On Tries, Suffix Trees, and Universal Variable-Length-to-Block Codes

Yuriy A. Reznik *
RealNetworks, Inc.
2601 Elliott Avenue, Seattle, WA 98121
E-mail: yreznik@real.com

October 30, 2001

Abstract

We show how a digital tree (trie) structure can be used for both parsing and encoding (in a Variable-Length to Block (VB) or Variable-Length to Variable-Length (VV) fashion) sequences of symbols from a stochastic source. As an example, we construct a simple VB code based on a *fixed database* adaptation model. We show that the average redundancy rate of this code for memoryless sources is asymptotically $\frac{C+\delta(n)}{\log n} + O\left(\log^{-2} n\right)$, where n is the length of a training sequence, C is a constant for which we derive a precise expression, and $\delta(n)$ is an oscillating function of a small magnitude.

1 Introduction

If we designate a set of n distinct strings as $S = \{s_1, \ldots, s_n\}$, and assume that each string is a sequence of symbols from a finite alphabet $\Sigma = \{\alpha_1, \ldots, \alpha_v\}$, $|\Sigma| = v$, then a *trie* (or digital tree [1, 7]) $T(S)$ over S can be constructed as follows. If $n = 0$, the trie is empty. If $n = 1$ (i.e. S has only one string), the trie is an *external node* containing a pointer to this single string in S. If $n > 1$, the trie is an *internal node* containing v pointers (or *branches*) to the child tries: $T(S_1), \ldots, T(S_v)$, where each set S_i $(1 \leqslant i \leqslant v)$ contains suffixes of all strings from S that begin with a corresponding first symbol. For example, if a string $s_j = u_j w_j$ (u_j is a first symbol, and w_j is a

*This author is also on leave from the Institute of Mathematical Machines and Systems of the National Academy of Sciences of Ukraine.

string containing the remaining symbols of s_j), and $u_j = \alpha_i$, then the string w_j will go into S_i. Thus, after all child tries $T(S_1), \ldots, T(S_v)$ are recursively processed, we arrive at a tree-like data structure, where the original strings $s_1, \ldots, s_n$ can be uniquely identified by the paths from the root node to non-empty external nodes (see Fig. 1.a).

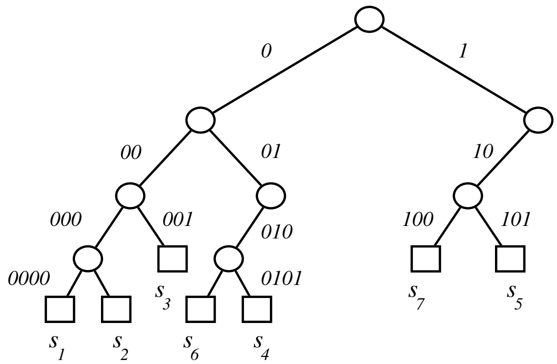
In a special case, when strings $s_1, \ldots, s_n$ represent successive suffixes of a given string $Z = s_1 = z_1 s_2 = \ldots = z_1 \ldots z_{n-1} s_n$, $z_i \in \Sigma$, the resulting structure $T(Z)$ is called a *suffix tree*.

We already know that the paths from root to external nodes in a trie represent the *shortest distinguishing prefixes* of the corresponding strings in S. At the same time, a *set of these prefixes* (or paths) in a trie does not necessarily constitute a complete *prefix code*. For example, in order to separate strings s_4 and s_6 (or s_5 and s_7), a trie on Fig. 1.a must use *one-way branches*, creating patterns that cannot be successfully matched (and/or encoded) by the trie.

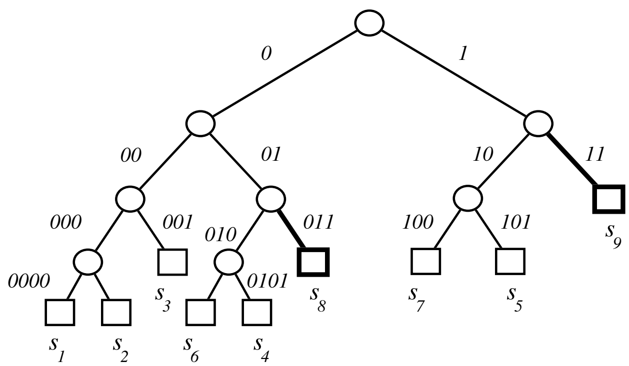
The last problem can be trivially solved by adding the missing nodes, or equivalently, by extending the set of strings in the trie to a (smallest possible) complete prefix set. We illustrate this procedure on Fig. 1.b, and call the resulting structure an *extended trie* (or *extended suffix tree* in a case of processing single sequence).

