Shortest Synchronizing Strings for Huffman Codes[*] (Extended Abstract)

Marek Tomasz Biskup

Institute of Informatics, University of Warsaw,
Banacha 2, 02-097 Warszawa, Poland
mbiskup@mimuw.edu.pl
http://www.mimuw.edu.pl/∼mbiskup

Abstract. Most complete binary prefix codes have a synchronizing string, that is a string that resynchronizes the decoder regardless of its previous state. This work presents an upper bound on the length of the shortest synchronizing string for such codes. Two classes of codes with a long shortest synchronizing string are presented. It is known that finding a synchronizing string for a code is equivalent to a finding a synchronizing string of some finite automaton. The Černý conjecture for this class of automata is discussed.

1 Introduction

Huffman codes are the most popular variable length codes. In the presence of channel errors a large part of an encoded message can be destroyed because of the loss of synchronization between the decoder and the coder. In case of some Huffman codes, under certain assumptions on the message source, the decoder will eventually resynchronize, and, from then on, symbols will be decoded correctly. These codes are called *synchronizing*. Capocelli et al. [1] proved that codes are synchronizing if and only if they have a *synchronizing string* — a string such that when received by the decoder always puts it into synchronization. Freiling et al. [2] proved that almost all Huffman codes have a synchronizing string. More precisely, they proved that the probability of drawing randomly a code without a synchronizing string decreases to zero with increasing code size.

Shützenberger [3] analyzed possible distribution of codewords' lengths in a synchronizing prefix codes. Rudner [4] gave an algorithm for the construction of a synchronizing Huffman code for a given distribution of codewords' lengths, that works under some assumptions on the distribution. His work was further extended in [5,6]. Capocelli et al. [7] showed how to modify a Huffman code by adding a little redundancy to create a synchronizing code. Ferguson and Rabinowitz [8] analyzed codes whose synchronizing string is a codeword.

[*] The research was partially supported by the grants of the Polish Ministry of Science and Higher Education N 206 004 32/0806 and N N206 376134.

The synchronization recovery of a Huffman code can be modeled with a finite automaton whose states are proper prefixes of codewords (or internal nodes of the code's tree). This automaton will be called a *Huffman automaton*. Such an automaton was used by Maxted and Robinson [9] to compute for a given code the average number of symbols lost before resynchronization.

A lot of research has been done in the area of automata synchronization. A synchronizing string for a finite automaton $\langle Q, \Sigma, \delta \rangle$ is a string s that brings all states to one particular state. That is $\delta(q_1, s) = \delta(q_2, s)$ for any states $q_1, q_2 \in Q$. An automaton will be called *synchronizing* if it has a synchronizing string. The famous Černý conjecture [10] states that a synchronizing finite automaton with N states has a synchronizing string of length $(N-1)^2$.

Although there are proofs for certain classes of automata, for instance in [11,12], the problem remains open. There are some bounds on the length of the shortest synchronizing string. For instance Pin [13] proved that $\frac{1}{6}(N^3 - N)$ is an upper bound. Some research has also been done to find automata with long shortest synchronizing strings. Černý [10] constructed a series of automata with the shortest synchronizing string of length $(N-1)^2$. Ananichev et al. [14] considered how long a synchronizing string can be if there is a letter that reduces the number of states by two. Trahtman [15] searched for worst-case automata.

Eppstein [16] gave an algorithm for testing whether an automaton is synchronizing and for the construction of a synchronizing string of length $O(N^3)$ for a synchronizing automaton. His algorithm requires $O(N^3)$ operations if the alphabet is of constant size. An overview of the area of automata synchronization is given in [17].

It is rather clear that a synchronizing string for a Huffman code is also a synchronizing string for the Huffman automaton of the code, and vice versa. Nevertheless, it seems that so far both areas of research have not been related. This paper fills this gap.

