Characterizations of Minimal Expected Length Codes [*]

Spencer Congero and Kenneth Zeger

IEEE Transactions on Information Theory
Submitted: November 13, 2023

Abstract

A property of prefix codes called strong monotonicity is introduced. Then it is proven that for a prefix code C for a given probability distribution, the following are equivalent: (i) C is expected length minimal; (ii) C is length equivalent to a Huffman code; and (iii) C is complete and strongly monotone. Also, three relations are introduced between prefix code trees called same-parent, same-row, and same-probability swap equivalence, and it is shown that for a given source, all Huffman codes are same-parent, same-probability swap equivalent, and all expected length minimal prefix codes are same-row, same-probability swap equivalent.

[*] **S. Congero and K. Zeger** are with the Department of Electrical and Computer Engineering, University of California, San Diego, La Jolla, CA 92093-0407 (scongero@ucsd.edu and ken@zeger.us).

1. Introduction

Huffman codes [13] were invented in 1952 and today are widely used in many practical data compression applications, such as for text, audio, image, and video coding. They are known to be optimal in the sense that they achieve the minimal possible expected codeword length among all prefix codes for a given finite discrete random source [5].

The main idea in the Huffman algorithm is to construct a code tree from a source by recursively merging two smallest-probability nodes until only one node with probability 1 remains. The initial source probabilities correspond to leaf nodes in the tree, and the binary paths from the tree's root to the leaves are the codewords.

However, for a given source, Huffman codes are not unique, due to choices that arise during the tree construction that can be decided arbitrarily. Specifically, there are three types of choices that can be taken during the tree building: (1) When two nodes are merged, the choice of which node becomes a left child and which becomes a right child is arbitrary; (2) If there are three or more smallest-probability nodes, then which two of them to merge is arbitrary; and (3) If there is a unique smallest-probability node and two or more second-smallest-probability nodes, then which of these to merge with the smallest-probability node is arbitrary. After the tree is constructed all edges from parents to left children are labeled 0 and all edges from parents to right children are labeled 1, or vice versa. Without loss of generality, we will not interpret this binary edge labeling as a choice for generating multiple Huffman codes, since the same-parent node swaps that we address later in the paper can account for such constructions, too. Such arbitrary choices as in (1)–(3) made during the Huffman construction process can affect not only the codeword assignments, but also the Huffman tree structure, and can even change the distribution of codeword lengths.

On one hand, if a source is chosen randomly from a continuous distribution (i.e., the source probabilities are randomly chosen to lie in $[0, 1]$ and to sum to 1), then with probability one there will be no ties among source probabilities and no ties among tree node probabilities in the Huffman construction process. In this case, the only variation of Huffman codes for a given randomly chosen source is due to left-versus-right child assignments when node merges occur during Huffman tree construction.

On the other hand, often source distributions are empirically determined through a frequency counting process, and probability estimates consist of a set of integers, normalized by their sum. In these cases, especially with small data sets, ties in probabilities can occur, and multiple Huffman trees can result. The differences in these Huffman trees can be due to some or all of the arbitrary choices mentioned above that are encountered during Huffman tree construction.

For many applications, the average length of a prefix code is the primary concern, in which case the choice of which Huffman or other optimal non-Huffman code to use may not matter, although an understanding of such code variations may be of theoretical interest. In some applications, however, the specific binary codewords included in an optimal code may be critical. A survey of lossless coding techniques can be found in [1].

One well-studied example where the codeword assignments matter is the design of lossless codes that are easily synchronizable. Since Huffman codes are variable-length codes, they are subject to loss of synchronization during decoding due to even a single bit error or erasure during transmission or storage. However, Huffman codes are known to often have a self-synchronizing string, which is a binary string (not necessarily a codeword) that, after being decoded starting at any internal tree node, always returns the decoding process to the root of the Huffman tree, thus restoring synchronization. It turns out that for a given source with multiple Huffman codes, some Huffman codes may have shorter self-synchronizing strings than others, and some Huffman codes may not have any self-synchronizing strings at all even if other Huffman codes do. These possibilities are illustrated in the following two examples.

Example 1.1 (Same-parent node swap produces shorter self-synchronizing string)**.**
Let H_1 and H_2 be Huffman codes for a source with symbols a,b,c,d, and probabilities $\frac{1}{2}$, $\frac{1}{4}$, $\frac{1}{8}$, $\frac{1}{8}$, respectively. The Huffman trees for H_1 and H_2 are shown in Figure 1. Huffman tree H_2 is obtained from H_1 by exchanging node b and its sibling (i.e., a same-parent node swap). The shortest self-synchronizing strings for Huffman trees H_1 and H_2 are 0 and 00, respectively.

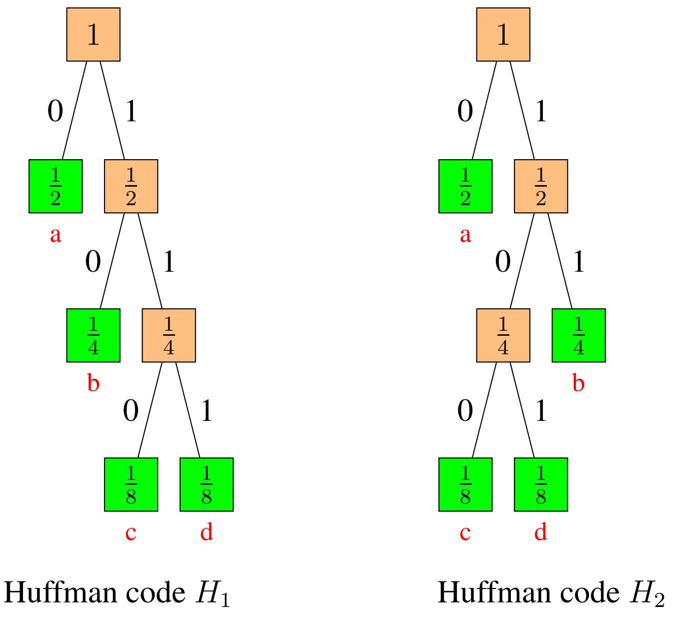

Figure 1: One Huffman tree is a same-parent node swap of another and has a shorter self-synchronizing string.

Example 1.2 (Same-parent node swap eliminates self-synchronizing string)**.**
Let H_1 and H_2 be Huffman codes for a source with symbols a,b,c,d,e,f,g,h,i, and probabilities $\frac{1}{4}$, $\frac{1}{8}$, $\frac{1}{8}$, $\frac{1}{8}$, $\frac{1}{8}$, $\frac{1}{16}$, $\frac{1}{16}$, $\frac{1}{16}$, $\frac{1}{16}$, respectively. The Huffman trees for H_1 and H_2 are shown in Figure 2. Huffman tree H_2 is obtained from H_1 by exchanging the leaf b and its sibling (i.e., a same-parent node swap). Huffman tree H_1 has a self-synchronizing string of 0011, which brings each internal node back to the root. Huffman tree H_2 does not have any self-synchronizing string, since any string which brings the root back to itself also brings the parents of a, b, and c back to one of themselves, and thus not to the root.

