2010 Data Compression Conference

A SAT-Based Scheme to Determine Optimal Fix-free Codes

Navid Abedini, Sunil P. Khatri and Serap A. Savari
Texas A&M University, College Station
[novid_abed, sunilkhatri]@tamu.edu, savari@ece.tamu.edu

Abstract

Fix-free or reversible-variable-length codes are prefix condition codes which can also be decoded in the reverse direction. They have attracted attention from several communities and are used in video standards. Two variations of fix-free codes (with additional constraints) have also been considered for joint source-channel coding: 1) "symmetric" fix-free codes, which require the codewords to be palindromes; 2) fix-free codes with distance constraints on pairs of codewords. We propose a new approach to determine the existence of a fix-free code with a given set of codeword lengths, for each of the three variations of the problem. We also describe a branch-and-bound algorithm to find the collection of optimal codes for asymmetric and symmetric fix-free codes.

1. Introduction

Fix-free codes denote a class of variable-length codes which are both prefix-free and suffix-free, that is, no codeword of a fix-free code is a prefix or suffix of another codeword. One reason why fix-free codes are popular for data compression purposes is that a finite sequence of fix-free codewords can be decoded instantaneously in both directions. Fix-free codes were introduced five decades ago by Schutzenberger [1] as *anagrammatic codes* and by Gilbert and Moore [2] under the title of *never-self-synchronizing codes*. Later, they were studied in other papers ([3]-[5]) under a wide variety of names such as reversible-variable-length, affix codes, bifix codes, bidirectional Huffman codes and fix-free codes. The bidirectional decodability enhances the robustness to channel noise ([5],[6]), as well as the speed of decoding process. This is why fix-free codes are used within the video standards H.263+ and MPEG-4.

Two of the three variations of fix-free codes which we consider in this paper are asymmetric and symmetric fix-free codes. In the latter case, each codeword is required to be a palindrome, while this constraint is not present in the former case. Symmetric fix-free codes have been found [6] to be more suitable for joint source-channel coding than asymmetric fix-free codes. Many papers have explored the properties of and/or proposed algorithms for the construction of both these classes of fix-free codes. For example, [8] and [9] introduced algorithms to respectively construct asymmetric and symmetric fix-free codes. A set of necessary and sufficient conditions on the existence of fix-free codes can be found for the asymmetric case in [10] and for the symmetric case in [4], [7] and [11]. However, these conditions are not exhaustive and there are instances for either variation where both conditions fail. For such instances, we need a different approach to determine if a fix-free code with a particular length sequence exists. Moreover, even when the sufficient conditions establish the existence of a fix-free code with a given set of codeword lengths, a procedure is needed to generate such a code. For fix-free codes with a length sequence $(l_1, l_2, \ldots, l_n)$ whose *Kraft-sum* is 1 (i.e., $\sum_{i=1}^{n} 2^{-l_i} = 1$), [5] provides such an algorithm. The idea of formulating the existence and code construction problem as a Boolean satisfiability (SAT) problem (see [12]) was first proposed in [13]. In this approach, the constraints on codewords are translated into a Boolean function in the Conjunctive Normal Form (CNF). If this function evaluates to TRUE, the underlying problem is guaranteed to have a solution. The problem of finding an asymmetric (or symmetric) fix-free code for a given set of codeword lengths can be converted to a SAT problem. If the SAT problem is satisfiable (i.e., the CNF formula evaluates to TRUE for some assignments of variables), a fix-free code exists for the given codeword lengths. There are numerous software packages to solve the SAT problem (see, e.g, [14],[19]). If a SAT solver determines that the SAT problem is satisfiable, it also provides information from which we can generate a code with given codeword lengths for the fix-free problem. In Section 2 we provide some basics of SAT. We offer new ways to map symmetric and asymmetric fix-free problems to a SAT problem, which are more efficient than the previous work [13]. Section 3 presents our formulation to map the problem of determining the feasibility of both the symmetric and asymmetric fix-free codes (for a given length sequence) to a SAT problem.

Error control techniques have always been of great interest in computer science and communications, and many error detection and correction techniques have been proposed. Maintaining a minimum distance between equal length blocks of code symbols is a common method to provide error control capability. If an error occurs during the transmission of a codeword, another word will be received by the decoder. Therefore, if the assigned codewords are sufficiently distant from each other, the received word is not likely to be identical to any other valid codewords other than the one transmitted, and the occurrence of errors can hence be detected. In Section 2 we also present a precise notion of distance in variable-length error correcting codes which was first discussed in [18]. In Section 3 we also present the first approach to establishing the feasibility (using SAT) of a length sequence for a variation of the fix-free problem with distance requirements.