Extended tries can be used to implement various non-block source codes. The main idea is to use such a trie to parse a sequence of symbols that we encode, and once an external node is reached (i.e. longest match found), transmit an index of this node, and restart the search process. Depending on a technique used for encoding indices of external nodes in a trie, the resulting code should have either variable-length-to-block (VB) or variable-length-to-variable-length (VV) structure [8]. Also, depending on the nature of the training set used to create an extended trie and/or frequency of its updates, we may be dealing with a number of possible static or adaptive (single-source or universal) coding algorithms.

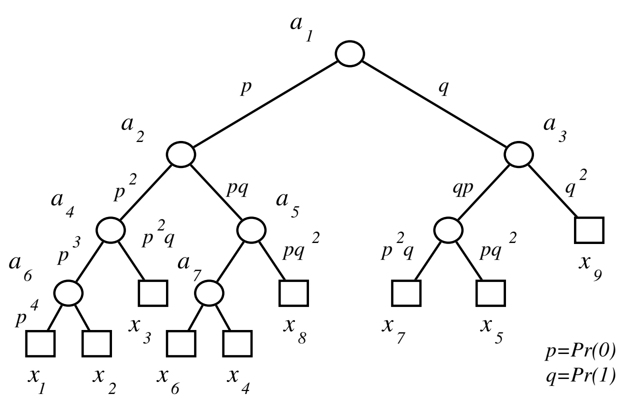
In this abstract we propose and study an algorithm that can be described as a variant of a *fixed-database universal data compression* scheme of Wyner and Ziv [13]. Both encoder (on the transmitter side) and decoder (on the receiver side) have access to a training pattern Z, representing a finite sequence of symbols produced by the source. This training sequence is used to construct an extended suffix tree $T(Z)$. The extended suffix tree is then used to define a VB code (see Fig. 1.c), which in turn, is used to encode/decode the sequences of symbols needed to be transmitted.



(a) A trie built from 7 suffixes of a string Z=000010100...



(b) An extended version of the same trie.



(c) Extended trie used as a VB code for a Bernoulli source.

Figure 1: A trie built from the first 7 suffixes of a binary string $Z = 000010100\ldots$ The same trie is then extended (b), and used to implement a variable-length-to-block code for a Bernoulli source (c).

2 Definitions and Main Results

Let n be the length of a training sequence $Z \in \Sigma^n$, $X(T)$ - the total number of external nodes (including ones added to eliminate one-way branches) in an extended suffix tree $T = T(Z)$ over the training sequence, $|x_j|$ - a depth of a j-th external node in the suffix tree (i.e. the length of a path from the root to an external node x_j).

The *average delay* of a non-block code defined by a tree T is (cf. [5, 4, 10]):

$$d(T) = \sum_{j=1}^{X(T)} \Pr(x_j)\, |x_j|, \tag{1}$$

where $\Pr(x_j)$ is the probability of string corresponding to a path from root to an external node x_j (in a memoryless case, $\Pr(x_j) = \prod_{i=1}^{|x_j|} \Pr(z_{j_i})$, where $z_{j_i} \in \Sigma$ are symbols occurring in the path from root to x_j).

If we designate a code assigned to an external node x_j as $f(x_j)$, and its length as $|f(x_j)|$, then the cost of the code defined by a tree T and mapping $f(.)$ is

$$C(T, f) = \frac{1}{d(T)} \sum_{j=1}^{X(T)} \Pr(x_j)\, |f(x_j)|\,. \tag{2}$$

In a case of a VB code based on tree T, it is sufficient to take $|f(x_j)| = \lceil \log X(T) \rceil < \log X(T) + 1$ bits (or nats) to transmit indices of its nodes. Hence, the cost of such a code [5, 4]

$$C(T) = \frac{1}{d(T)} \lceil \log X(T) \rceil < \frac{1}{d(T)} \left(\log X(T) + 1\right). \tag{3}$$