Numerous theoretical and algorithmic studies of synchronizable Huffman and non-Huffman optimal prefix codes have made use of the non-uniqueness of Huffman codes to search for short synchronizing binary strings. Some of these investigations include Longo and Galasso [16] in 1982, Ferguson and Rabinowitz [8] in 1984, Escott and Perkins [7] in 1998, Huang and Wu [12]

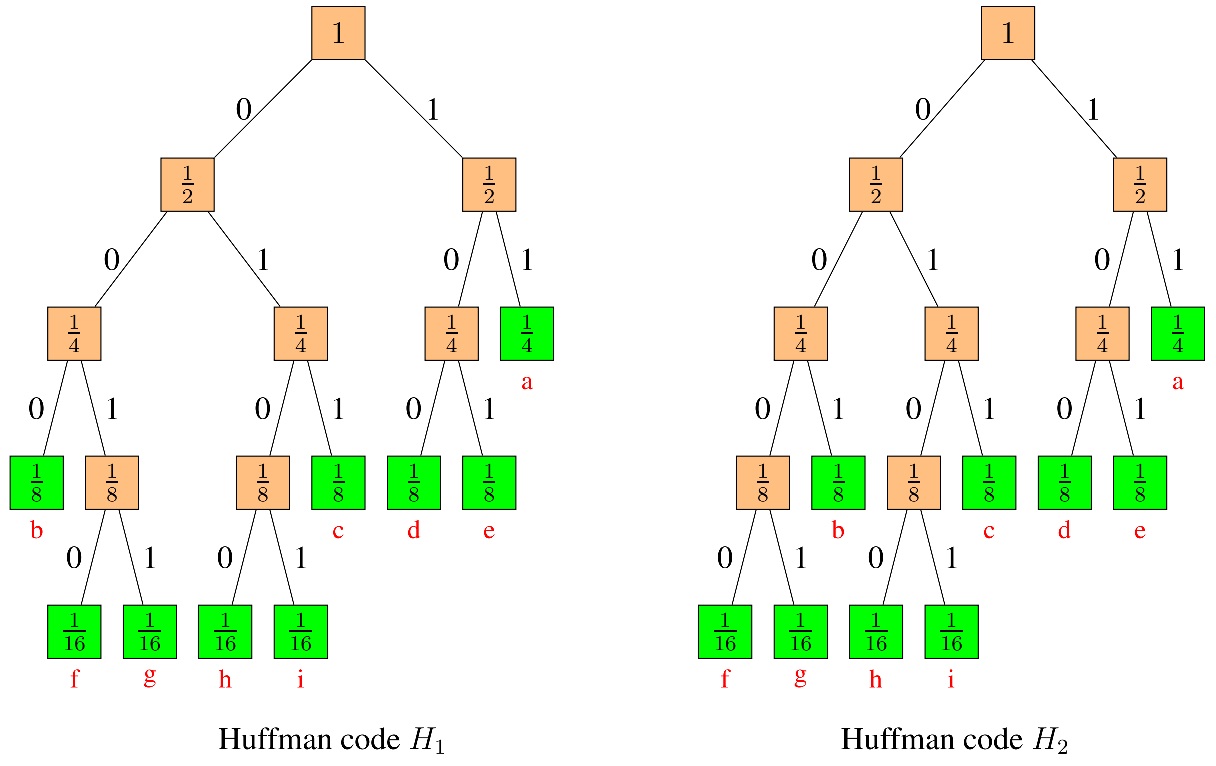

Figure 2: Huffman tree H_2 is a same-parent node swap of H_1, but has no self-synchronizing string whereas H_1 does.

in 2003, and Higgs, Perkins, and Smith [11] (see also [10]) in 2009. Other related work considers correcting bit errors in Huffman encoded data streams, in which case the choice of Huffman code can affect performance. Some of these include: Lee, Chang, Ho, and Lee [14] in 2000, Zhou and Zhang [20] in 2002, Cao [2] in 2006, Cao, Yao, and Chen [3] in 2007, Zhou and Au [19] in 2010, and Wang, Zhao, and Sun [17] in 2018.

The non-uniqueness of Huffman codes leads to the question of how to effectively describe the similarities among them, as well as their differences from non-Huffman codes. In 1978, Gallager [9] gave a characterization of Huffman codes as precisely those prefix codes possessing a "sibling property", which stipulates that a code is complete and the nodes of its code tree can be listed in order of non-increasing probability with each node being adjacent in the list to its sibling.

For a given source, the broader class of optimal prefix codes is somewhat larger then its subset of Huffman codes. No characterization analogous to the sibling property has been previously given for optimal prefix codes. Only the sufficient condition given by the sibling property has been known.

One known necessary condition for a prefix code to be optimal is "monotonicity", which states that any code tree node with a larger probability than another node must not appear on a lower row than the smaller-probability node. The following example illustrates that monotonicity is not sufficient for a complete prefix code to be optimal.

Example 1.3 (A monotone prefix code that is not optimal)**.**
Let H be a Huffman code for a source with symbols $a,b,c,d,$ and probabilities $\frac{3}{8}, \frac{3}{8}, \frac{1}{8}, \frac{1}{8},$ respec-

tively, and let C be another prefix code for the same source. The code trees for H and C are shown in Figure 3. The code C is monotone because any node probability on a given row is at least as large as any node probability on a lower row. However, C is not optimal since its expected length is 2, whereas the Huffman code H has a smaller expected length of $\frac{15}{8}$.

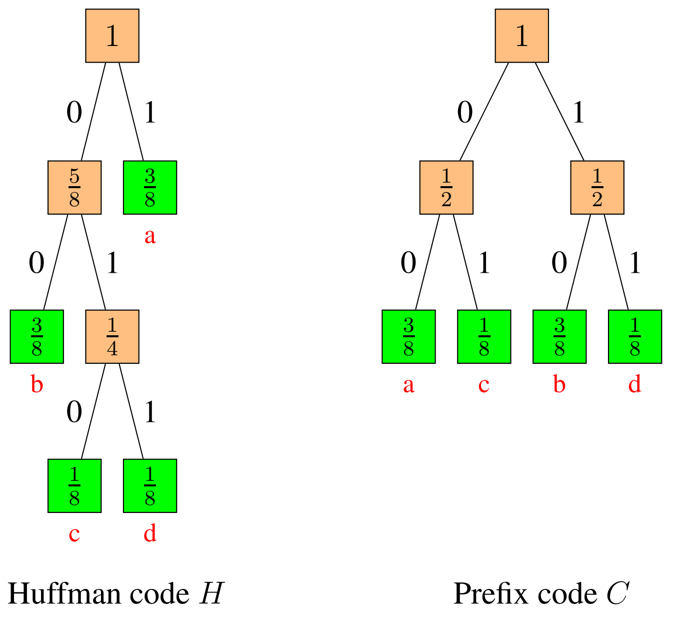

Figure 3: A Huffman tree and code tree illustrating monotonicity without strong monotonicity.

In this paper we first provide a necessary and sufficient characterization of optimal prefix codes by introducing a new criterion called "strong monotonicity". In particular we show that for a prefix code C for a given source, the following are equivalent: (i) C is optimal; (ii) C is length equivalent to a Huffman code; and (iii) C is complete and strongly monotone.

