

# Complete Variable-Length “Fix-Free” Codes

David Gillman\*

Institute for Mathematics & Its Applications

University of Minnesota

Minneapolis, MN 55455

and

Ronald L. Rivest†

Laboratory for Computer Science

Massachusetts Institute of Technology

Cambridge, MA 02139

November 22, 1994

## Abstract

A set of codewords is *fix-free* if it is both prefix-free and suffix-free: no codeword in the set is a prefix or a suffix of any other. A set of codewords  $\{x_1, x_2, \dots, x_n\}$  over a  $t$ -letter alphabet  $\Sigma$  is said to be *complete* if it satisfies the Kraft inequality with equality, so that

$$\sum_{1 \leq i \leq n} t^{-|x_i|} = 1 .$$

The set  $\Sigma^k$  of all codewords of length  $k$  is obviously both fix-free and complete. We show, surprisingly, that there are other examples of complete fix-free codes, ones whose codewords have a variety of lengths. We discuss such variable-length (complete) fix-free codes and techniques for constructing them.

## 1 Introduction

While investigating the properties of Huffman codes [1], we became interested in codes that were both prefix-free and suffix-free (or “fix-free”). Fix-free codes have a number of interesting properties. For example, a word constructed by concatenating together a number of codewords from a fix-free code can be uniquely parsed from either end. (And fix-free codes

---

\*Supported by NSF grant 9212184-CCR and Darpa contract N00014-92-J-1799. Email address: [gillman@ima.umn.edu](mailto:gillman@ima.umn.edu)

†Supported by NSF grants CCR-8914428 and CCR-9310888, and the Siemens Corporation. Email address: [rivest@theory.lcs.mit.edu](mailto:rivest@theory.lcs.mit.edu)

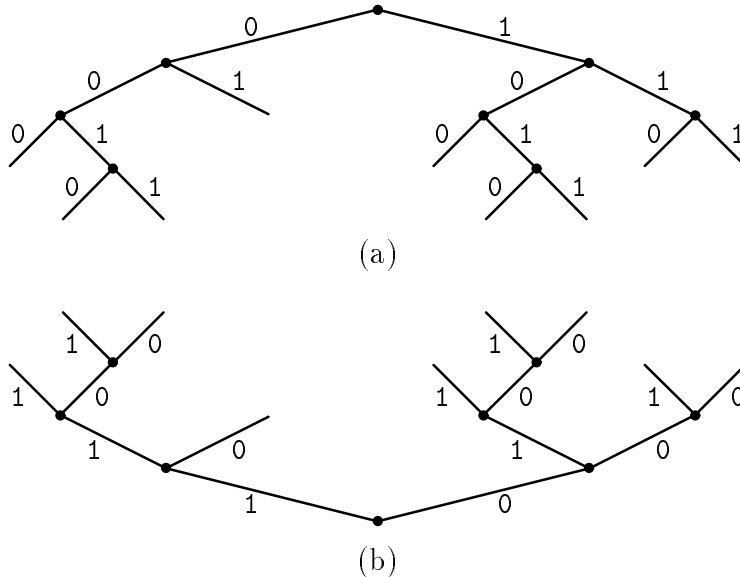


Figure 1: (a) Prefix tree and (b) suffix tree for fix-free code  $A = \{01, 000, 100, 110, 111, 0010, 0011, 1010, 1011\}$ .

have terrible synchronization properties: being suffix-free means that if the channel drops a bit the receiver can not easily resynchronize.<sup>1</sup>) However, it was not clear to us whether fix-free codes could be as efficient as ordinary prefix-free codes. We thus became interested in *complete* “fix-free” codes, as defined in the abstract above. Preliminary investigation led us to conjecture that a complete fix-free code over a given  $t$ -letter alphabet  $\Sigma$  must be of the form  $\Sigma^k$  for some integer  $k$ .

**Conjecture.** A complete fix-free code over a given  $t$ -letter alphabet  $\Sigma$  must be of the form  $\Sigma^k$  for some integer  $k$ . That is, there are no variable-length complete fix-free codes.