First we explain that Huffman code synchronization is equivalent to Huffman automaton synchronization. Then, we prove an upper bound on the length of the shortest *merging string* for a set of two states of a Huffman automaton: the root of the code's tree and another internal node of the tree. A merging string for a set of states is a string that brings all states of the set to the same state. The proof is constructive and an algorithm for the construction of the shortest merging string for such nodes is given. The execution of this algorithm also suffices for answering whether a code is synchronizing. Then we present an upper bound on the length of the shortest synchronizing string of a Huffman automaton. For most (but not all) codes the bound is better than the Černý conjecture. Also an algorithm for the construction of a synchronizing string for a Huffman automaton is presented. To the author's best knowledge, this class of automata has not been studied yet. The bounds presented here are better than the bounds $O(N^3)$ for general automata. Both algorithms are faster than the one of Eppstein [16].

Afterwards, results of experimental search for worst-case codes are shown. Three classes of Huffman codes are presented. The codes give a lower estimate on the possible upper bounds of the length of the shortest synchronizing or merging string. It is conjectured (but, unfortunately, not proved) that these classes of codes are the worst-case codes. It is interesting that the length of their synchronizing or merging strings is much lower than the bound proved.

Due to limited length of the paper, the most difficult proofs are omitted.

2 Definitions and Notation

A *word* is a string of letters, for instance $w = w_0 w_1 \dots w_{k-1}$. The empty word is denoted by ϵ. The subword of a word w from the position p to $q-1$ is denoted by $w[p..q)$. The length of a word w is denoted by $|w|$. A sequence of k letters a is denoted a^k. For instance, for w=‘abc’, $w[1..2)$=‘b’, $w[1..3)$=‘bc’, $w[0..1)$=‘a’, $|w| = 3$ and 0^4 = ‘0000’.

A *complete binary tree* is a tree with each node being either an *internal node* with two children, or a *leaf* with no children. Each left outgoing edge is labeled with 0 (0-edge). Each right outgoing edge is labeled with 1 (1-edge). The root of a tree is denoted by ε. Each node n has a unique binary string $\pi(n)$ that is formed of labels on the path from the root to n. We have $\pi(\varepsilon) = \epsilon$. The number of leaves in a tree is denoted by N. The height of a tree is denoted by h. In this paper, a code C such that $C = \{\pi(n) | n \text{ is a leaf of } T\}$ for some complete binary tree T, is called a *Huffman code*. The tree T is called a *Huffman tree*. We refer to a node n of T using the string $\pi(n)$. For instance, the node 10 is the left son of the right son of the root.

Let a *Huffman Automaton* (HA) $\mathcal{T}$ be an automaton whose states are internal nodes of the Huffman tree T. The transition function $\delta(n, b)$, $b \in \{0, 1\}$, brings an automaton from the node n to its b-edge child, if it is not a leaf, or to the root otherwise. The function δ^* is the extension of δ to strings: $\delta^*(q, b_0 \dots b_{k-1}) = \delta(\delta^*(q, b_0 \dots b_{k-2}), b_{k-1})$ and $\delta^*(q, \epsilon) = q$. For a subset S of states of a Huffman automaton we denote, $\delta(S, a) := \{\delta(q, a) | q \in S\}$. The same convention is used for δ^*.

We say that a word w *brings* a node n to a node n' if $n' = \delta^*(n, w)$. Then n' is the result of *applying* w to n. In addition, we say that w brings a node n to a leaf if $\delta^*(n, w) = \varepsilon$ and w is not empty. This is justified because the construction of the Huffman automaton $\mathcal{T}$ may be seen as merging the leaves of the tree T with the root of T. We say that w brings a node n to n' *without loops* if none of the nodes $\delta^*(n, w[0, 1)), \delta^*(n, w[0, 2)), \dots, \delta^*(n, w[0, |w| - 2))$ is the root.

The values T, $\mathcal{T}$, δ, δ^*, N, h, ε, π depend on the code C. We assume that it is always clear from the context which code (or, equivalently, which Huffman tree) is being considered.