Secondly, we investigate the transformation of code trees to other code trees using the graph theoretic idea of swapping tree nodes. Swapping two nodes with a similar trait can transform one code tree into another code tree that maintains certain properties, and can serve as a method for creating new Huffman codes or new optimal prefix codes from existing ones.

Specifically, we consider swaps of two code tree nodes when they: (i) have the same parent; (ii) lie in the same row; or (iii) have the same probability. For any given source, these three types of node swaps always preserve the expected length of a prefix code. For example, any optimal prefix code will be transformed by any of these operations into another optimal prefix code.

Some prior work related to node swapping has motivated some of our results.

In 1982, Longo and Galasso [16] used same-probability node swaps in the context of determining probability density attraction regions for Huffman codes. They showed that, using only same-probability node swaps, Huffman codes could be transformed into other Huffman codes. However, they ignored the distinguishing effects of same-row node swaps that could transform Huffman codes to optimal non-Huffman codes. Our results more finely characterize the relationship between these codes by using either same-parent or same-row node swaps. Also we use a different proof technique and try to clarify some unaddressed points in [16].

Ferguson and Rabinowitz [8] studied synchronous codes and considered codes that were same-parent swap equivalent (calling them "strongly-equivalent") and length equivalent (calling them "weakly equivalent"), but did not provide results characterizing the class of Huffman or optimal prefix codes.

We characterize both the class of all Huffman codes for a given source and also the broader class of all optimal prefix codes for a given source in terms of same-probability and either same-row or same-parent node swap transformations.

Example 1.4 (Two Huffman codes and a third optimal code tree).
Let H_1 and H_2 be two different Huffman codes and let C be a non-Huffman tree for a source with symbols a,b,c,d, and probabilities $\frac{1}{3}, \frac{1}{3}, \frac{1}{6}, \frac{1}{6}$, respectively. The three trees are shown in Figure 4. After nodes c and d were combined during the Huffman construction algorithm for building H_1 and H_2, 3 different nodes had probability $\frac{1}{3}$, leading to different trees resulting from different node merges. All 3 codes achieve the minimum possible average length of 2, but the codes for H_1 and H_2 are not same-row swap equivalent, since the codes are not length equivalent.

However, H_1 and H_2 are same-parent, same-probability swap equivalent. To see this, first transform tree H_1 by exchanging node a with the parent of nodes c and d (i.e., a same-probability node swap). Then perform a same-parent node swap on nodes a and b, and then another same-parent node swap on the two children of the root. The result of these three operations is the Huffman tree H_2.

The code C is not same-parent, same-probability swap equivalent to either H_1 or H_2, but it is same-row, same-probability swap equivalent to both Huffman codes. To see this, transform H_2 to C by exchanging nodes b and c (i.e., a same-row node swap).

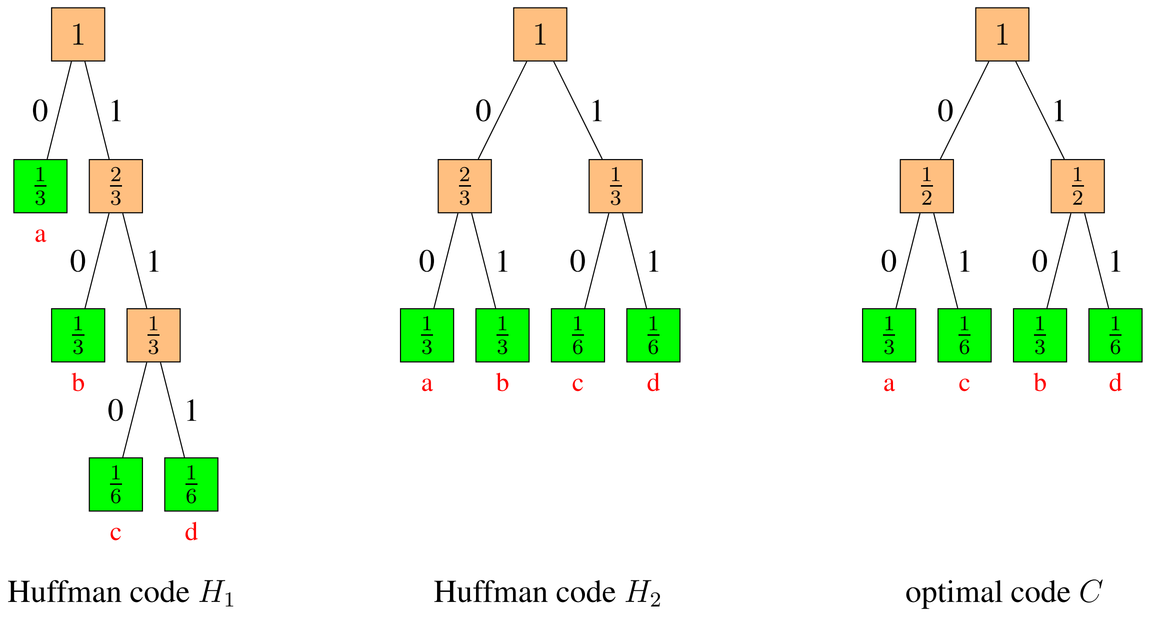

Figure 4: Two Huffman trees and an optimal third code tree for a single source.

We show that in fact for a given source, any Huffman code can be transformed into any other Huffman code by a sequence of same-parent and same-probability node swaps. Then we show that for a given source, any optimal code can be transformed into any other optimal code by a sequence of same-row and same-probability node swaps. These characterizations of Huffman codes and optimal codes refine a result in [16] and supplement the foundational understanding of optimal lossless source coding.

In what follows we will define terminology and then present our main results. One of these results (Theorems 2.4) is exploited in another recent work [4] to prove results about competitive optimality of Huffman codes.

An *alphabet* is a finite set S, and a *source* of size n with alphabet S is a random variable X such that $|S| = n$ and $P(X = y) = P(y)$ for all $y \in S$. We denote the probability of any subset $B \subseteq S$ by $P(B) = \sum_{y \in B} P(y)$.

A *code* for source X is a mapping $C : S \longrightarrow \{0,1\}^*$ and the binary strings $C(1), \ldots, C(n)$ are called *codewords* of C. A *prefix code* is a code where no codeword is a prefix of any other codeword.

A *code tree* for a prefix code C is a rooted binary tree whose leaves correspond to the codewords of C; specifically, the codeword associated with each leaf is the binary word denoting the path from the root to the leaf. The *length* of a code tree node is its path length from the root. The rth *row* of a code tree is the set of nodes whose length is r, and we will view a code tree's root as being on the top of the tree with the tree growing downward. That is, row r of a code tree is "higher" in the tree than row $r + 1$. If x and y are nodes in a code tree, then x is a *descendant* of y if there is a downward path of length zero or more from y to x. Two nodes in a tree are called *siblings* if they have the same parent. For any collection A of nodes in a code tree, let $P(A)$ denote the probability of the set of all leaf descendants of A in the tree.

A (binary) *Huffman tree* is a code tree constructed from a source by recursively merging two smallest-probability nodes[1] until only one node with probability 1 remains. The initial source probabilities correspond to leaf nodes in the tree. A *Huffman code* for a given source is a mapping of source symbols to binary words by assigning the source symbol corresponding to each leaf in the Huffman tree to the binary word describing the path from the root to that leaf.

