

run-length encoding based wavelet trees (options 3, 4, 6, 9, 12, 14) are much slower due to the γ or δ decoding processes that are required for each query. The search performance of all types of wavelet trees reduces when the alphabet size increases, because the wavelet tree becomes taller and more queries on bit arrays at tree nodes are involved. We also conducted experiments on NORMAL+SC and NORMAL+AC and observed that they are unacceptably slow while SC can be a bit faster than AC for wavelet trees with very skewed bit arrays. Both SC and AC based wavelet tree can very precisely reach the 0th-order entropy compressed size of the text.

REFERENCES

[1] G. Navarro and V. Mäkinen, "Compressed full-text indexes," *ACM Computing Surveys*, vol. 39, no. 1, p. article 2, 2007.

[2] R. Grossi, A. Gupta, and J. S. Vitter, "High-order entropy-compressed text indexes," in *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2003, pp. 841–850.

[3] R. Grossi and J. S. Vitter, "Compressed suffix arrays and suffix trees with applications to text indexing and string matching," *SIAM Journal on Computing*, vol. 35, no. 32, pp. 378–407, 2005, (also in STOC2000).

[4] M. Burrows and D. Wheeler, "A block sorting data compression algorithm," Digital Systems Research Center, Tech. Rep., 1994.

[5] P. Ferragina and G. Manzini, "Indexing compressed text," *Journal of the ACM*, vol. 52, no. 4, pp. 552–581, 2005.

[6] P. Ferragina, R. Giancarlo, and G. Manzini, "The myriad virtues of wavelet trees," *Information and Computation*, vol. 207, no. 8, pp. 849 – 866, 2009, (also in ICALP2006).

[7] A. Brodnik and J. I. Munro, "Membership in constant time and almost-minimum space," *SIAM Journal on Computing*, vol. 28, no. 5, pp. 1627–1640, Oct. 1999.

(a) normal texts

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Genome	**1.43**	**1.42**	29.47	27.09	2.19	13.78	1.81	**1.42**	27.17	2.19	2.18	26.38	1.78	14.19
Protein	**3.16**	**3.74**	56.78	52.28	5.40	14.24	3.80	**3.45**	47.72	5.02	4.67	43.23	3.55	12.14
English	**3.19**	**5.47**	87.17	82.71	7.82	8.86	3.52	**3.32**	47.86	4.99	4.83	45.00	3.32	5.52

(b) BWT of texts

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Genome	**1.44**	**1.43**	30.13	28.09	2.14	14.58	1.77	**1.43**	28.20	2.15	2.15	28.17	1.79	14.50
Protein	**3.15**	**3.73**	73.68	72.35	5.09	18.54	3.73	**3.45**	66.32	4.73	4.37	60.39	3.47	15.97
English	**3.18**	**5.55**	102.01	106.21	6.95	11.03	3.42	**3.30**	63.63	4.27	4.12	60.27	3.26	6.56

Table VII
THE TIME COST (SECOND) FOR 1,000,000 SELECT QUERIES.

[8] R. Raman, V. Raman, and S. S. Rao, "Succinct indexable dictionaries with applications to encoding k-ary trees, prefix sums and multisets," *ACM Transactions on Algorithms*, vol. 3, no. 4, p. 43, 2007, (also in SODA2002).

[9] P. Ferragina, G. Manzini, V. Mäkinen, and G. Navarro, "Compressed representations of sequences and full-text indexes," *ACM Transactions on Algorithms*, vol. 3, no. 2, 2007, (Also in SPIRE2004).

[10] V. Mäkinen and G. Navarro, "Implicit compression boosting with applications to self-indexing," in *the Proceedings of String Processing and Information Retrieval Symposium (SPIRE)*, 2007, pp. 229–241.

[11] W. K. Hon, R. Shah, and J. S. Vitter, "Compression, indexing, and retrieval for massive string data," in *the Proceedings of Annual Symposium on Combinatorial Pattern Matching (CPM)*, 2010, pp. 260–274.

[12] W.-K. Hon, R. Shah, and J. S. Vitter, "Ordered pattern matching: Towards full-text retrieval," in *Purdue University Tech Rept*, 2006.

[13] J. S. Vitter, *Algorithms and Data Structures for External Memory*, ser. Foundations and Trends in Theoretical Computer Science. Hanover, MA: now Publishers, 2008.