We attempted to prove this conjecture, but were surprised to find out that it is false. Here is the first counterexample we found, over a binary alphabet:

$$A = \{01, 000, 100, 110, 111, 0010, 0011, 1010, 1011\}$$

The “prefix tree” and “suffix tree” for the code  $A$  are given in Figure 1.

Any such counterexample to the conjecture generates a family of counterexamples by forming products: let  $A^k$  be the set of concatenations of  $k$  words from  $A$ . Since  $A$  is fix-free,  $A^k$  must also be fix-free.

Given that the conjecture is false, it is natural to ask for general techniques for constructing fix-free codes, much as one can construct Huffman codes. However, the problem of constructing fix-free codes seems much more difficult, and we do not know how to generate all such codes. For use in applications, it would seem advantageous to have codes with arbitrarily large ratios between the lengths of the longest and shortest codewords. In

---

<sup>1</sup>The receiver first parses the transmission into codewords and then decodes each one. Because of the suffix-free property, the receiver will parse incorrectly from the dropped bit onward, and the end of a parsed word will never coincide with the end of a source word. This is what we mean by failure to resynchronize.

the set  $A$  just described, the ratio of the lengths of the longest and shortest words is 2. In the next section we show that  $A$  is an example of a general construction that gives a ratio of longest word to shortest word approaching 3. We further generalize this by developing a recursive construction which gives arbitrarily large ratios. We leave as an open problem the question of constructing “efficient” fix-free codes, given a probability distribution on a source alphabet.

## 2 General Construction Techniques

### 2.1 First Construction Method

Here is a general construction of a complete fix-free code  $A$  over a  $t$ -letter alphabet  $\Sigma$ .

**Theorem 1** *Let  $S$  be any subset of  $\Sigma^k$ , and  $p$  be a fixed positive integer less than  $k$ , such that for any two (not necessarily distinct) words of  $S$  (say,  $x = x[1]x[2] \dots x[k]$  and  $y = y[1]y[2] \dots y[k]$ ), we have that*

$$x[i] \neq y[i + p] \text{ for some } i, 1 \leq i \leq k - p . \quad (1)$$

*Then the set of codewords*

$$F_{k,p}(S) = S \cup \Sigma^p S \Sigma^p \cup (\Sigma^{k+p} - S \Sigma^p - \Sigma^p S) \quad (2)$$

*is both fix-free and complete.*

**Proof:** Condition (1) ensures that no element of  $S$  can be a prefix or a suffix of any string in  $\Sigma^p S \Sigma^p$ . It is immediate that no element of  $S$  is a prefix or a suffix of any string in  $(\Sigma^{k+p} - S \Sigma^p - \Sigma^p S)$ , and that no element of  $(\Sigma^{k+p} - S \Sigma^p - \Sigma^p S)$  is a prefix or a suffix of any string in  $\Sigma^p S \Sigma^p$ . Thus  $F_{k,p}(S)$  is fix-free.

To show that  $F_{k,p}(S)$  is complete, we verify that the Kraft inequality is tight. Condition (1) ensures that  $S \Sigma^p$  and  $\Sigma^p S$  are disjoint. Let  $s = |S|$ . Then  $F_{k,p}(S)$  contains  $s$  words of length  $k$ ,  $st^{2p}$  words of length  $k + 2p$ , and  $t^{k+p} - st^{p+1}$  words of length  $k + p$ . We thus have

$$\sum_{x \in F_{k,p}(S)} t^{-|x|} = st^{-k} + st^{2p}t^{-k-2p} + (t^{k+p} - st^{p+1})t^{-k-p} \quad (3)$$

$$= st^{-k} + st^{-k} + (1 - st^{-k+1}) \quad (4)$$

$$= 1 , \quad (5)$$

so that  $F_{k,p}$  is complete. ■

The ratio of the longest word to the shortest word in this fix-free code is  $(2p + k)/k$ ; if we choose  $p = k - 1$  (which we certainly can do for some  $S$ ), the ratio is  $3 - 2/k$ , which approaches 3 as  $k$  goes to infinity. It is possible to pick a set  $S$  satisfying condition (1) for any  $k$  and any  $p < k$ ; for example, let  $S$  be a set containing a single element  $x$  for which  $x_1 \neq x_{p+1}$ . In our example from the previous section the set  $A$  arises by letting  $S = \{01\}$  and  $p = 1$  (so that  $k = 2$ ).