Given a source with alphabet S and a prefix code C, for each $y \in S$ the length of the binary codeword $C(y)$ is denoted $l_C(y)$. Two codes C_1 and C_2 are *length equivalent* if $l_{C_1}(y) = l_{C_2}(y)$ for every source symbol $y \in S$. The *average length* of a code C for a source with alphabet S is $\sum_{y \in S} l_C(y)P(y)$. A prefix code is *optimal* for a given source if no other prefix code achieves a smaller average codeword length for the source. In particular, Huffman codes are known to be optimal (e.g., see [5]).

A code is *complete* if every non-root node in its code tree has a sibling, or, equivalently, if every node has either zero or two children.[2] A code C for a given source is *monotone* if for any two nodes in the code tree of C, we have $P(u) \geq P(v)$ whenever $l_C(u) < l_C(v)$. Optimal prefix codes are always monotone (Lemma 1.6) and are also well-known to be complete.

The *Kraft sum* of a sequence of nonnegative integers $l_1, \ldots, l_k$ is $2^{-l_1} + \cdots + 2^{-l_k}$. We extend the definition of "Kraft sum" to also apply to sets of source symbols with respect to a code or sets

[1]For more details about Huffman codes, the reader is referred to the textbook [5, Section 5.6].

[2]Our usage of the word "complete" has also been referred to in the literature as "full", "extended", "saturated", "exhaustive", and "maximal".

of code tree nodes. In each case, we use the notation K_C to denote the Kraft sum. If C is a prefix code for a source with alphabet S, and $U \subset S$, then the Kraft sum of U is

$$K_C(U) = \sum_{x \in U} 2^{-l_C(x)}$$

or equivalently, the Kraft sum of the corresponding sequence of codeword lengths of the set of all leaf descendants of U in the code tree of C. The same summation is used to compute the Kraft sum of a set of code tree nodes.

The following lemma is a standard result in most information theory textbooks and will be used in the proofs of Lemma 2.2 and Theorem 2.4.

Lemma 1.5 (Kraft Inequality converse [5, Theorem 5.2.1])**.** *If a sequence $l_1, \ldots, l_n$ of positive integers satisfies $2^{-l_1} + \cdots + 2^{-l_n} \leq 1$, then there exists a binary prefix code whose codeword lengths are $l_1, \ldots, l_n$.*

The following lemma is used in the proof of Lemma 3.4.

Lemma 1.6 (Gallager [9, p. 670])**.** *For any source, if a prefix code is optimal, then it is monotone.*

As noted earlier, Gallager characterized Huffman codes (Lemma 1.8) using the following property.

Definition 1.7 (Gallager [9, p. 669])**.** A binary code tree has the *sibling property* if each node (except the root) has a sibling and if the nodes can be listed in order of non-increasing probability with each node being adjacent in the list to its sibling.

The next lemma is very useful in proving results about Huffman codes, and will be exploited in the proofs of Theorem 2.4, Lemma 3.4, and Theorem 3.8.

Lemma 1.8 (Gallager [9, Theorem 1])**.** *For any source, a prefix code is a Huffman code if and only if its code tree has the sibling property.*

For the remainder of this section, we describe our main results. Theorem 2.4 shows that for a prefix code C for a given probability distribution, the following are equivalent: (i) C is optimal; (ii) C is length equivalent to a Huffman code; and (iii) C is complete and strongly monotone. In this theorem, the Huffman code in case (ii) may vary for different choices of C. Figure 5 depicts these results along with some known prior art.

Next, we describe our remaining results in this paper.

Huffman codes for the same source can differ from each other in multiple ways. For example, "twisting" a Huffman tree about any fixed node in the tree (i.e., swapping two same-parent nodes) creates a new Huffman code for the source (Lemma 3.3).

Another transformation of a Huffman tree is to cut branches of the tree at two nodes whose probabilities equal each other and then exchange those subtrees (i.e., perform a same-probability node swap). Two such nodes need not lie on the same row in the tree. This transformation also results in a new Huffman code for the same source (Lemma 3.4).

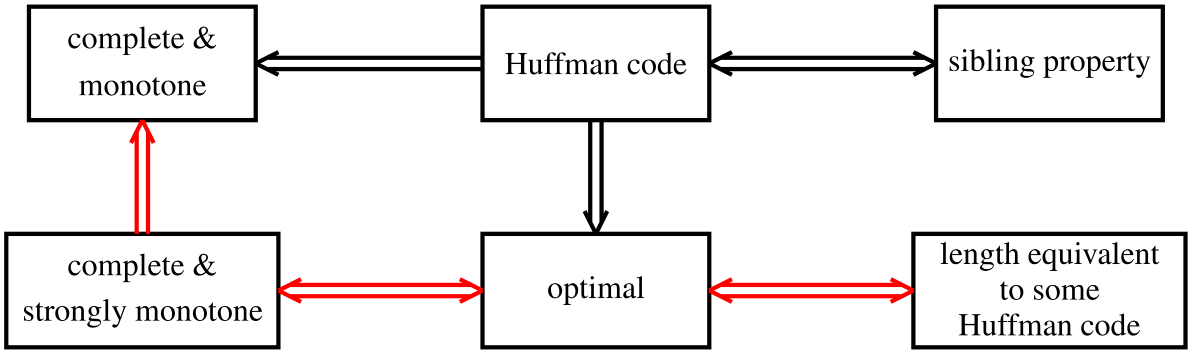
complete & monotone
Huffman code
sibling property
complete & strongly monotone
optimal
length equivalent to some Huffman code

Figure 5: Logical implications of prefix code properties for a given source. The red arrows indicate new results presented in this paper.

Any combination of these same-parent and same-probability node swaps transforms Huffman trees into Huffman trees. In fact, we show in Theorem 3.8 that the converse is also true, namely that all Huffman codes for a given source are related by these transformations.

More general questions exist about the broader class of optimal prefix codes. In this case, we know that any combination of same-parent and same-probability node swaps transforms optimal codes to other optimal codes (Lemma 3.2), neither of which is necessarily a Huffman code. It turns out not to be true that any optimal prefix code can be obtained from any other optimal prefix code by a sequence of same-parent and same-probability node swaps (see H_2 and C in Example 1.4). We describe another node swap involving cutting branches of a Huffman tree at two different nodes on the same row and then exchanging the hanging subtrees (i.e., swapping same-row nodes). This operation preserves the average length of any prefix code, so it maps optimal prefix codes to optimal prefix codes, but the operation need not transform Huffman codes into other Huffman codes. (see H_1 and H_2 in Example 1.4).

We show that for a given source, two complete prefix codes are length equivalent if and only if they are same-row swap equivalent (Theorem 3.6). This then implies that any prefix code for a given source is length equivalent to a particular Huffman code (and thus is optimal) if and only if its code tree can be obtained from the Huffman tree by a sequence of same-row node swaps (Corollary 3.7).

If we replace same-parent node swaps by the more general same-row node swaps, then we can characterize optimal prefix codes in another way by using node swapping. Specifically, we show that all optimal prefix codes are same-row, same-probability swap equivalent to each other (Theorem 3.9).