The redundancy of this code (for a given tree T or a corresponding training sequence):

$$R(T) = C(T) - h, \tag{4}$$

where h is the entropy of the source. The average redundancy (over all possible trees T or training sequences of length n produced by a source), therefore, is

$$\begin{aligned} \bar{R} &:= E\{R(T)\} = E\{C(T)\} - h = \frac{1}{E\{d(T)\}} E\left\{\lceil \log X(T) \rceil\right\} - h \qquad (5) \\ &< \frac{1}{\bar{d}} \left(E\left\{\log X(T)\right\} + 1\right) - h \leqslant \frac{1}{\bar{d}} \left(\log \bar{X} + 1\right) - h, \qquad (6) \end{aligned}$$

where the last transition is due to Jensen inequality for $E\{\log(.)\}$.

We now present our main result concerning the average redundancy rate $\bar{R}$ of our code for memoryless sources.

Theorem 1 *Consider a memoryless source over binary alphabet $\Sigma = \{0, 1\}$ and probabilities $p = \Pr\{0\}, q = \Pr\{1\}, p + q = 1$. Then the average redundancy rate of the VB code constructed by an extended suffix tree over a fixed database of size n is:*

$$\begin{aligned}\bar{R} &< \frac{1}{\bar{d}}\left(\log \bar{X} + 1\right) - h & (7)\\ &= \frac{h}{\log n}\left(h + \log e\,(1+\gamma) - \frac{h_2}{2h} + \log\frac{\log e}{h} + 1 + \delta_1(n,p)\right) + O\left(\frac{1}{\log^2 n}\right),\end{aligned}$$

where: $h = -p\log p - q\log q$ is the entropy of the source, $h_2 = p\log^2 p + q\log^2 q$, $\gamma = 0.57721\ldots$ is the Euler constant, and $\delta_1(n,p)$ is a fluctuating function of small amplitude. The base of logarithms depends on a unit of information (e.g. bits or nats) used to express entropy, delay, and redundancy.

Combining all the constants in (7), we can show that the redundancy of our code is decreasing at a rate of $O(\frac{1}{\log n})$, which is a noticeable improvement over a $O(\frac{\log\log n}{\log n})$ rate of the fixed-database version of a Lempel-Ziv code [13]. While a similar order of improvement has already been obtained with the help of two training sequences [12], we believe that our algorithm should be more practical, as it achieves the same result using only one sequence.

3 Sketch of Proof

Let $A(T)$ to be the number of internal nodes in the tree T. We know that for a binary tree: $X(T) = A(T) + 1$ (proof by induction, see, e.g. [6]).

According to Krichevsky [8, Lemma 2.2.1]: $d(T) = \sum_{j=1}^{X(T)} \Pr(x_j)|x_j| = \sum_{i=1}^{A(T)} \Pr(a_i)$, where a_i represent internal nodes in the tree. Therefore, we only need to estimate $A(T)$, and $\sum \Pr(a_i)$.

According to Jacquet and Szpankowski [3] statistics in (independent) tries and prefix trees are essentially the same, so we can use the former. Using techniques from [7, 11, 9] for a trie constructed in Bernoulli model we can write:

$$\bar{A} = \sum_{r=1}^{\infty}\sum_{s=0}^{r}\binom{s}{r}\sum_{k=2}^{n}\binom{k}{n}\left(p^s q^{r-s}\right)^k\left(1 - p^s q^{r-s}\right)^{n-k}$$

$$
\begin{aligned}
&= \sum_{r=1}^{\infty}\sum_{s=0}^{r}\binom{s}{r}\left(1-\left(1-p^s q^{r-s}\right)^n - n\,p^s q^{r-s}\left(1-p^s q^{r-s}\right)^{n-1}\right);\\
\bar{d} &= \sum_{r=1}^{\infty}\sum_{s=0}^{r}\binom{s}{r}\sum_{k=2}^{n}\binom{k}{n}\left(p^s q^{r-s}\right)^k\left(1-p^s q^{r-s}\right)^{n-k} p^s q^{r-s}\\
&= \sum_{r=1}^{\infty}\sum_{s=0}^{r}\binom{s}{r}p^s q^{r-s}\left(1-\left(1-p^s q^{r-s}\right)^n - n\,p^s q^{r-s}\left(1-p^s q^{r-s}\right)^{n-1}\right).
\end{aligned}
$$