Consider a source with n symbols, indexed by $\{1,2,\ldots,n\}$, and assume $p_1 \geq p_2 \geq \cdots \geq p_n$, where p_i is the probability of the i^{th} symbol. Each symbol i is mapped into a codeword c_i which has length l_i. A minimum-redundancy or optimal code is one which minimizes the average codeword length (i.e., $\sum_{i=1}^{n} p_i l_i$) over a class of codes. The sequence of codeword lengths for an optimal code is called a dominant length sequence [11], [13]. A number of papers (e.g., [15]-[17]) study the dominant length sequences for a binary prefix condition code with n words, and Gilbert [16] showed that the number of such codes grows roughly as $0.148\,(1.791)^n$. In this paper we present a branch-and-bound algorithm to find dominant length sequences for asymmetric fix-free codes, and extend the results of [11] for the symmetric variation. Our new results show that the number of asymmetric and symmetric fix-free dominant length sequences appear to grow slowly with n compared to binary prefix condition codes. Therefore for moderate sizes of n it may be possible to store all dominant length sequences and easily obtain the best one of these for a particular application.

2. Preliminaries and terminology

In this section, we review the basics of Boolean satisfiability (SAT), and also discuss the different notions of distance used in the study of instantaneous, especially fix-free, codes.

Definition 1: A **literal** is defined as a Boolean variable or its complement.

Definition 2: A **conjunctive normal form (CNF)** Boolean formula f on n Boolean variables $v_1, v_2, \cdots, v_n$ is a conjunction (logical AND) of m clauses $c_1, c_2, \cdots, c_m$. Each clause is a disjunction (logical OR) of its constituent literals. For example, $f = (v_1 + v_2).(\overline{v_1} + \overline{v_2} + \overline{v_3})$ is a CNF formula with two clauses, $c_1 = (v_1 + v_2)$ and $c_2 = (\overline{v_1} + \overline{v_2} + \overline{v_3})$. **Boolean satisfiability (SAT)** is the problem of determining whether a Boolean formula expressed in conjunctive normal form (CNF) has a satisfying assignment (i.e., an assignment of values to some or all variables of f which makes the function f evaluate to TRUE). In the above example a satisfying assignment of variables for function f is $v_1 = 1, v_2 = 0, v_3 = 0$. Theoretically, the SAT process can have two outcomes 1) the problem is determined to be not satisfiable and 2) the problem is satisfiable. In the latter case, a SAT solver also returns a satisfying assignment of variables for the given CNF formula. In practice, however, if the SAT solver cannot determine the satisfiability of the formula within a specified amount of time, it will return *unknown*.

For error correcting purposes, it is common (see e.g. [18]) to consider a minimum required "distance" between codewords of a code C. For example, a minimum distance of 3 between two codewords of the same length provides the capability of correcting one bit error. Traditionally, distance requirements are expressed based on the notion of Hamming distance.

Definition 3: The **Hamming distance** $h(x^i, x^j)$ between two codewords x^i and x^j of the same length is defined as the number of positions in which x^i and x^j differ. The **minimum block distance** b_k for the codeword length L_k of a code C is the minimum Hamming distance between all distinct codewords of C with length L_k [18].

$$b_k = \min_{\substack{x^i, x^j \in C,\ i \neq j \\ |x^i| = |x^j| = L_k}} \{h(x^i, x^j)\} \tag{1}$$

The **overall minimum block distance** b_{min} of a code C is the minimum block distance for all possible lengths. In other words, $b_{min} = \min_{1 \leq k \leq m} b_k$, where L_m is the maximum codeword length in code C**.** The **diverging distance** $d(x^i, x^j)$ between two codewords with different lengths $|x^i|$ and $|x^j|$ is the Hamming distance between the l-length prefixes of codewords x^i and x^j with $l = min\{|x^i|, |x^j|\}$[18]. Similarly, the **converging distance** $c(x^i, x^j)$ is defined for l-length suffixes of x^i and x^j.

$$\begin{gathered} d(x^i, x^j) = h(x_1^i \cdots x_l^i, x_1^j \cdots x_l^j) \\ c(x^i, x^j) = h\left(x_{|x^i|-l+1}^i \cdots x_{|x^i|}^i, x_{|x^j|-l+1}^j \cdots x_{|x^j|}^j\right) \end{gathered} \tag{2}$$

The **minimum diverging distance** d_{min} (**minimum converging distance** c_{min}) of a code C is the minimum value of all diverging (converging) distances between every possible pair of codewords in C: $d_{min} = \min_{\substack{x^i,x^j \in C \\ |x^i| \neq |x^j|}} \{d(x^i, x^j)\}$ ($c_{min} = \min_{\substack{x^i,x^j \in C \\ |x^i| \neq |x^j|}} \{c(x^i, x^j)\}$).