[14] P. Ferragina, G. Manzini, V. Mäkinen, and G. Navarro, "An alphabet-friendly FM-index," in *the Proceedings of String Processing and Information Retrieval Symposium (SPIRE)*, 2004, pp. 150–160.

[15] L. Foschini, R. Grossi, A. Gupta, and J. S. Vitter, "When indexing equals compression: Experiments with compressing suffix arrays and applications," *ACM Transactions on Algorithms (TALG)*, vol. 2, no. 4, pp. 611–639, 2006, (also in DCC2004 and SODA2004).

[16] T. Gagie, S. J. Puglisi, and A. Turpin, "Range quantile queries: Another virtue of wavelet trees," in *the Proceedings of International Symposium on String Processing and Information Retrieval (SPIRE)*, 2009, pp. 1–6.

[17] P. Ferragina, R. Giancarlo, G. Manzini, and M. Sciortino, "Boosting textual compression in optimal linear time," *Journal of the ACM*, vol. 52, no. 4, pp. 688–713, 2005, (also in CPM2001 and SODA2004).

[18] P. Elias, "Universal codeword sets and representations of the integers." *IEEE Transactions on Information Theory*, vol. IT-21, pp. 194–203, 1975.

[19] T. Gagie and G. Manzini, "Move-to-front, distance coding, and inversion frequencies revisited," *Theoretical Computer Science*, vol. 411, pp. 2925–2944, 2010.

[20] G. Manzini, "An analysis of the Burrows-Wheeler transform," *Journal of the ACM*, vol. 48, no. 3, pp. 407–430, 2001, (also in SODA1999).

[21] F. Claude and G. Navarro, "Practical rank/select queries over arbitrary sequences," in *the Proceedings of International Symposium on String Processing and Information Retrieval (SPIRE)*, 2008, pp. 176–187.

[22] R. González, S. Grabowski, V. Mäkinen, and G. Navarro, "Practical implementation of rank and select queries," in *The 4th Workshop on Efficient and Experimental Algorithms (WEA)*, 2005, pp. 27–38.

[23] J. Barbay and G. Navarro, "Compressed representations of permutations, and applications," in *the Proceedings of International Symposium on Theoretical Aspects of Computer Science (STACS)*, 2009, pp. 111–122.

[24] D. E. Knuth, *Sorting and Searching*, 2nd ed., ser. The Art of Computer Programming. Reading, MA: Addison-Wesley, 1998, vol. 3.

[25] P. G. Howard and J. S. Vitter, "Analysis of arithmetic coding for data compression," *Information Processing & Management*, vol. 28, no. 6, pp. 749–764, 1992.

APPENDIX

Lemma 10. $|B_{rle,\gamma}| \leq 2nH_0(B) + 2\log n + 2$

Proof:

$$|B_{rle,\gamma}| = 1 + \sum_{i=1}^{m} |\gamma(\ell_i)| = 1 + \sum_{i=1}^{m} (2\lfloor \log \ell_i \rfloor + 1) \leq 1 + \sum_{i=1}^{m} (2\log \ell_i + 1) = 2 \sum_{\ell_i > 1} \log \ell_i + m + 1 \leq 2m' \log \frac{n}{m'} + m + 1$$

$$\leq 2(t+1)\log \frac{n}{t+1} + 2t + 2 \leq 2(t+1)\log \frac{n}{t} + 2t + 2 = 2\left(t + t\log \frac{n}{t}\right) + 2\log \frac{n}{t} + 2 \leq 2nH_0(B) + 2\log n + 2$$

The second and third inequalities are due to the Jensen's inequality and the fact that $m \leq 2t+1$ and $m' \leq t+1$ (Lemma 3) and $x\log(n/x)$ increases over $1 \leq x \leq t+1 \leq \lfloor n/2 \rfloor + 1$. The last inequality is due to the fact that $t\log(n/t) + t \leq nH_0(B)$ in Lemma 3. ■