2. Characterization of expected length minimizing prefix codes

In this section we give a new characterization of optimal prefix codes for a given source. While all Huffman codes are optimal and were characterized by Gallager in terms of the sibling property, not all optimal codes are Huffman codes. However, it turns out that any optimal code is length equivalent to some Huffman code for the source as shown in Theorem 2.4.

In Theorem 2.4 we also prove a second characterization of optimal prefix codes. Specifically, we show that a prefix code is optimal if and only if it is complete and strongly monotone. The combination of completeness and strong monotonicity is weaker than the sibling property, and thus a broader class of prefix codes (namely, the optimal ones) satisfies this combination.

Definition 2.1. Given a source with alphabet S, a prefix code C is *strongly monotone* if $P(A) \geq P(B)$ whenever $A, B \subseteq S$ and $K_C(A) = 2^{-i} > 2^{-j} = K_C(B)$ for some integers i and , j.

The strongly monotone property reduces to Gallager's monotone property when each of A and B consists of all leaf descendants of a single tree node. Example 1.3 illustrates that these two properties are not equivalent. Specifically, the example shows that prefix code C is not strongly monotone because $K_C(\{c,d\}) = 2^{-1} > 2^{-2} = K_C(\{a\})$ but $P(\{c,d\}) = \frac{1}{4} < \frac{3}{8} = P(\{a\})$. The code H is strongly monotone since it is a Huffman code.

The following lemma easily follows from the proof of Lemma 1.5. This lemma relies on our definition of sources (and thus codes) to be finite. Prefix codes for infinite sources need not satisfy the lemma below (e.g. [15, p. 2027]).

Lemma 2.2. *A prefix code is complete if and only if its Kraft sum equals* 1.

Lemma 2.3. *If two prefix codes are length equivalent, then each of the following properties holds for one code if and only if it holds for the other code:*

(1) completeness

(2) strong monotonicity

(3) optimality.

Proof. Let S be the source alphabet. Let C and C' be length equivalent prefix codes, i.e., $l_C(y) = l_{C'}(y)$ for all $y \in S$. Then for all $y \in S$,

$$K_C(y) = 2^{-l_C(y)} = 2^{-l_{C'}(y)} = K_{C'}(y).$$

Since

$$\sum_{y \in S} K_C(y) = \sum_{y \in S} K_{C'}(y),$$

Lemma 2.2 shows C is complete if and only if C' is complete.

Suppose C is strongly monotone. Let $A, B \subseteq S$ with $K_{C'}(A) = 2^{-i}$ and $K_{C'}(B) = 2^{-j}$ for some integers $0 \leq i < j$. Since $K_C(A) = K_{C'}(A) = 2^{-i}$ and $K_C(B) = K_{C'}(B) = 2^{-j}$, we have $P(A) \geq P(B)$ since C is strongly monotone. Thus C' is also strongly monotone.

Let X be a source random variable. The average length of code C is

$$E[l_C(X)] = \sum_{y \in S} P(y) l_C(y) = \sum_{y \in S} P(y) l_{C'}(y) = E[l_{C'}(X)],$$

so C is optimal if and only if C' is. ■

The following theorem is our first main result.

Theorem 2.4. *For a given source, if C is a prefix code, then the following are equivalent:*

(1) C is complete and strongly monotone;

(2) C is length equivalent to a Huffman code; and

(3) C is optimal.

Proof. Let S be the source alphabet, and let X be the source random variable on S. Define $P(u) = P(X = u)$ for all $u \in S$.

$(1) \implies (2)$

Suppose C is complete and strongly monotone. Consider the following operation on the code tree for C. Suppose row $k \geq 1$ of the code tree has the property that all non-leaves are listed in order of non-increasing probability moving left-to-right in the row. Permute all nodes in row k such that they are listed in order of non-increasing probability moving left-to-right in the row. Note that the non-leaves remain in the same order among themselves as they were prior to performing the operation, and therefore all nodes in each row $m > k$ remain in the same order in their row as they were prior to performing the operation. Let C' be the new code corresponding to the code tree obtained after this operation. This operation may change the codewords assigned to symbols in S, but it does not change the lengths of any codewords, and so C and C' are length equivalent. Moreover, once this operation has been performed, the probabilities of the parents of the nodes in row k are listed in order of non-increasing probability moving left-to-right in row $(k-1)$.

Therefore, we can apply this node permutation operation iteratively on successively higher rows, beginning on the bottom row of the code tree for C where there are only leaves, and then moving upward, row-by-row. Once the leaves in the bottom row have been ordered, the probabilities of their parents have been ordered, and we can then proceed by induction to conclude that after this operation has been performed on each row, the nodes in each row of the resulting code tree are listed in order of non-increasing probability moving left-to-right in the row. Let C' be the code corresponding to the resulting code tree. Inductively, based on the argument in the previous paragraph, C' is length equivalent to C, and so C' is complete and strongly monotone by Lemma 2.3. Consider the sequence of probabilities of nodes in the code tree for C' listed in raster-scan order, beginning with the root node and proceeding downward, row-by-row, moving left-to-right in each row. Then the probability of each node is adjacent in the sequence to the probability of its sibling. Also, as concluded previously, all probabilities of nodes in the same row are listed in non-increasing order in the sequence. Furthermore, since C' is strongly monotone, the rightmost node u in a given row $k \geq 0$ and the leftmost node v in row $(k+1)$ satisfy $P(u) \geq P(v)$, since $K_{C'}(u) = 2^{-k} > 2^{-(k+1)} = K_{C'}(v)$. Therefore, the sequence of probabilities is listed in order of non-increasing probability, and so the code tree for C' satisfies the sibling property since C' is also complete. Thus, by Lemma 1.8, C' is a Huffman code. Since C is length equivalent to C', we are done.

$(2) \implies (3)$
Suppose C is length equivalent to a Huffman code H. Then by Lemma 2.3, C is optimal because H is.

$(3) \implies (1)$
Suppose C is optimal. If C is not complete, then there exists a non-root node u without a sibling in the code tree for C. Replacing the parent of u by u itself results in a code tree where the lengths of all leaf descendants of u have been decreased by 1, and the lengths of all other leaves have remained unchanged. Therefore, the expected length of the new code is $P(u)$ less than the expected length of C. Since $P(u) > 0$, this shows C is not optimal, which is a contradiction. Thus C is complete.