3. Problem formulation

In this section, we introduce a new SAT-based method to determine whether or not there exists a fix-free code C with the given length sequence $(l_1, l_2, \cdots, l_n)$, where $n = |C|$, $l_1 \leq l_2 \leq \cdots \leq l_n$, and l_i is the length of i^{th} codeword in C. We consider three variations of fix-free codes: 1) asymmetric fix-free codes, 2) symmetric fix-free codes and 3) asymmetric fix-free codes with distance constraints.

We encode the constraints of the decision and assignment problem for a particular length sequence as an instance of a Boolean satisfiability problem. This SAT instance is fed as the input to a SAT solver (*MiniSat* [19] in our case). If the SAT solver returns a *satisfiable* result, it will also provide a satisfying assignment of the variables from which we can trivially construct the fix-free code. If the SAT solver returns an *unsatisfiable* result, we can assert that there exists no fix-free code with the given length sequence, satisfying the requirements of the corresponding variation. In the rest of this section we explain the construction of the SAT instance that we use for each variation of our problem.

3.1 Clauses for asymmetric fix-free codes

Consider a fix-free code C with codewords $x^1, x^2, \cdots, x^n$, where $n = |C|$. Each codeword x^i, $i \in \{1, \ldots, n\}$, has l_i bits and is expressed as $v_1^i v_2^i \cdots v_{l_i}^i$, where $v_j^i \in \{0,1\}$ is the j^{th} bit in codeword x^i. Suppose $l_1 \leq l_2 \leq \cdots \leq l_n$. In this variation, each pair of codewords x^i and x^j, $i < j$, (i.e. $l_i \leq l_j$) are required to satisfy

(i) The prefix condition: the first l_i bits of x^j differs from the codeword x^i.

(ii) The suffix condition: the last l_i bits of x^j differs from the codeword x^i.

These conditions are identical when $l_i = l_j$. A Boolean expression for the prefix condition is

$$\overline{\wedge_{k=1}^{l_i}\left(v_k^i = v_k^j\right)}. \tag{3}$$

Define an intermediate variable $z_{m,n}^{i,j} \equiv \left(v_m^i = v_n^j\right)$. Then (3) can be rewritten as

$$\overline{\wedge_{k=1}^{l_i}\left(z_{k,k}^{i,j}\right)}. \tag{4}$$

By one of De Morgan's laws, (4) is equivalent to

$$\vee_{k=1}^{l_i}\left(\overline{z_{k,k}^{i,j}}\right) = \left(\overline{z_{1,1}^{i,j}} + \overline{z_{2,2}^{i,j}} + \cdots + \overline{z_{l_i,l_i}^{i,j}}\right). \tag{5}$$

The SAT clauses to describe the statement $z_{k,k}^{i,j} \equiv \left(v_k^i = v_k^j\right)$ are

$$\left(\overline{z_{k,k}^{i,j}} + v_k^i + \overline{v_k^j}\right) \wedge \left(\overline{z_{k,k}^{i,j}} + \overline{v_k^i} + v_k^j\right) \wedge \left(z_{k,k}^{i,j} + \overline{v_k^i} + \overline{v_k^j}\right) \wedge \left(z_{k,k}^{i,j} + v_k^i + v_k^j\right). \tag{6}$$

Let $C_{k,k}^{i,j}$ denote the conjunction of these four clauses. Therefore to encode the prefix condition between two codewords x^i and x^j with $l_i \leq l_j$ as a SAT problem, we need to consider the conjunction of all clauses in (6) for $k = 1,2, \cdots. l_i$ and the single clause in (5). We denote these clauses by $S_P^{i,j}$, i.e.,

$$S_P^{i,j} = \left(\overline{z_{1,1}^{i,j}} + \overline{z_{2,2}^{i,j}} + \cdots + \overline{z_{l_i,l_i}^{i,j}}\right) \prod_{k=1}^{l_i} C_{k,k}^{i,j} \tag{7}$$

The suffix condition between codewords x^i and x^j with $l_i \leq l_j$ can be similarly encoded as a SAT problem. For the suffix condition, the last l_i bits of x^j is compared to the codeword x^i,

$$S_S^{i,j} = \left(\overline{z_{1,(l_j-l_i+1)}^{i,j}} + \overline{z_{2,(l_j-l_i+2)}^{i,j}} + \cdots + \overline{z_{l_i,l_j}^{i,j}}\right) \prod_{k=1}^{l_i} C_{k,(l_j-l_i+k)}^{i,j} \tag{8}$$

The SAT instance for finding a fix-free code C with the word lengths $l_1 \leq l_2 \leq \cdots \leq l_n$, where $n = |C|$ is obtained by a conjunction of the clauses for prefix and suffix conditions between each pair of codewords,

$$\mathcal{S}_{asym}(\mathrm{C}) = \prod_{\substack{i=1\ to\ n-1 \\ j=i+1\ to\ n}} \left(S_P^{i,j} . S_S^{i,j}\right) \tag{9}$$

where $S_P^{i,j}$ and $S_S^{i,j}$ are defined as in (7) and (8).