(a) normal texts

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Genome-10	40.57	45.82	88.91	88.26	78.76	**39.10**	**35.17**	**32.43**	68.14	53.08	61.21	76.11	44.14	48.07
Genome-20	38.98	44.91	74.49	74.58	75.46	**33.59**	**31.67**	**30.44**	54.35	48.13	56.37	62.14	41.53	42.90
Genome-40	38.14	44.50	66.73	67.11	73.67	**30.91**	**29.95**	**29.49**	45.32	45.06	53.69	53.99	39.38	40.37
Genome-80	37.95	44.25	62.58	62.80	72.49	**29.61**	**29.13**	**29.06**	40.69	43.58	52.24	49.34	38.61	39.12
Genome-160	37.90	44.10	60.22	60.38	71.80	**29.02**	**28.78**	**28.63**	38.19	42.84	51.47	46.82	38.23	38.49
Protein-10	24.92	33.69	66.82	65.92	58.86	**23.95**	**22.78**	**21.83**	49.14	37.85	36.64	48.90	26.05	27.60
Protein-20	22.50	32.53	55.43	55.22	55.81	**20.86**	**20.27**	**19.78**	37.21	33.48	31.27	37.04	22.95	23.64
Protein-40	21.18	31.96	49.35	49.41	54.41	**19.37**	**20.75**	**18.80**	30.86	31.11	28.37	30.41	22.28	21.68
Protein-80	20.56	31.62	46.10	46.17	53.19	**18.59**	**18.44**	**18.33**	27.40	29.83	26.86	26.73	20.56	20.68
Protein-160	20.21	31.48	44.36	44.44	52.74	**18.23**	**18.15**	**18.09**	29.28	29.12	26.07	24.77	21.29	20.16
English-10	62.91	108.49	191.73	188.62	190.34	**44.63**	**43.12**	**42.20**	94.58	71.82	86.30	110.72	64.08	65.05
English-20	50.68	105.13	167.59	166.93	184.12	**38.05**	**37.31**	**36.70**	70.10	61.68	69.49	79.43	50.62	51.63
English-40	53.71	103.73	155.23	155.13	181.38	**34.83**	**34.49**	**34.19**	56.83	56.48	70.61	71.37	54.04	54.33
English-80	47.97	103.19	148.82	148.93	180.93	**33.23**	**33.02**	**32.90**	49.62	53.46	60.19	62.48	51.45	51.10
English-160	47.36	102.52	145.16	145.32	180.64	**32.40**	**32.30**	**32.23**	45.79	52.02	55.38	53.54	44.48	44.97

(b) BWT of texts

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Genome-10	40.60	45.80	89.12	88.42	78.77	**39.15**	**35.22**	**32.50**	68.26	56.79	61.20	76.18	44.13	48.25
Genome-20	38.81	44.91	74.55	74.72	75.53	**33.65**	**31.74**	**30.45**	53.70	48.03	56.39	62.18	40.97	42.98
Genome-40	37.98	44.47	66.87	67.13	74.70	**30.96**	**29.98**	**29.49**	45.37	45.29	53.77	54.02	39.41	40.38
Genome-80	37.51	44.21	62.55	62.81	72.53	**29.61**	**29.13**	**29.07**	40.69	43.59	52.23	49.35	38.62	39.35
Genome-160	37.49	44.23	60.26	60.48	72.01	**29.03**	**29.04**	**28.72**	38.21	43.02	51.49	46.92	38.24	38.51
Protein-10	25.07	34.68	67.01	65.90	58.95	**24.08**	**22.86**	**21.88**	48.98	37.95	36.83	49.14	26.23	27.75
Protein-20	22.55	32.56	55.47	55.30	55.95	**20.95**	**20.35**	**19.82**	37.33	33.46	31.33	37.16	22.99	23.76
Protein-40	21.22	31.97	49.41	49.40	54.11	**19.37**	**19.07**	**18.85**	30.87	31.08	28.39	30.48	23.00	21.71
Protein-80	20.58	31.64	47.33	46.19	53.18	**18.61**	**18.48**	**18.33**	27.44	29.84	26.87	26.75	20.58	20.98
Protein-160	20.25	31.50	44.37	44.44	52.74	**18.23**	**18.15**	**18.09**	25.54	29.20	26.33	24.79	20.19	20.30
English-10	62.56	108.12	190.67	190.46	190.03	**44.46**	**42.87**	**41.70**	93.52	71.35	85.68	109.26	63.64	64.49
English-20	50.62	105.14	167.50	166.86	184.12	**38.01**	**37.26**	**36.70**	69.97	61.64	67.11	79.23	50.54	51.53
English-40	53.66	103.54	155.31	155.37	181.45	**34.90**	**34.57**	**38.92**	56.89	56.31	68.21	71.42	54.11	54.35
English-80	47.94	102.91	149.02	149.08	180.56	**33.24**	**33.07**	**32.90**	49.66	53.49	60.25	60.14	49.03	49.11
English-160	44.88	102.54	145.23	147.88	179.62	**32.40**	**32.31**	**32.26**	45.79	52.02	55.38	53.55	44.50	44.97