Suppose C is not strongly monotone. Then there exist $A, B \subseteq S$ and integers i and j, such that $0 \le i < j$, $K_C(A) = 2^{-i}$, $K_C(B) = 2^{-j}$, and $P(A) < P(B)$. Denote the symmetric difference of A and B by $A\Delta B = (A - B) \cup (B - A)$. Then

$$\begin{aligned} K_C(A-B) &= K_C(A) - K_C(A\cap B) = 2^{-i} - K_C(A \cap B) \\ K_C(B-A) &= K_C(B) - K_C(A\cap B) = 2^{-j} - K_C(A \cap B) \\ K_C(S-(A\Delta B)) &= 1 - K_C(A-B) - K_C(B-A) \\ &= 1 - 2^{-i} - 2^{-j} + 2K_C(A\cap B) \end{aligned}$$

and

$$P(A-B) = P(A) - P(A\cap B) < P(B) - P(A \cap B) = P(B-A). \tag{1}$$

Let C' be a prefix code such that:

$$l_{C'}(y) = l_C(y) + \begin{cases} j-i & \text{if } y \in A-B \\ i-j & \text{if } y \in B-A \\ 0 & \text{if } y \in S-(A\Delta B). \end{cases} \tag{2}$$

Note that such a prefix code exists by Lemma 1.5, since

$$\begin{aligned} &\sum_{y\in S} 2^{-l_{C'}(y)} \\ &= 2^{-(j-i)} \sum_{y\in A-B} 2^{-l_C(y)} + 2^{j-i} \sum_{y \in B-A} 2^{-l_C(y)} + \sum_{y\in S-(A\Delta B)} 2^{-l_C(y)} && (3) \\ &= 2^{-(j-i)} K_C(A-B) + 2^{j-i}K_C(B-A) + K_C(S-(A\Delta B)) \\ &= 2^{-j} - 2^{-(j-i)}K_C(A\cap B) + 2^{-i} - 2^{j-i}K_C(A\cap B) + 1 - 2^{-i} - 2^{-j} + 2K_C(A\cap B) && (4) \\ &= 1 + (2 - 2^{j-i} - 2^{-(j-i)})K_C(A\cap B) \\ &\le 1 && (5) \end{aligned}$$

where (3) follows from (2); (4) follows from $K_C(A) = 2^{-i}$ and $K_C(B) = 2^{-j}$; and (5) follows from $j > i$. Equality holds in (5) if and only if $K_C(A \cap B) = 0$, since $2^{j-i} \geq 2$. Finally,

$$\begin{aligned}
E[l_{C'}(X)] &= \sum_{y \in A-B} P(y) l_{C'}(y) + \sum_{y \in B-A} P(y) l_{C'}(y) + \sum_{y \in S-(A\Delta B)} P(y) l_{C'}(y) \\
&= \sum_{y \in A-B} P(y)(l_C(y) + (j-i)) + \sum_{y \in B-A} P(y)(l_C(y) - (j-i)) + \sum_{y \in S-(A\Delta B)} P(y) l_C(y) \\
&= (j-i)(P(A-B) - P(B-A)) + E[l_C(X)] \\
&< E[l_C(X)], \qquad (6)
\end{aligned}$$

where (6) follows from $j - i \geq 1$ and $P(A-B) < P(B-A)$ by (1). But then C is not optimal, which is a contradiction. Therefore, C is strongly monotone. ■

We note that a proof of $(3) \implies (2)$ in Theorem 2.4 does not seem to have previously appeared explicitly in the literature, although it may be hinted at in the proof of [5, Lemma 5.8.1, p. 123] and also in 1995 by Yamamoto and Itoh [18].

The following corollary immediately follows from Theorem 2.4 and describes which codes perform as well as Huffman codes in terms of average length.

Corollary 2.5. *For a given source, a complete prefix code C is optimal if and only if C is strongly monotone.*

3. Swapping code tree nodes

Definition 3.1. A *node swap* in a code tree is an exchange of the subtrees rooted at two distinct nodes neither of which is a descendant of the other. Such a node swap is called a *same-row* (respectively, *same-probability*) node swap if the two nodes are in the same row (respectively, have the same probability). A *same-parent* node swap is a same-row node swap of two siblings. Prefix codes C_1 and C_2 are *same-parent swap equivalent* (respectively, *same-row swap equivalent*, *same-probability swap equivalent*) if the code tree of C_1 can be transformed into the code tree of C_2 by a sequence of same-parent (respectively, same-row, same-probability) node swaps. Additionally, prefix codes C_1 and C_2 are *same-parent, same-probability swap equivalent* (respectively, *same-row, same-probability swap equivalent*) if the code tree of C_1 can be transformed into the code tree of C_2 by a sequence of same-parent (respectively, same-row) and/or same-probability node swaps.

The relations of same-parent, same-row, and same-probability swap equivalence are all reflexive, symmetric, and transitive, and thus are equivalence relations. Therefore, they each naturally induce equivalence classes of prefix codes.

For a given source, any same-parent node swap transforms a Huffman tree to another Huffman tree, since in the Huffman construction process when two nodes are merged the choice of their order in a sibling pair is arbitrary.

The following two lemmas are straightforward, so we omit their proofs.

Lemma 3.2. *For a given source, same-parent, same-row, and same-probability node swaps preserve the expected length of any complete prefix code.*

Lemma 3.3. *For a given source, any same-parent node swap of a Huffman code produces another Huffman code.*

Since each merging in a Huffman tree construction has two possible orderings, and there are $n-1$ internal nodes in a binary tree with n leaves, there are 2^{n-1} different Huffman codes in each same-parent swap equivalence class that contains at least one Huffman code. As shown in Lemma 3.3, any Huffman code in such an equivalence class can be obtained from another Huffman code in the class by performing a sequence of same-parent node swaps. However, two Huffman codes for a given source need not be related by a sequence of same-parent node swaps (i.e., they can be in different same-parent swap equivalence classes).

The next lemma shows that same-probability node swaps convert Huffman trees to other Huffman trees for the same source.

Lemma 3.4. *For a given source, any same-probability node swap of a Huffman code produces another Huffman code.*

Proof. By Lemma 1.8, the Huffman tree H satisfies the sibling property. That is, there exists a sequence $\sigma_H = u_1, u_2, \dots, u_i, \dots, u_j, \dots$ of the nodes of H in non-increasing order of their probabilities where siblings appear adjacent in the list.

Suppose nodes u_i and u_j are swapped in Huffman tree H to give code tree C, where $P(u_i) = P(u_j)$. Swapping nodes modifies the tree H by altering two edges, and retaining all the same nodes and the other edges.

The nodes of the new code tree C are the same as the nodes of H. Modify the sequence of nodes of H by exchanging u_i with u_j to obtain the sequence $\sigma_C = u_1, u_2, \ldots, u_j, \ldots, u_i, \ldots$, which is a sequence of the nodes of C.

All of the probabilities of the nodes in C remain the same as they were in H (since the parent nodes of u_i and u_j in C have children with the same probabilities as they had in H), so the probabilities of the nodes in the modified sequence σ_C are in non-increasing order. Also, u_i and u_j appear adjacent in σ_C to their siblings, since u_j and u_i, respectively, do so in σ_H. Thus, the second tree satisfies the sibling property, and so is a Huffman tree by Lemma 1.8. ■

Note that if a Huffman tree node x lies at least two rows above node y, then Lemma 1.6 implies the probability of x is at least as great as the probability of the parent of y, which is strictly greater than the probability of y, so x and y cannot have the same probability. Therefore, in a Huffman tree, any two nodes with the same probability must either lie in the same row or in adjacent rows. This means that any same-probability node swap in a Huffman tree consists of swapping two nodes in the same row or adjacent rows. However, sequences of same-probability node swaps can transform one Huffman tree into another where a node can move farther than one row away, as the following example illustrates.