3.2 Clauses for symmetric fix-free codes

In addition to the clauses used by the asymmetric variation (expressed in (9)), checking the feasibility of a length sequence in the symmetric version requires additional clauses constraining each codeword to be a palindrome. However, we can utilize the properties of symmetric codes to decrease the number of variables and clauses in the SAT instance and consequently reduce the runtime of the SAT solver. In the following, we express a more efficient formulation of the fix-free problem when we are considering symmetric codewords. We begin with two simple observations.

In a symmetric code C each codeword $x^i = x_1^i x_2^i \cdots x_{m-1}^i x_m^i$ is a palindrome, and so $x_k^i = x_{m-k+1}^i$, $k = 1,2,\cdots,\lfloor m/2 \rfloor$. Therefore, we only need $\lceil |x^i|/2 \rceil$ variables to express the codeword x^i as opposed to $|x^i|$ variables for an asymmetric code. It is also known (see, e.g., [11]) that a palindrome σ^1 is a prefix of another palindrome σ^2 if and only if σ^1 is also a suffix of σ^2. Therefore the SAT based formulation of symmetric fix-free codes requires only an encoding of the prefix condition. The number of clauses can be reduced further when we consider codewords x^i, x^j with $|x^i| = |x^j| = l$. In this case, it is sufficient to compare the first $\lceil l/2 \rceil$ bits of the two palindromes. Based on these observations, we express the following efficient formulation of the symmetric fix-free problem.

The SAT instance for finding a symmetric fix-free code C with $n = |C|$ word lengths $l_1 \leq l_2 \leq \cdots \leq l_n$ can be written as $\mathcal{S}_{sym}(C) = \prod_{\substack{i=1\ to\ n-1 \\ j=i+1\ to\ n}} \left(S^{i,j}\right)$, where $S^{i,j}$ is given by

$$S^{i,j} = \left(\sum_{k=1}^{\left\lceil \frac{l_i}{2} \right\rceil} \overline{z_{k,k}^{i,j}} + \sum_{k=\left\lceil \frac{l_i}{2} \right\rceil+1}^{\min\left\{l_i, \left\lceil \frac{l_j}{2} \right\rceil\right\}} \overline{z_{l_i-k+1,k}^{i,j}} + \sum_{k=\min\left\{l_i, \left\lceil \frac{l_j}{2} \right\rceil\right\}+1}^{l_i} \overline{z_{l_i-k+1,l_j-k+1}^{i,j}} \right) \prod_{k=1}^{\left\lceil \frac{l_i}{2} \right\rceil} C_{k,k}^{i,j} \prod_{k=\left\lceil \frac{l_i}{2} \right\rceil+1}^{\min\left\{l_i, \left\lceil \frac{l_j}{2} \right\rceil\right\}} C_{l_i-k+1,k}^{i,j} \prod_{k=\min\left\{l_i, \left\lceil \frac{l_j}{2} \right\rceil\right\}+1}^{l_i} C_{l_i-k+1,l_j-k+1}^{i,j}$$

When $l_i = l_j$, the preceding expression reduces to $S^{i,j} = \left(\sum_{k=1}^{\lceil l_i/2 \rceil} \overline{z_{k,k}^{i,j}}\right) \prod_{k=1}^{\lceil l_i/2 \rceil} C_{k,k}^{i,j}$.

As an example of the efficiency of this formulation, the feasibility check of a symmetric fix-free code with $l_1 = \cdots = l_4 = 3$, $l_5 = l_6 = 4$, $l_7 = \cdots = l_{10} = 5$, $l_{11} = \cdots = l_{14} = 6$, $l_{15} = \cdots = l_{20} = 7$, $l_{21} = \cdots = l_{24} = 8$ and $l_{25} = l_{26} = 9$ requires 6273 clauses and 1684 variables while the approach of [13] uses 35526 SAT clauses and 156 variables.