Table VIII
THE CONSTRUCTION TIME (SECOND) OF WAVELET TREES FOR LOW-ENTROPY DATA.

(a) normal texts

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Genome-10	**0.24**	0.44	12.57	12.70	0.67	0.68	**0.26**	**0.24**	6.37	0.39	0.39	6.36	**0.26**	0.68
Genome-20	**0.23**	0.46	12.81	13.60	0.64	0.43	**0.24**	**0.23**	6.50	0.35	0.35	6.50	**0.24**	0.43
Genome-40	**0.22**	0.44	13.29	14.43	0.61	0.32	**0.23**	**0.22**	6.85	0.32	0.32	6.85	**0.22**	0.32
Genome-80	**0.22**	0.44	13.53	15.17	0.59	0.26	**0.22**	**0.23**	7.18	0.31	0.31	7.18	**0.22**	0.26
Genome-160	**0.21**	0.43	13.32	15.77	0.59	0.24	**0.22**	**0.22**	7.56	0.30	0.30	7.56	**0.22**	0.24
Protein-10	**0.29**	0.87	23.36	24.28	1.31	0.90	**0.48**	**0.47**	12.22	0.78	0.49	6.71	**0.31**	0.65
Protein-20	**0.24**	0.86	24.04	25.76	1.24	0.64	**0.44**	**0.43**	12.37	0.68	0.40	6.31	**0.25**	0.42
Protein-40	**0.22**	0.85	24.44	27.26	1.19	0.51	**0.42**	**0.42**	12.97	0.63	0.34	6.30	**0.23**	0.31
Protein-80	**0.21**	0.85	24.79	28.64	1.16	0.45	**0.41**	**0.41**	13.43	0.59	0.31	6.53	**0.21**	0.25
Protein-160	**0.20**	0.84	25.44	30.66	1.15	0.43	**0.41**	**0.41**	14.13	0.58	0.29	6.71	**0.21**	0.23
English-10	**0.33**	1.95	49.98	55.62	2.70	0.83	**0.53**	**0.52**	13.58	0.85	0.56	7.61	**0.34**	0.46
English-20	**0.27**	1.95	52.02	60.37	2.60	0.62	**0.48**	**0.47**	13.63	0.73	0.44	6.91	**0.27**	0.33
English-40	**0.24**	1.96	54.37	65.05	2.56	0.52	**0.45**	**0.45**	13.99	0.66	0.37	6.76	**0.24**	0.27
English-80	**0.22**	1.96	57.19	69.45	2.54	0.47	**0.44**	**0.44**	14.53	0.62	0.33	6.97	**0.22**	0.24
English-160	**0.22**	1.96	59.82	73.27	2.51	0.45	**0.43**	**0.43**	15.05	0.60	0.31	7.18	**0.22**	0.23