Definition 1. *A* synchronizing string *for a Huffman code is a string w_s such that ww_s is a sequence of codewords for any binary word w.*

Equivalently, a synchronizing string is a string that brings any node of the Huffman automaton to the root.

Definition 2. *Let $\mathcal{A} = \langle Q, \Sigma, \delta \rangle$ be a finite automaton. A* synchronizing string *for $\mathcal{A}$ is a word w such that $|\delta^*(Q, w)| = 1$. A* merging string *for a set of states $R \subseteq Q$ of the automaton $\mathcal{A}$ is a word w such that $|\delta^*(R, w)| = 1$.*

Definition 3. *Let $\mathcal{A} = \langle Q, \Sigma, \delta \rangle$ be a finite automaton. The* power automaton *for $\mathcal{A}$ is the automaton $\mathcal{P}(\mathcal{A}) = (\mathcal{P}(Q), \Sigma, \delta_{\mathcal{P}})$, where $\mathcal{P}(Q)$ denotes the set of all subsets of Q and $\delta_{\mathcal{P}}(S, a) = \delta(S, a)$ for $S \in \mathcal{P}(Q)$.*

The operation of the power automaton $\mathcal{P}(\mathcal{A})$ can be seen as movements of coins that lie on some states of the automaton $\mathcal{A}$. If the power automaton is in a state $S \subseteq Q$, the coins lie on the states $q \in S$. Then, if the power automaton makes a transition by a letter a, the coins move according to the transition function δ of the automaton $\mathcal{A}$. If a coin is in the state p then it moves onto the state $\delta(p, a)$. If more than one coin goes to the same state only one of them is kept. It easy to see that after applying the letter a the set of states with coins is exactly $\delta_{\mathcal{P}}(S, a)$. This analogy helps to visualize the operation of the power automaton and gives some intuition. For instance, the string w is synchronizing if and only if applying w to the automaton $\mathcal{A}$ with a coin on each state results in just one coin left.

An automaton is *synchronizing* if it has a synchronizing string. A Huffman code is *synchronizing* if it has a synchronizing string.

Theorem 4. *A synchronizing string for a Huffman code C is a synchronizing string for the Huffman automaton $\mathcal{T}$ of the code. A synchronizing string s for the Huffman automaton $\mathcal{T}$, such that s brings all nodes to the root, is a synchronizing string for the Huffman code.*

Thus a Huffman code is synchronizing if and only if its Huffman automaton is synchronizing.

3 Merging String for a Pair of States

Theorem 5. *Let C be a synchronizing Huffman code of size N, let T be the Huffman tree for C, let $\mathcal{T}$ be the Huffman automaton of the code C. For any node n of $\mathcal{T}$ there is a merging string s_n for the set $\{n, \varepsilon\}$, with*

$$|s_n| \leq \sum_{p \in Q(T) \setminus \{\varepsilon\}} h_p, \tag{1}$$

where $Q(T)$ is the set of the internal nodes of T and h_p is the height of the subtree of T rooted at p.

Proof. Let us consider a merging string s_n for $\{n, \varepsilon\}$ of minimal length (it exists because C is synchronizing, but it need not be unique). The string s_n brings both nodes to the root, because otherwise we could remove the last letter of s_n and the result would still merge n and ε.

Let $\{n_i, m_i\}$ be the unordered pairs of nodes that appear when consecutive prefixes of s_n are applied to the initial set $\{n, \varepsilon\}$, i.e.

$$\{n_i, m_i\} = \delta^* \left(\{n, \varepsilon\}, s_n[0..i)\right), \quad i = 0, \ldots, |s_n|. \tag{2}$$

We have $\{n_0, m_0\} = \{n, \varepsilon\}$ and $\{n_{|s_n|}, m_{|s_n|}\} = \{\varepsilon\}$ (a singleton is also considered a pair).