3.3 Clauses for asymmetric fix-free codes with distance constraints

Imposing distance constraints between the codewords of a code introduces an error correcting capability to the code. In practice [18], we require $b_{min} = d_{min} + c_{min} = d$ and $d_{min} = \lceil d/2 \rceil$, where typical values for d are 3, 5 and 7. The distance constraints between each pair of codewords (x^i, x^j), $i < j$, are as follows: 1) When $l_i = l_j$, we require $h(x^i, x^j) \geq d$. 2) When $l_i < l_j$, we require $d(x^i, x^j) = h\left(x_1^i \cdots x_{l_i}^i, x_1^j \cdots x_{l_i}^j\right) \geq \lceil d/2 \rceil$ and $c(x^i, x^j) = h\left(x_1^i \cdots x_{l_i}^i, x_{l_j-l_i+1}^j \cdots x_{l_j}^j\right) \geq \lfloor d/2 \rfloor$. To summarize, we have requirements of the form $h\left(x_1^i \cdots x_{l_i}^i, x_r^j \cdots x_{r+l_i-1}^j\right) \geq d'$, where $r \in \{1, l_j - l_i + 1\}$ and $d' \in \{d, \lceil d/2 \rceil, \lfloor d/2 \rfloor\}$. For small values of d, like the ones considered in earlier literature, it is more efficient to alternatively consider $h\left(x_1^i \cdots x_{l_i}^i, x_r^j \cdots x_{r+l_i-1}^j\right) \neq k$ for $k = 0,1,\cdots,d'-1$. In the following we express the formulation of these distance constraints as a SAT problem.

Note that $h\left(x_1^i \cdots x_{l_i}^i, x_r^j \cdots x_{r+l_i-1}^j\right) = k$ means two sequences, $x_1^i \cdots x_{l_i}^i$ and $x_r^j \cdots x_{r+l_i-1}^j$, differ in k positions and are the same in the remaining $l_i - k$ positions. Let $z_{m,m+r-1}^{i,j} \equiv \left(v_m^i = v_{m+r-1}^j\right)$ indicate if two sequences agree in the m^{th} position; here $v_m^i = x_m^i$, the m^{th} bit of codeword x^i. Therefore the constraint $h\left(x_1^i \cdots x_{l_i}^i, x_r^j \cdots x_{r+l_i-1}^j\right) = k$ is the same as having k values of $z_{m,m+r-1}^{i,j}$'s, for $m \in \{1,\cdots,l_i\}$, be equal to 0 and the remaining values be equal to 1. For example, $\overline{z_{1,r}^{i,j}}\,\overline{z_{2,r+1}^{i,j}} \cdots \overline{z_{k,k+r}^{i,j}}\, z_{k+1,k+r+1}^{i,j} z_{k+2,k+r+2}^{i,j} \cdots z_{l_i,r+l_i-1}^{i,j}$ says that two sequences, $x_1^i \cdots x_{l_i}^i$ and $x_r^j \cdots x_{r+l_i-1}^j$, differ in the first k position and are the same in the remaining $l_i - k$ positions, resulting in the Hamming distance of the two sequences being equal to k. Since any k out of l_i positions can be selected there are a total number of $\binom{l_i}{k} = \frac{l_i!}{k!\,(l_i-k)!}$ Boolean expressions like the previous one. Hence, $h\left(x_1^i \cdots x_{l_i}^i, x_r^j \cdots x_{r+l_i-1}^j\right) = k$ can be expressed as the disjunction of these $\binom{l_i}{k}$ statements. By one of De Morgan's laws, $h\left(x_1^i \cdots x_{l_i}^i, x_r^j \cdots x_{r+l_i-1}^j\right) \neq k$ is equivalent to the conjunction of the complements of the $\binom{l_i}{k}$ conditions mentioned above. Let $S_{\neq k}(i,j,r)$ denote the Boolean formula in conjunctive normal form for the preceding statements. Therefore $h\left(x_1^i \cdots x_{l_i}^i, x_r^j \cdots x_{r+l_i-1}^j\right) \geq d'$ can be expressed as a SAT problem as $S_{\geq d'}(i,j,r) = \prod_{k=0}^{d'-1} S_{\neq k}(i,j,r) \prod_{l=1}^{l_i} C_{l,l+r-1}^{i,j}$, where the $C_{l,l+r-1}^{i,j}$ clauses, defined as in (6), describe the statement $z_{l,l+r-1}^{i,j} \equiv \left(v_l^i = v_{l+r-1}^j\right)$. To determine the feasibility of the length sequence $(l_1, l_2, \cdots, l_{|C|})$ for the distance-constrained variation of the problem, we consider the conjunction of all Boolean expressions $S_{\geq d'}(i,j,r)$ for each pair of codewords x^i and x^j $(i < j)$, $\mathcal{S}_{\geq d}(\mathrm{C}) = \prod_{\substack{\mathrm{i,j,}d',\mathrm{r} \\ \mathrm{j}>i}} \left(\mathrm{S}_{\geq d'}(\mathrm{i,j,r})\right)$, where d' and r for each $i,j \in \{1, \cdots, |C|\}$ $(i < j)$ are given by