(b) BWT of texts

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Genome-10	**0.24**	0.44	12.52	12.66	0.69	0.68	**0.26**	**0.24**	6.36	0.39	0.39	6.33	**0.26**	0.68
Genome-20	**0.23**	0.44	12.78	13.57	0.65	0.43	**0.24**	**0.23**	6.51	0.35	0.35	6.49	**0.24**	0.43
Genome-40	**0.23**	0.45	13.29	14.41	0.61	0.32	**0.22**	**0.22**	6.86	0.32	0.32	6.85	**0.22**	0.31
Genome-80	**0.22**	0.44	13.52	15.18	0.62	0.26	**0.22**	**0.22**	7.18	0.31	0.31	7.18	**0.22**	0.26
Genome-160	**0.22**	0.44	13.31	15.79	0.59	0.24	**0.22**	**0.22**	7.58	0.30	0.30	7.56	**0.22**	0.24
Protein-10	**0.29**	0.87	23.33	24.25	1.32	0.90	**0.48**	**0.47**	12.22	0.78	0.49	6.71	**0.30**	0.64
Protein-20	**0.24**	0.85	24.06	25.76	1.23	0.63	**0.44**	**0.43**	12.37	0.68	0.39	6.31	**0.25**	0.42
Protein-40	**0.22**	0.85	24.41	27.18	1.19	0.51	**0.42**	**0.42**	12.95	0.63	0.34	6.28	**0.23**	0.31
Protein-80	**0.21**	0.84	24.74	28.62	1.16	0.45	**0.41**	**0.41**	13.43	0.59	0.31	6.52	**0.21**	0.25
Protein-160	**0.20**	0.84	25.39	30.63	1.15	0.43	**0.41**	**0.40**	14.12	0.58	0.29	6.70	**0.21**	0.23
English-10	**0.33**	1.95	51.29	58.81	2.67	0.83	**0.53**	**0.52**	13.66	0.84	0.56	7.63	**0.33**	0.47
English-20	**0.27**	1.96	52.97	62.91	2.60	0.62	**0.48**	**0.47**	13.62	0.73	0.44	6.91	**0.27**	0.33
English-40	**0.24**	1.95	54.93	66.97	2.57	0.52	**0.45**	**0.45**	14.00	0.66	0.37	6.75	**0.24**	0.27
English-80	**0.22**	1.95	57.28	70.75	2.53	0.47	**0.44**	**0.44**	14.55	0.62	0.33	6.96	**0.22**	0.24
English-160	**0.22**	1.95	59.63	73.94	2.51	0.45	**0.43**	**0.43**	15.09	0.60	0.31	7.19	**0.22**	0.22

Table IX
THE TIME COST (SECOND) FOR 1,000,000 MEMBERSHIP QUERIES OVER LOW-ENTROPY DATA.

(a) normal texts

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Genome-10	**0.43**	0.81	19.34	19.44	1.10	1.14	**0.46**	**0.43**	9.77	0.65	0.65	9.77	**0.46**	1.12
Genome-20	**0.41**	0.79	19.62	20.72	1.03	0.72	**0.42**	**0.41**	9.93	0.58	0.57	9.94	**0.43**	0.73
Genome-40	**0.40**	0.79	20.28	21.95	0.98	0.54	**0.41**	**0.40**	10.42	0.52	0.52	10.41	**0.41**	0.55
Genome-80	**0.39**	0.79	20.57	23.04	0.96	0.46	**0.40**	**0.39**	10.92	0.49	0.49	10.93	**0.40**	0.47
Genome-160	**0.39**	0.79	20.21	23.86	0.94	0.42	**0.39**	**0.39**	11.47	0.48	0.48	11.48	**0.39**	0.43
Protein-10	**0.53**	1.57	35.81	37.12	2.15	1.50	**0.87**	**0.84**	18.71	1.28	0.82	10.32	**0.55**	1.08
Protein-20	**0.45**	1.54	36.69	39.24	2.00	1.09	**0.80**	**0.79**	18.84	1.11	0.65	9.65	**0.46**	0.72
Protein-40	**0.41**	1.53	37.19	41.36	1.91	0.90	**0.77**	**0.76**	19.74	1.01	0.56	9.57	**0.42**	0.54
Protein-80	**0.39**	1.53	37.61	43.35	1.85	0.81	**0.75**	**0.74**	20.45	0.95	0.50	9.93	**0.39**	0.45
Protein-160	**0.38**	1.53	38.52	46.23	1.83	0.77	**0.74**	**0.73**	21.38	0.92	0.47	10.19	**0.38**	0.41
English-10	**0.60**	3.55	76.31	84.63	4.41	1.40	**0.96**	**0.94**	20.87	1.40	0.95	11.74	**0.62**	0.81
English-20	**0.49**	3.55	79.26	91.53	4.25	1.08	**0.86**	**0.85**	20.78	1.19	0.73	10.59	**0.50**	0.59
English-40	**0.44**	3.56	82.54	98.27	4.15	0.92	**0.82**	**0.81**	21.31	1.07	0.61	10.30	**0.44**	0.49
English-80	**0.41**	3.56	86.59	104.66	4.10	0.84	**0.79**	**0.79**	22.10	1.00	0.54	10.61	**0.41**	0.44
English-160	**0.40**	3.56	90.40	110.32	4.08	0.80	**0.78**	**0.77**	22.79	0.96	0.50	10.93	**0.40**	0.41