Let us look at the subsequence $\{n_{i_k}, \varepsilon\}, k = 0, \ldots, l$, of this sequence formed of pairs containing the root. Each node p, appears in this subsequence as the partner of ε at most once, because pairs do not repeat in $\{n_i, m_i\}$ (otherwise we could shorten the string s_n). The string $s_n[i_k, i_{k+1})$, that brings $\{n_{i_k}, \varepsilon\}$ to $\{n_{i_{k+1}}, \varepsilon\}$, is a string that either brings the node n_{i_k} to a leaf without loops or that brings ε to a leaf without loops. In either case the length of $s_n[i_k, i_{k+1})$ is at most $h_{n_{i_k}}$ (note that in the second case the node $n_{i_{k+1}}$ is in the subtree of n_{i_k}). We get

$$|s_n| = \sum_{k=0}^{l-1} |s_n[i_k, i_{k+1})| \le \sum_{k=0}^{l-1} h_{n_{i_k}} \le \sum_{p \in Q(T) \setminus \{\varepsilon\}} h_p. \tag{3}$$

The value of h_ε is not counted because the set $\{\varepsilon\}$ appears only as the last element of the sequence $\{n_{i_k}, \varepsilon\}$. □

Let H_T be the value of the bound in Theorem 5. H_T is the sum of heights of all the nontrivial subtrees of T apart from the whole tree. We will compare H_T with Π_T — the sum of depths of all the internal nodes, and with W_T — the sum of depths of all the leaves of T (that is the sum of codewords' lengths).

Lemma 6. *Let T be a complete binary tree, let $Q(T)$ be the set of internal nodes of T, let $L(T)$ be the set of leaves of T, let h_n and N_n be, respectively, the height and the number of leaves of the subtree rooted at the node n of T, let $|\pi(n)|$ be the distance from the root to n and let N_T be the number of leaves of T. Let us define*

$$H_T = \sum_{n \in Q(T) \setminus \{\varepsilon\}} h_n, \qquad \Pi_T = \sum_{n \in Q(T)} |\pi(n)|, \tag{4}$$

$$W_T = \sum_{n \in L(T)} |\pi(n)|, \qquad S_T = \sum_{n \in Q(T)} N_n. \tag{5}$$

Then the following holds:

$$H_T \le \Pi_T = W_T - 2N_T + 2 \le W_T = S_T - N_T. \tag{6}$$

Corollary 7. *Let w_i be codewords of a Huffman code. Then*

$$|s_n| \le \sum_i |w_i| \qquad \textit{and} \qquad |s_n| \le (N-2)(h-1). \tag{7}$$

The result of Theorem 5 can be improved if we notice that the sequence $\{n_{i_k}, \varepsilon\}$, defined in the proof of Theorem 5, cannot contain two nodes n_{i_k} and $n_{i_{k'}}$ that are roots of identical subtrees of T. Indeed, otherwise we could shorten the string s_n in the same way as before. This gives the following result.

Corollary 8. *The bound of Theorem 5 can be improved to:*

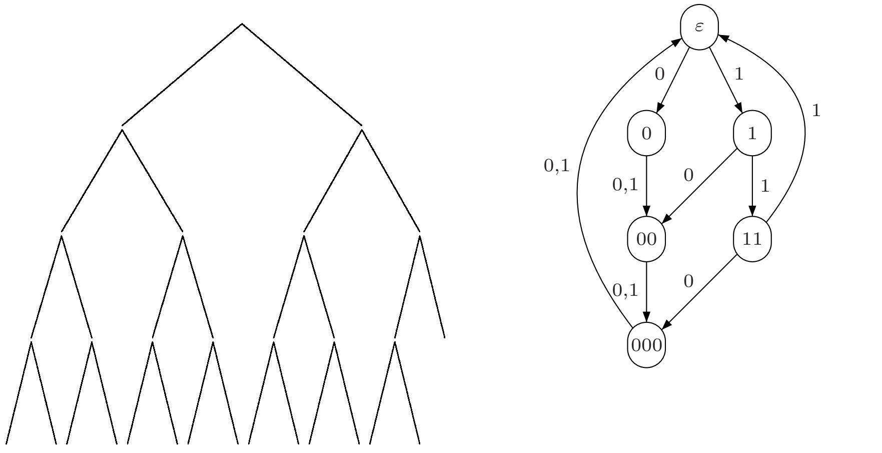
$$|s_n| \leq \sum_{t \in \mathscr{T}(T) \setminus \{T\}} h_t \tag{8}$$

where $\mathscr{T}(T)$ is the set all distinct subtrees of T.