$$\begin{cases} d' = d\,, r = 1 & \text{if } (l_i = l_j) \\ (\,d' = \lceil d/2 \rceil, r = 1)\ and\ \left(\,d' = \lfloor d/2 \rfloor, r = l_j - l_i + 1\right) & \text{if } (l_i < l_j) \end{cases}$$

4. Dominant length sequences

The concept of dominant length sequences has recently appeared in [13] and is related to majorization [20]. Following [13], we say that the length sequence $(l_1, l_2, \cdots, l_n)$ *dominates* another length sequence $(l_1', l_2', \cdots, l_n')$ whenever $\sum_{j=1}^{i} l_j' \geq \sum_{j=1}^{i} l_j$ for each $i \in \{1, \cdots, n\}$. Under the preceding condition, if $(l_1, l_2, \cdots, l_n)$ is the length sequence of a fix-free code, then $(l_1', l_2', \cdots, l_n')$ cannot be a sequence of codeword lengths corresponding to a fix-free code with lower redundancy. The length sequence $(l_1, l_2, \cdots, l_n)$ of a fix-free code is called a *dominant* sequence if it is not dominated by the length sequence of another fix-free code.

Our new results on dominant length sequences for asymmetric and symmetric binary fix-free codes are presented in this section, and are based on a *branch-and-bound* algorithm (see, e.g., [22]). This algorithm utilizes the SAT based checks described in section 3. To implement the algorithm, we use a tree organization in which the leaves, which are nodes at level n, correspond to the length sequences of codewords for a source with n symbols and each node at level $k < n$ is the sequence of first k elements of its descendents. We start from the root which is an empty length sequence and traverse the tree in a depth-first manner. Each node m (or partial length sequence) is tested against certain criteria. If at least one of these criteria is met, the search rooted at node m will be bounded since the subtree rooted at this node cannot contain a dominant length sequence. If not, we branch further (i.e., we continue searching through the subtree rooted at this node). During the search, we store a database D of feasible and infeasible partial length sequences, and this information is used while running the algorithm for larger values of n. The pseudo code of the algorithm for the asymmetric binary fix-free case is presented in Table 1.

```
FIND_DOM_ASYMMETRIC(n)
(i)     $P \leftarrow [\,]$                        //initializing $P$ as an empty array
(ii)    Answer ← { }
(iii)   BRANCH_ASYMMETRIC(P,n)         //branching
(iv) RETURN(Answer)

BRANCH_ASYMMETRIC(P,n)
1   if (Kraft_Sum(P) < 5/8){
2       $l^* \leftarrow -log_2\left(\left(\frac{5}{8} - Kraft\_Sum(P)\right)/\left(n - length(P)\right)\right)$
3   } else{ $l^* \leftarrow 2n - 3$ }
4   for i ←P[length(P)] to $l^*${              // branching the node P
5       $P'[j] \leftarrow P[j]\ \ \forall j = 1, \dots, length(P)$ ,  $P'[length(P) + 1] \leftarrow i$              // $P'$ is a child of P
6       Bound ← FALSE
7       $P^*[j] \leftarrow P'[j]\ \ \forall j = 1, \dots, length(P')$ ,  $P^*[length(P') + j] \leftarrow i\ \ \forall j = 1, \dots, n - length(P')$
8       if ($P^*$ is dominated by any length sequence in Answer){
9           Bound←TRUE                       //$P^*$, the best leaf rooted from $P'$, is dominated
10      } else if (Kraft_sum($P'$) $\leq 1$ ){
11          f1 ← CHECK_ASYMMETRIC_DATABASE($P'$)
12          if ( f1 = unknown){
13              f2 ← CHECK_ASYMMETRIC_FEASIBILITY($P'$)    //checking if $P'$ is feasible using SAT
                                                             //based test if necessary
14              $D \leftarrow D \cup (P', f2)$            //Updating the feasibility database with $P'$result
15          }
16          if (length($P'$) = n){              // $P'$ is a leaf
17              if ( f1 = True OR f2 = True){
18                  Answer ← {$Answer, P'$}
19                  Bound←TRUE
20              }
21          } else{                              // $P'$ is not a leaf so it possibly can be branched
22              if ( f1 = True OR f2 = True){
23                  BRANCH_ASYMMETRIC($P', n$)
24              }
25          }
26      }
27      if (Bound = True){
28          EXIT                          // search is bounded
29      }
30  }
```