## 2.2 Second (Recursive) Construction Method

We now generalize the above construction. We describe how to construct a complete fix-free code recursively and prove that the code is fix-free as long as there exists a number  $m$  and sets of codewords  $S_1, \dots, S_r$  such that:

1. For  $i = 1, 2, \dots, r$ , we have that  $S_i$  is a subset of  $\Sigma^{k_i}$ , and

$$k_1 < k_2 < \dots < k_r < m .$$

2. The set  $S = S_1 \cup \dots \cup S_r$  is fix-free, but not complete.
3. If  $j \neq i + 1$  then  $S_j \Sigma^{m-k_j} \cap \Sigma^{m-k_i} S_i = \emptyset$ .
4. There is a sequence of strings  $x_1, x_2, \dots, x_r$  such that  $x_i \in S_i$  for  $1 \leq i \leq r$ , and  $x_i$  is a suffix of some string in  $x_{i+1} \Sigma^{m-k_{i+1}}$  for  $1 \leq i \leq r - 1$ .

*Remark.* Conditions (1) – (4) are the most general we know of to ensure the validity of the construction that follows. The special case to keep in mind while reading the construction is the one in which  $S_i = \{x_i\}$  for every  $i$ . We illustrate this special case in the next section.

We assume that  $S_1, \dots, S_r$  satisfy conditions (1) – (4), and using them we construct a complete fix-free code  $T_r$  as follows.

*Step 0.* Let  $T_0$  denote  $S \cup (\Sigma^m - S\Sigma^*)$ ; thus  $T_0$  is the union of  $S$  with the set of all codewords of length  $m$  that do not begin with a word from  $S$ .

*Step  $i$ ,* for  $i = 1, 2, \dots, r$ . Let  $T_i$  denote the set of strings in  $T_{i-1}$ , except that each string  $x$  of  $T_{i-1}$  having a proper suffix in some  $S_j$  is replaced by  $x\Sigma^{m-k_j}$ . (Each such word  $x$  is replaced by  $t^{m-k_j}$  words of length  $|x| + m - k_j$ .) Since  $S$  is suffix-free by condition (2) no string of  $T_{i-1}$  has more than one such suffix, so this step is well-defined.

It is easy to argue by induction that each  $T_i$  is both prefix-free and complete. (In fact it is helpful to think of  $T_i$  as a prefix tree, with certain branches being extended to create  $T_{i+1}$ .) Therefore, it only remains to argue that  $T_r$  is suffix-free.

**Claim 1** *Let  $0 \leq i \leq r$ . If  $x \in T_i$  is a proper suffix of some string in  $T_i$  then  $x \in S_1 \cup \dots \cup S_{r-i}$ . In particular,  $T_r$  is fix-free.*

*Proof.* It will suffice to show that each new string created in step  $i$  contains no proper suffix from among  $(T_0 \cup \dots \cup T_r) \setminus (S_1 \cup \dots \cup S_{r-i})$ . We prove this by induction on  $i$ . The case  $i = 0$  is obvious. Suppose a string  $x$  was created in Step  $i > 0$  by appending some string in  $\Sigma^{m-k_j}$ . By induction we have  $j \leq r - (i - 1)$ . By condition (3), if  $x$  has a proper suffix  $y \in S$ , then  $y$  must be an element of  $S_{j-1}$ , which is contained in  $S_1 \cup \dots \cup S_{r-i}$ . We must now show that  $x$  has no proper suffixes from among  $(T_0 \cup \dots \cup T_r) \setminus S$ .