Example 3.5 (Two Huffman trees with a source symbol two rows apart)**.**
Let H_1 and H_2 be two different Huffman codes for a source with symbols a,b,c,d,e and probabilities $\frac{1}{3}$, $\frac{1}{3}$, $\frac{1}{9}$, $\frac{1}{9}$, $\frac{1}{9}$, respectively. The two trees are shown in Figure 6. The tree H_1 can be transformed into H_2 by performing a same-probability node swap between leaf a and the parent of leaf c, and subsequently performing a same-probability node swap between leaf c and leaf e. Note that the source symbol c appears in row 2 of H_1 and row 4 of H_2. In fact, one can construct examples where the same source symbol appears in two different Huffman trees in arbitrarily distant rows.

By Lemma 3.3 and Lemma 3.4 any sequence of same-parent node swaps and same-probability node swaps converts Huffman codes to Huffman codes for the same source. Thus, the set of all codes obtained by such transformations on a given Huffman code is an equivalence class containing Huffman codes only. In fact, this equivalence class contains all Huffman codes for a given source (Theorem 3.8).

Same-row node swaps preserve codeword lengths assigned to source symbols. If a same-row node swap is not a same-parent node swap, then it may convert a Huffman code into a non-Huffman code (e.g., the prefix code C in Figure 3). However, these non-Huffman codes are still optimal (Corollary 3.7).

The following lemma shows that length equivalence between two complete prefix codes is the same as being able to transform one into the other using only same-row node swaps.

Theorem 3.6. *For a given source, two complete prefix codes are length equivalent if and only if they are same-row swap equivalent.*

Proof. Same-row node swaps preserve the lengths of all codewords, so length equivalence follows immediately.

Conversely, suppose C_1 and C_2 are code trees for two length equivalent prefix codes. For any complete code tree for the source, assign labels to its nodes as follows. Label each leaf node by

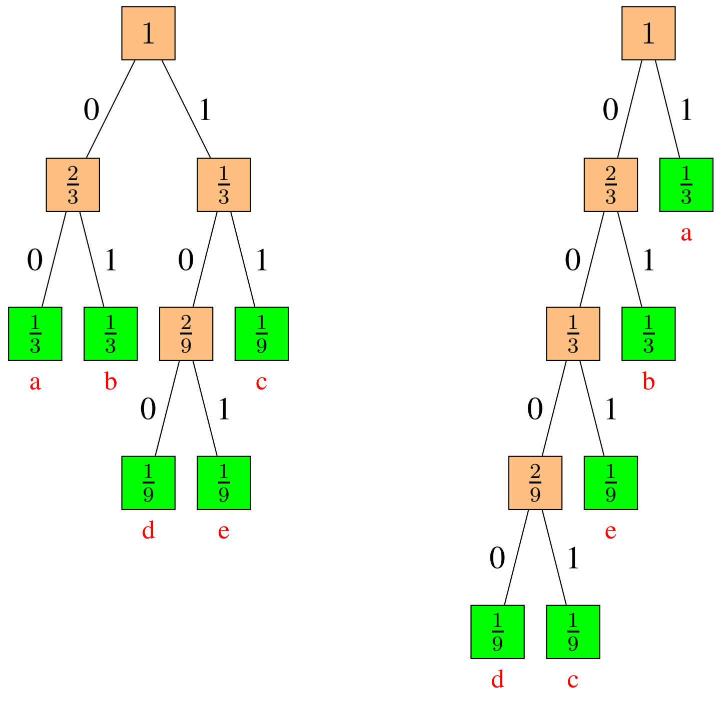

Figure 6: Two Huffman trees for the same source with source symbol c appearing on rows differing in level by two.

the source symbol of its associated codeword, and label each parent node as the ordered pair (a, b), where a is the label of its left child and b is the label of its right child.

Let d be the common depth (i.e., longest codeword length) of C_1 and C_2. All nodes in row d in both C_1 and C_2 are leaves, and since C_1 and C_2 are length equivalent, the set of leaf labels in row d is identical for C_1 and C_2.

We will show that if the set of node labels in some row $m \in \{1, \ldots, d\}$ is identical for C_1 and C_2, then there exists a sequence of same-row node swaps that transforms C_1 into a code tree C_1' such that the set of node labels in row $m-1$ is identical for C_1' and C_2. Then, since we have already shown the set of node labels in the lowest row d is identical for C_1 and C_2, we conclude by induction that there exists a sequence of same-row node swaps that transforms C_1 into a code tree C_1^* whose root has the same label as the root of C_2. The label of the root node of a code tree specifies the code tree exactly (it's a parenthetically nested list of node pairs used to form the tree), so C_1^* and C_2 are the same code tree, and we conclude C_1 and C_2 are same-row swap equivalent.

Suppose the set of node labels in some row $m \in \{1, \ldots, d\}$ is identical for C_1 and C_2. In other words, the same node labels appear in row m of each code tree, but the node labels are possibly arranged in a different order. Then there exists a permutation that maps the arrangement of node labels in row m of C_1 to the arrangement of node labels in row m of C_2. By a standard group theory result that any finite permutation can be obtained from any other permutation by a sequence of transpositions (e.g., see [6, p. 107]), there exists a sequence of same-row node swaps that achieves this permutation and transforms C_1 into a code tree C_1' whose sequence of node labels in row m is identical to that of C_2. Since sibling nodes are adjacent in this sequence, the set of labels of the parent nodes in row $m-1$ is identical in C_1' and C_2.

Since C_1 and C_2 are length equivalent, the set of leaf node labels in row $m-1$ is identical in C_1 and C_2, and the same is true for C_1' and C_2 since the leaf nodes in row $m-1$ of C_1 were unaffected by the same-row node swaps performed on the nodes in row m. Therefore, the set of node labels in row $m-1$ is identical for C_1' and C_2, and the induction argument is complete. ■

The following theorem shows that prefix code optimality is equivalent to being able to transform the code into some Huffman code using only same-row node swaps.

Corollary 3.7. *For a given source, a prefix code is optimal if and only if it is same-row swap equivalent to a Huffman code.*

Proof. It follows immediately from Theorem 3.6 and Theorem 2.4. ■

Corollary 3.7 guarantees for any optimal prefix code the existence of some Huffman code that is same-row swap equivalent. However, different optimal prefix codes need not all be same-row swap equivalent to the same Huffman code. The next theorem indicates, however, that all Huffman codes can be transformed to each other using only same-parent, same-probability node swap operations. Then, by combining Corollary 3.7 and Theorem 3.8, we obtain Theorem 3.9, which states that all optimal prefix codes are same-row, same-probability swap equivalent.

Theorem 3.8. *For a given source, all Huffman codes are same-parent, same-probability swap equivalent to each other.*

Proof. For any complete code tree for the source, assign labels to its nodes as follows. Label each leaf node by the source symbol of its associated codeword, and label each parent node as the ordered pair (a, b), where a is the label of its left child and b is the label of its right child. Given a node label u, let $P(u)$ be the probability of the node labeled u.

Let H_1 and H_2 be two Huffman code trees for a source. Then in particular, H_1 and H_2 are complete by Lemma 1.8. Since H_1 and H_2 are code trees for the same source, the set of leaf node labels is identical in H_1 and H_2.

Let $(a_1, b_1), (a_2, b_2), \dots$ and $(a_1', b_1'), (a_2', b_2'), \dots$ be the sequences of the parent node labels (i.e., pairs of merged child labels) in H_1 and H_2, respectively, listed in the order of each step of the Huffman construction that produced each code tree.