Table 1. Branch-and-bound algorithm to find dominant sequences of asymmetric fix-free codes

FIND_DOM_ASYMMETRIC(n) returns dominant length sequences corresponding to asymmetric binary fix-free codes with n codewords. Suppose $\boldsymbol{P} = (l_1, \dots, l_k)$ is a partial length sequence, where $k = length(\boldsymbol{P})$. ***BRANCH_ASYMMETRIC(P,n)*** checks the subtree rooted at node ***P***, by recursively calling itself for those children which can be branched upon, to find the dominant sequences with partial sequence $\boldsymbol{P}$. The next codeword length l_{k+1} satisfies $l_k \leq l_{k+1} \leq l^*$, where l^* is determined (lines 1-3) based on following observations,

(i) An optimal asymmetric fix-free code has Kraft-Sum not less than 0.625 [21].

(ii) A fix-free code for n codewords can be obtained from the Huffman code with n codewords by concatenating each Huffman codeword with itself in reverse and removing an intermediate bit. The maximum codeword length for this code is $2n - 3$.

In each iteration of FOR loop (lines 4-30), the descendants of node $\boldsymbol{P}$ are examined in a depth-first order, using the bounding criteria. Each check is performed only if all of the preceding checks fail. The bounding checks are as follows;
Line 8: If $\boldsymbol{P}^*$, the best leaf rooted from $\boldsymbol{P}'$ (a child of $\boldsymbol{P}$ which we are checking at this iteration), is dominated by some previously found dominant sequences, then none of the remaining leaves rooted from $\boldsymbol{P}$ can be dominant and hence $\boldsymbol{P}$ should be bounded.
Line 10: If the Kraft-sum of $\boldsymbol{P}'$ is greater than 1, $\boldsymbol{P}'$ is infeasible and should be bounded.
Line 11: ***CHECK_ASYMMETRIC_DATABASE(P')*** searches for $\boldsymbol{P}'$ in the maintained database and returns *TRUE* or *FALSE* if it is found to be respectively feasible or infeasible. If there is no information about this sequence in the database, it will return *UNKNOWN*.
Line 13: ***CHECK_ASYMMETRIC_FEASIBILITY***($\boldsymbol{P}'$) checks the feasibility of $\boldsymbol{P}'$ by first testing the necessary and sufficient conditions (Theorems 1 and 2 of [10]). If they fail, we perform the SAT-based test. There are two cases for $\boldsymbol{P}'$ to be considered,
(i) *Lines 16-21*: $\boldsymbol{P}'$ is a leaf. If it is feasible, it will be added to the answer and node $\boldsymbol{P}$ will be bounded since other leaves with partial sequence $\boldsymbol{P}$ are dominated by $\boldsymbol{P}'$. Otherwise, the next child of $\boldsymbol{P}$ will be examined.
(ii) *Lines 22-24*: $\boldsymbol{P}'$ is an internal node. If it is feasible, it will be branched by calling ***BRANCH_ASYMMETRIC***($\boldsymbol{P}', \boldsymbol{n}$). Otherwise, the next child of $\boldsymbol{P}$ will be examined.

The algorithm for the symmetric version is similar to the presented algorithm for the asymmetric case after substituting the symmetric version of functions instead of asymmetric ones. Note also that the 5/8 lower bound on the Kraft-sum is true for the asymmetric case only, therefore for symmetric problem we always consider $l^* = 2n - 3$. It should also be noted that in ***CHECK_SYMMETRIC_FEASIBILITY***($\boldsymbol{P}'$), we use the necessary and sufficient conditions presented in [11] for symmetric binary fix-free codes

5. Results

The results of running the algorithm for both the asymmetric and symmetric variations are presented respectively in Tables 2 and 3.

N	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Number	1	1	1	2	2	4	7	8	10	14	20	29	38	55	66	79	99	124

Table 2. Number of dominant sequences for asymmetric binary fix-free codes

N	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Number	1	1	1	1	2	2	3	3	4	4	6	6	8	11	11	13	13	17

19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
18	21	22	24	26	29	32	34	36	42	43	46	48	49	53

Table 3. Number of dominant sequences for symmetric binary fix-free codes

It is seen that the number of dominant sequences for symmetric fix-free codes grows very slowly. The growth of this number for the asymmetric case is faster, however, there are much fewer asymmetric fix-free dominant sequences compared to the Huffman codes. For example, there exist 5269 dominant sequences for Huffman codes with $n = 18$ codewords while this number is just 124 for the asymmetric fix-free codes. Based on our results, we propose the following conjecture.