Suppose for purposes of contradiction that  $y \in (T_0 \cup \dots \cup T_r) \setminus S$  and  $y$  is a proper suffix of  $x$ . The last  $m$  letters of  $y$  have an element of  $S_j$  as a prefix; therefore  $y$  was created at some step  $i' > 0$  by appending some string in  $\Sigma^{m-k_j}$ . Snip the last  $m - k_j$  letters off  $x$  and  $y$  to get  $x'$  and  $y'$ . Then  $x'$  was created at step  $i - 1$  and  $y'$  was created at step  $i' - 1$  and  $y'$  is a suffix of  $x'$ . Also,  $y' \notin S$ , which contradicts the inductive hypothesis. ■

**Claim 2** *The longest word in  $T_r$  has length  $m + \sum_{j=1}^r (m - k_j)$ .*

*Proof.* We show by induction that for  $0 \leq i \leq r-1$ , the string  $x_{r-i}$  given by condition (4) is a suffix of some string in  $T_i$  of length  $m + \sum_{j=1}^i (m - k_{r-(j-1)})$ . The case  $i = 0$  follows from condition (3). Suppose  $i > 0$ . By induction  $x_{r-(i-1)}$  is a suffix of some string in  $T_{i-1}$  of length  $m + \sum_{j=1}^{i-1} (m - k_{r-(j-1)})$ . The construction of  $T_i$  appends all strings of length  $m - k_{r-(i-1)}$  to this string, and by condition (4) one of the new strings so created has  $x_{r-i}$  as a suffix. This completes the induction.

We have shown that  $x_1$  is a suffix of some string of  $T_{r-1}$  of length  $m + \sum_{j=1}^{r-1} (m - k_{r-(j-1)})$ . Now the construction of  $T_r$  appends all strings of length  $m - k_1$ . This creates strings in  $T_r$  of length  $m + \sum_{j=1}^r (m - k_j)$ . By a similar argument and condition (3) every string created after step 0 has length  $m + \sum_{j=s}^{s'} (m - k_j)$  for some  $1 \leq s \leq s' \leq r$ . Therefore, the strings whose existence we just established are the longest in  $T_r$ . ■

The construction in Theorem 1 is a special case of the construction of  $T_r$  given here, with  $r = 1$ ,  $S = S_1$ ,  $k = k_1$ , and  $m = k + p$ . Hypothesis (1) in Theorem 1 implies that condition (3) holds.

### 3 Example

In this section we construct, for each whole number  $n$ , a binary fix-free code in which the longest word has length  $(5n^2 + 13n + 2)/2$  and the shortest word has length  $3n + 1$ . The ratio of longest word to shortest word is therefore greater than  $5n/6$ . It is possible to improve the constants slightly with a little effort; that is, to attain a larger ratio for the same maximum length string.

Let  $n \geq 1$  be an integer. Referring to the notation of the previous section, set  $m = 6n + 1$  and  $r = n$ . For  $1 \leq i \leq r$ , put  $k_i = 3n + i$  and  $x_i = \mathbf{0}^{2^{i-1}} \mathbf{1} \mathbf{0}^{3n-i-1} \mathbf{1}$ . The collection of singleton sets  $S_i = \{x_i\}$ ,  $i = 1, \dots, r$ , clearly satisfies conditions (1) – (4). The construction of the fix-free code proceeds as in the previous section.

### 4 Further Work

There are fix-free codes unaccounted for by the construction of this paper. (We have written a C program to search for such codes; this program and its output are available from the authors.) We do not know how common fix-free codes are among the complete prefix-free codes. It would be interesting to find an upper bound on the average codeword length of an optimal fix-free encoding of an arbitrary  $n$ -letter source alphabet; for example, some constant multiple of the entropy of the source alphabet.

#### Acknowledgments

We would like to thank Peter Elias and Mojdeh Mohtashemi for helpful discussions.

## References

- [1] David A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.