Let i be the smallest index such that $(a_i, b_i) \neq (a_i', b_i')$. The set of node labels available for merging during step i is identical in the Huffman construction of H_1 and H_2, since such a node label is either a leaf node label common to both H_1 and H_2, or is a parent node label constructed in a step prior to i. Then in particular, all the node labels in $\{a_i, b_i, a_i', b_i'\}$ are available for merging at step i in the Huffman construction of both H_1 and H_2. Also, since the Huffman algorithm combines at each step two nodes of smallest probability, we have $\{P(a_i), P(b_i)\} = \{P(a_i'), P(b_i')\}$.

If $a_i' \notin \{a_i, b_i\}$, then there exists $u_i \in \{a_i, b_i\}$ such that $P(u_i) = P(a_i')$. Since both u_i and a_i' appear in H_1, we can remove the edges connecting u_i and a_i' to their parent nodes in H_1, and add edges connecting u_i and a_i' to the parent nodes of a_i' and u_i, respectively. Since $P(u_i) = P(a_i')$, this operation is a same-probability node swap, and by Lemma 3.4 the resulting code tree is a Huffman code tree.

After performing this same-probability node swap if necessary, we arrive at a Huffman code tree that combines a_i' and $v_i \in \{a_i, b_i\}$ in step i, where $P(v_i) = P(b_i')$. If $v_i \neq b_i'$, then an analogous argument as above shows a same-probability node swap can be performed on v_i and b_i' to produce a new Huffman code tree.

After performing this same-probability node swap if necessary, we arrive at a Huffman code tree that combines a_i' and b_i' in step i. If necessary, perform a same-parent node swap on the sibling pair $\{a_i', b_i'\}$ such that their parent label is (a_i', b_i'), and name this code tree H_1', which is a Huffman code tree by Lemma 3.3. This same-parent node swap and the same-probability node swaps described previously leave the labels of parent nodes in H_1 obtained in any steps prior to i unaffected, and as a consequence, the Huffman constructions of H_1' and H_2 produce identical parent node labels at each step up to and including step i.

By induction on i, there exists a sequence of same-parent node swaps and same-probability node swaps that transforms H_1 into a Huffman code tree H_1^*, such that H_1^* and H_2 produce identical parent node labels at every step of the Huffman construction. In particular, the root of H_1^* has the same label as the root of H_2. The label of the root node of a code tree specifies the code tree exactly, so H_1^* and H_2 are the same code tree, and the lemma is proved. ■

Theorem 3.9. *For a given source, all optimal prefix codes are same-row, same-probability swap equivalent to each other.*

Proof. Let C_1 and C_2 be optimal prefix codes. By Corollary 3.7, C_1 is same-row swap equivalent to some Huffman code H_1, and C_2 is same-row swap equivalent to some Huffman code H_2. Then by Theorem 3.8, H_1 is same-parent, same-probability swap equivalent to H_2. Therefore, C_1 and C_2 are same-row, same-probability swap equivalent to each other. ■

References

[1] J. Abrahams, “Code and parse trees for lossless source encoding”, *Proceedings of the Compression and Complexity of Sequences*, Salerno, Italy, pp. 145 – 171, June 1997.

[2] L. Cao, “Self-synchronization strings in Huffman equivalent codes”, *IEEE Information Theory Workshop*, Chengdu, China, pp. 347 – 350, October 2006.

[3] L. Cao, L. Yao, C. W. Chen, “MAP decoding of variable length codes with self-synchronization strings”, *IEEE Transactions on Signal Processing*, vol. 55, no. 8, pp. 4325 – 4330, September 2007.

[4] S. Congero and K. Zeger, “Competitive advantage of Huffman and Shannon-Fano codes”, *IEEE Transactions on Information Theory* (submitted November 13, 2023). Also available on: arXiv:2311.07009 [cs.IT].

[5] T. M. Cover and J. A. Thomas, *Elements of Information Theory*, 2nd edition, New Jersey, Wiley-Interscience, 2006.

[6] D. S. Dummit and R. M. Foote, *Abstract Algebra*, 3rd edition, John Wiley & Sons, Inc., 2004.

[7] A. E. Escott and S. Perkins, “Binary Huffman equivalent codes with a short synchronizing codeword”, *IEEE Transactions on Information Theory*, vol. 44, no. 1, pp. 346 – 351, January 1998.

[8] T. Ferguson and J. Rabinowitz, “Self-synchronizing Huffman codes”, *IEEE Transactions on Information Theory*, vol. 30, no. 4, pp. 687 –6 93, July 1984.

[9] R. G. Gallager, “Variations on a theme by Huffman”, *IEEE Transactions on Information Theory*, vol. 24, no. 6, pp. 668 – 8674, November 1978.

[10] M. Higgs, “The construction of variable length codes with good synchronisation properties”, *Doctoral thesis*, University of South Wales Prifysgol De Cymru, February 26, 2007.

[11] M. B. J. Higgs, S. Perkins, and D. H. Smith, “The construction of variable length codes with good synchronization properties”, *IEEE Transactions on Information Theory*, vol. 55, no. 4, pp. 1696 – 1700, April 2009.

[12] Y.-M. Huang and S.-C. Wu, “Shortest synchronizing codewords of a binary Huffman equivalent code”, *International Conference on Information Technology: Coding and Computing*, Las Vegas, NV, pp. 226 – 231, April 2003.

[13] D. A. Huffman, “A method for the construction of minimum-redundancy codes”, *Proceedings of the IRE*, vol. 40, no. 9, pp. 1098 – 1101, September 1952.

[14] Y.-S. Lee, W.-S. Chang, H.-H. Ho, C.-Y. Lee, “Construction of error resilient synchronization codeword for variable-length code in image transmission”, *International Conference on Image Processing*, vol. 3, pp. 360 – 363, September 2000.

[15] T. Linder, V. Tarokh, and K. Zeger, “Existence of optimal codes for infinite source alphabets”, *IEEE Transactions on Information Theory*, vol. 43, no. 6, pp. 2026 – 2028, November 1997.

[16] G. Longo and G. Galasso, “An application of informational divergence to Huffman codes”, *IEEE Transactions on Information Theory*, vol. 28, no. 1, pp. 36 – 43, January 1982,

[17] D. Wang, X. Zhao, Q. Sun, “Novel fault-tolerant decompression method of corrupted Huffman files”, *Wireless Personal Communications*, vol. 102, no. 4, pp. 2555 – 2574, October 2018.

[18] H. Yamamoto and T. Itoh, “Competitive optimality of source codes”, *IEEE Transactions on Information Theory*, vol. 41, no. 6, pp. 2015 – 2019, November 1995.

[19] J. Zhou, O. C. Au, “Error recovery of variable length code over BSC with arbitrary crossover probability”, *IEEE Transactions on Communications*, vol. 58, no. 6, pp. 1654 – 1666, June 2010.

[20] G. Zhou and Z. Zhang, “synchronization recovery of variable-length codes”, *IEEE Transactions on Information Theory*, vol. 48, no. 1, pp. 219 – 227, February 2002.