Conjecture: For each dominant length sequence $(l_1, \dots, l_n)$ corresponding to an asymmetric or a symmetric fix-free code with n codewords, we have $l_i \leq n$ for $i = 1,2, \dots, n$ and $\sum_{i=1}^{n} l_i \leq \sum_{i=1}^{n} i = n(n+1)/2$.

The sequence $(1,2, \dots, n)$ corresponding to the symmetric code $\{0,11,101,1001, \dots\}$ is always a dominant sequence in both the symmetric and asymmetric cases. In our experiments, this length sequence plays a similar role to the dominant length sequence $(1,2, \dots, n-1, n-1)$ of a binary prefix condition code: it has the maximum sum-length and the maximum codeword length among all dominant sequences.

References

[1] M. P. Schutzenberger, "On an application of semi-group methods to some problems in coding," *IRE Trans. Inform. Theory*, vol. IT-2, pp. 47–60, Sept. 1956.

[2] E. N. Gilbert and E. F. Moore, "Variable-length binary encodings," *Bell Sys. Tech. J.*, vol. 38, pp. 933-967, 1959.

[3] J. Berstel and D. Perrin, *Theory of Codes*, Orlando: Academic Press, 1985.

[4] Y. Takishima, M. Wada, and H. Murakami,"Reversible variable length codes," *IEEE Trans. Commun.*, vol. 43, pp. 158–162, Feb.-Apr. 1995.

[5] A. S. Fraenkel and S. T. Klein, "Bidirectional Huffman coding," *Computer J.*, vol. 33, pp. 296-307, 1990.

[6] R. Bauer and J. Hagenauer, "Iterative source/channel-decoding using reversible variable length codes," *Proc. DCC*, pp. 93–102, Snowbird Utah, March 2000.

[7] C. W. Tsai and J. L. Wu, "Modified symmetrical reversible variable-length code and its theoretical bounds," *IEEE Trans. Inform. Theory*, vol. 47, pp. 2543-2548, Sept. 2001.

[8] H.-W. Tseng and C.-C. Chang, "A fast and simple algorithm for the construction of asymmetrical reversible variable length codes," *Real Time Imagine,* vol. 9, pp. 3-10, 2003.

[9] H.-W. Tseng and C.-C. Chang, "Construction of symmetrical reversible variable length codes using backtracking," *Computer J.*, vol. 46, no. 1, pp. 100-105, 2003.

[10] C. Ye and R. W. Yeung, "Some basic properties of fix-free codes," *IEEE Trans. Inform. Theory*, vol. 47, pp. 72-87, 2001.

[11] S. A. Savari, "On optimal reversible-variable-length codes," *ITA* 2009.

[12] M. Davis, G. Logemann, and D. Loveland, "A machine program for theorem proving," *Communications of the ACM 5*, vol. 7, pp. 394–397, 1962.

[13] S. A. Savari, "On minimum-redundancy fix-free codes," *Proc. DCC*, Snowbird, UT, March 2009.

[14] J. P. Marques Silva and K. A. Sakallah, "GRASP - a new search algorithm for satisfiability," *ICCAD,* pp 220-227, 1996.

[15] E. Norwood, "The number of different possible compact codes," *IEEE Trans. Inform. Theory*, vol. IT-13, pp. 613-616, 1967.

[16] E. N. Gilbert, "Codes based on inaccurate source probabilities," *IEEE Trans. Inform. Theory*, vol. IT-17, pp. 304-314, 1971.

[17] S. Even and A. Lempel, "Generation and enumeration of all solutions of the characteristic sum condition," *Inform. and Cont.*, vol. 21, pp. 476-482, 1972.

[18] V. Buttigieg and P. G. Farrell, "Constructions for variable-length error-correcting codes," *in Proceedings of Cryptography and Coding, 5th IMA Conference*, Cirencester, UK, pp. 282-291, Dec. 95.

[19] N. Een and N. Sorensson, "MiniSat: A SAT solver with conflict clause minimization," *SAT 2005*.

[20] G. H. Hardy, J. E. Littlewood, and G. Polya, *Inequalities*, UK : Cambridge Univ. Press, 1952.

[21] S. Yekhanin, "Improved upper bound for the redundancy of fix-free codes," *IEEE Trans. Inform. Theory*, vol. 50, no. 11, pp. 2815-2818, Nov. 2004.

[22] A. H. Land and A. G. Doig, "An automatic method for solving discrete programming problems," *Econometrica*, vol. 28, pp. 497–520, 1960.