(b) BWT of texts

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Genome-10	**0.44**	0.80	19.28	19.39	1.10	1.10	**0.46**	**0.43**	9.75	0.65	0.65	9.73	**0.46**	1.10
Genome-20	**0.41**	0.80	19.56	20.69	1.02	0.72	**0.42**	**0.41**	9.93	0.57	0.57	9.92	**0.43**	0.72
Genome-40	**0.40**	0.81	20.27	21.91	0.98	0.54	**0.40**	**0.40**	10.40	0.52	0.53	10.42	**0.41**	0.55
Genome-80	**0.39**	0.79	20.57	23.03	0.95	0.46	**0.40**	**0.39**	10.93	0.49	0.50	10.92	**0.40**	0.47
Genome-160	**0.39**	0.79	20.20	23.88	0.94	0.42	**0.39**	**0.39**	11.47	0.48	0.48	11.48	**0.39**	0.43
Protein-10	**0.53**	1.57	35.76	37.08	2.16	1.50	**0.87**	**0.84**	18.72	1.28	0.83	10.31	**0.55**	1.06
Protein-20	**0.45**	1.54	36.72	39.25	1.99	1.09	**0.80**	**0.78**	18.83	1.11	0.66	9.64	**0.46**	0.72
Protein-40	**0.41**	1.53	37.15	41.29	1.90	0.90	**0.76**	**0.75**	19.71	1.01	0.55	9.56	**0.42**	0.54
Protein-80	**0.39**	1.52	37.55	43.29	1.86	0.81	**0.75**	**0.74**	20.42	0.95	0.50	9.90	**0.39**	0.46
Protein-160	**0.38**	1.52	38.47	46.20	1.83	0.77	**0.74**	**0.73**	21.36	0.92	0.47	10.19	**0.38**	0.41
English-10	**0.60**	3.55	78.36	89.34	4.40	1.41	**0.96**	**0.94**	20.96	1.40	0.95	11.80	**0.61**	0.81
English-20	**0.49**	3.56	80.67	95.33	4.25	1.08	**0.86**	**0.85**	20.81	1.19	0.73	10.60	**0.50**	0.59
English-40	**0.44**	3.56	83.42	101.16	4.14	0.92	**0.81**	**0.81**	21.31	1.06	0.61	10.30	**0.44**	0.49
English-80	**0.41**	3.55	86.77	106.60	4.11	0.84	**0.79**	**0.78**	22.12	1.00	0.54	10.62	**0.41**	0.44
English-160	**0.40**	3.55	90.14	111.31	4.07	0.80	**0.78**	**0.78**	22.85	0.96	0.50	10.92	**0.40**	0.41

Table X
THE TIME COST (SECOND) FOR 1,000,000 RANK QUERIES OVER LOW-ENTROPY DATA.