Where after approximation ($n \to \infty$)

$$\left(1-p^s q^{r-s}\right)^n \sim e^{-np^s q^{r-s}},$$

decomposition of the exponent [7]:

$$e^{-x} = 1 - x + \frac{1}{2\pi i}\int_{-3/2-i\infty}^{-3/2+i\infty}\Gamma(z)\,x^{-z}dz.$$

and evaluation of the residuals in the resulting integrals we arrive at:

$$
\begin{aligned}
\bar{A} &= n\frac{\log e}{h} + O(1);\\
\bar{d} &= \frac{1}{h}\left(\log n - h - \log e\,(1+\gamma) + \frac{h_2}{2h}\right) + \delta(n,p) + O\left(\frac{1}{n}\right),
\end{aligned}
$$

where $\delta(n,p)$ is an oscillating function of small magnitude, $\gamma = 0.57721\ldots$ is the Euler constant, and h is the entropy of the source.

The resulting expression for the expected redundancy of our code:

$$
\begin{aligned}
\bar{R} &< \frac{1}{\bar{d}}\left(\log\left(\bar{A}+1\right)+1\right) - h\\
&= \frac{1}{\bar{d}}\left(\log n + \log\frac{\log e}{h} + 1\right) - h + O\left(\frac{1}{n}\right)\\
&= \frac{h}{\log n}\left(h + \log e\,(1+\gamma) - \frac{h_2}{2h} + \log\frac{\log e}{h} + 1 + \delta_1(n,p)\right) + O\left(\frac{1}{\log^2 n}\right).
\end{aligned}
$$

where

$$\delta_1(n,p) = -h\delta(n,p).$$

In the final version of this paper we intend to extend our analysis to *Markovian* sources, which should be possible using the results of Jacquet and Szpankowski [2, 3].

References

[1] E. Fredkin, Trie Memory, *Comm. ACM*, 3 (1960) 490–500.

[2] P. Jacquet, and W. Szpankowski, Analysis of digital tries with Markovian dependency, *IEEE Trans. Inform. Theory*, 37 (1991) 1470–1475.

[3] P. Jacquet and W. Szpankowski, Autocorrelation on Words and Its Applications. Analysis of Suffix Trees by String-Ruler Approach, *J. Combin. Theory Ser. A*, 66, (1994) 237–269.

[4] F. Jelinek, and K. S. Schneider, On Variable-Length-to-Block Coding, *IEEE Trans. Inform. Theory*, 18 (6) (1972) 765–774.

[5] G. L. Khodak, Redundancy Estimates for Word-Based Codes for Bernoulli Sequences, *Probl. Inform. Trans.*, 8, (2) (1972) 21–32

[6] D. Knuth, *The Art of Computer Programming. Fundamental Algorithms. Vol. 1* (Addison-Wesley, Reading MA, 1968).

[7] D. Knuth, *The Art of Computer Programming. Sorting and Searching. Vol. 3* (Addison-Wesley, Reading MA, 1973).

[8] R. E. Krichevsky, Universal Data Compression and Retrieval. (Kluwer, Norwell, MA, 1993).

[9] Yu. A. Reznik, Some Results on Tries with Adaptive Branching, Proc. 6th Annual International Computing and Combinatorics Conference, *LNCS*, 1858 (Springer-Verlag, New York, 2000) 148–158.

[10] S. A. Savari, and R. G. Gallager, Generalized Tunstall Codes for Sources with Memory, *IEEE Trans. Inform. Theory*, 43 (2) (1997) 658–668.

[11] W. Szpankowski, Some Results on V-ary Asymmetric Tries, *J. Algorithms*, 9, (1988) 224–244.

[12] A.D. Wyner and A.J. Wyner, Improved Redundancy of a Version of the Lempel-Ziv Algorithm, *IEEE Trans. Inform. Th.*, 41 (1995) 723–732.

[13] A.D. Wyner and J. Ziv, Fixed Data Base Version of the Lempel-Ziv Data Compression Algorithm, *IEEE Trans. Information Theory*, 37 (1991) 878–880.

[14] J. Ziv and A. Lempel, A Universal Algorithm for Sequential Data Compression, *IEEE Trans. Information Theory*, 23 (1977) 337–343.