(a) normal texts

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Genome-10	**0.62**	1.15	30.14	31.02	1.60	1.37	**0.65**	**0.62**	15.14	0.89	0.90	15.04	**0.65**	1.38
Genome-20	**0.57**	1.11	30.82	33.20	1.50	0.85	**0.58**	**0.57**	15.61	0.79	0.79	15.62	**0.59**	0.86
Genome-40	**0.56**	1.11	32.11	35.38	1.46	0.68	**0.56**	**0.55**	16.62	0.73	0.73	16.63	**0.57**	0.68
Genome-80	**0.55**	1.10	32.88	37.36	1.44	0.60	**0.56**	**0.55**	17.63	0.71	0.71	17.63	**0.56**	0.61
Genome-160	**0.55**	1.10	32.56	38.86	1.43	0.57	**0.55**	**0.55**	18.69	0.69	0.70	18.69	**0.55**	0.58
Protein-10	**0.61**	2.13	54.17	57.55	3.07	1.63	**1.13**	**1.11**	27.68	1.63	0.93	13.97	**0.63**	1.05
Protein-20	**0.55**	2.13	57.10	62.25	2.98	1.30	**1.09**	**1.07**	29.12	1.49	0.79	14.21	**0.56**	0.75
Protein-40	**0.52**	2.13	58.78	66.56	2.91	1.16	**1.07**	**1.06**	31.26	1.42	0.71	14.80	**0.53**	0.62
Protein-80	**0.51**	2.13	60.30	70.39	2.88	1.10	**1.06**	**1.05**	32.98	1.37	0.67	15.76	**0.51**	0.56
Protein-160	**0.50**	2.13	62.38	75.74	2.87	1.07	**1.05**	**1.05**	34.78	1.36	0.64	16.51	**0.50**	0.53
English-10	**0.69**	4.91	116.25	131.86	6.78	1.58	**1.23**	**1.22**	30.21	1.79	1.05	15.23	**0.70**	0.84
English-20	**0.60**	4.94	124.01	145.61	6.72	1.34	**1.17**	**1.17**	31.95	1.61	0.87	15.28	**0.61**	0.67
English-40	**0.57**	4.98	131.44	159.13	6.68	1.22	**1.15**	**1.15**	33.76	1.52	0.79	15.78	**0.57**	0.61
English-80	**0.55**	4.96	139.05	171.14	6.69	1.17	**1.13**	**1.14**	35.56	1.47	0.73	16.78	**0.55**	0.57
English-160	**0.55**	4.96	145.35	180.59	6.60	1.15	**1.13**	**1.15**	37.00	1.45	0.70	17.63	**0.54**	0.55

(b) BWT of texts

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Genome-10	**0.63**	1.18	29.99	30.86	1.62	1.35	**0.65**	**0.62**	15.05	0.90	0.90	14.93	**0.65**	1.36
Genome-20	**0.57**	1.14	30.73	33.11	1.51	0.85	**0.58**	**0.57**	15.61	0.78	0.79	15.58	**0.59**	0.85
Genome-40	**0.56**	1.14	32.12	35.34	1.47	0.67	**0.56**	**0.55**	16.60	0.73	0.74	16.62	**0.57**	0.68
Genome-80	**0.56**	1.12	32.85	37.37	1.44	0.61	**0.56**	**0.55**	17.62	0.70	0.71	17.62	**0.56**	0.61
Genome-160	**0.55**	1.12	32.55	38.88	1.43	0.58	**0.55**	**0.55**	18.67	0.70	0.69	18.68	**0.56**	0.58
Protein-10	**0.61**	2.16	54.14	57.50	3.06	1.63	**1.13**	**1.11**	27.67	1.62	0.93	13.97	**0.63**	1.04
Protein-20	**0.55**	2.14	57.19	62.30	2.97	1.30	**1.08**	**1.07**	29.10	1.49	0.81	14.20	**0.56**	0.74
Protein-40	**0.52**	2.14	58.75	66.50	2.90	1.16	**1.06**	**1.06**	31.23	1.42	0.71	14.77	**0.53**	0.62
Protein-80	**0.51**	2.14	60.16	70.27	2.88	1.10	**1.06**	**1.05**	32.96	1.37	0.67	15.76	**0.51**	0.56
Protein-160	**0.50**	2.12	62.35	75.72	2.86	1.09	**1.06**	**1.05**	34.78	1.36	0.64	16.53	**0.51**	0.53
English-10	**0.69**	5.07	119.83	139.69	6.80	1.58	**1.23**	**1.22**	30.44	1.77	1.05	15.25	**0.70**	0.84
English-20	**0.60**	5.09	126.48	151.92	6.76	1.33	**1.17**	**1.17**	31.81	1.61	0.87	15.29	**0.61**	0.67
English-40	**0.57**	5.08	132.97	163.44	6.71	1.22	**1.15**	**1.14**	33.58	1.52	0.78	15.79	**0.57**	0.60
English-80	**0.55**	5.01	139.07	173.05	6.72	1.18	**1.14**	**1.14**	35.38	1.47	0.73	16.80	**0.55**	0.57
English-160	**0.54**	4.99	144.79	181.59	6.60	1.15	**1.13**	**1.13**	36.97	1.46	0.70	17.63	**0.55**	0.55

Table XI
THE TIME COST (SECOND) FOR 1,000,000 SELECT QUERIES OVER LOW-ENTROPY DATA.