The idea of identifying common subtrees can be formalized by introducing a *minimized Huffman automaton.* Although this does not give here a better estimate on the length of the shortest merging string for the set $\{n, \varepsilon\}$, it is interesting in itself.

Definition 9. *A* minimized Huffman automaton *for a Huffman code C is an automaton made of the Huffman automaton for C by merging the states that are roots of identical subtrees of the Huffman tree T for C.*

It is easy to see that minimized Huffman automata have exactly two edges, labeled with 0 and 1, going out of each node. An example of a minimized Huffman automaton is presented in Fig. 1.



(a) Huffman tree (b) Minimized Huffman automaton

Fig. 1. A Huffman tree and its minimized Huffman automaton

We will say that a set V of states of $\mathcal{T}$ corresponds to the set V_m of states of the minimized Huffman automaton $\mathcal{T}_m$ if V_m is the smallest set satisfying: if $q \in V$ and q is merged to a state q' of $\mathcal{T}_m$ then $q' \in V_m$.

Theorem 10. *Let C be a synchronizing Huffman code, let $\mathcal{T}$ be the Huffman automaton for C and let $\mathcal{T}_m$ be the minimized Huffman automaton for C. Let V be a set of states of $\mathcal{T}$ and V_m the corresponding set of states of $\mathcal{T}_m$. If s is a merging string for V then s is a merging string for V_m. If s' is a merging string for V_m that brings all nodes of V_m to the root then s' is a merging string for V.*

Note that the minimized Huffman automaton is implicitly used in Corollary 8, where we consider all non-identical subtrees of a Huffman tree. The roots of such subtrees are the states of the minimized Huffman automaton.

Theorem 5 leads to an algorithm for finding the shortest merging string for a set $\{n_0, \varepsilon\}$, where n_0 is any state of $\mathcal{T}$. First a graph $G = (V, E)$ is created. The vertices of G are unordered pairs $\{n, \varepsilon\}$, where n is a state of $\mathcal{T}$. The edges of G are weighted; $\{n_1, \varepsilon\} \to \{n_2, \varepsilon\}$ is an edge if there is a string w that brings $\{n_1, \varepsilon\}$ to $\{n_2, \varepsilon\}$ without passing through any other pair $\{n, \varepsilon\}$. The weight of the edge is the length of the shortest such string w (note that the string w need not be unique).

Such a string w will be the label of the edge $\{n_1, \varepsilon\} \to \{n_2, \varepsilon\}$, although it will not be stored explicitly. Instead, for retrieving the label w, we will store a mark M. The mark will depend on the target pair of the edge. If the target is a pair $\{n_2, \varepsilon\}$ with $n_2 \neq \varepsilon$, the mark is equal to either n_1 if $n_2 = \delta(n_1, w)$, or to ε if $n_2 = \delta(\varepsilon, w)$. We always have $n_2 = \delta(M, w)$. The node n_2 is in the subtree of the node M and w is formed of labels on the path from M to n_2. If the target of an edge is a singleton $\{\varepsilon\}$, that is $n_2 = \varepsilon$, the mark M is the leaf $\delta(\varepsilon, w)$. In this case the word w is formed of labels on the path from ε to M. In either case the label w can be recovered in $O(|w|)$.

The construction of the graph requires DFS-traversing the Huffman tree with a pair of nodes $\{n_1, n_2\}$, starting at $\{n, \varepsilon\}$ and applying transitions of the Huffman automaton to both nodes of the pair. The traversing goes forward until a set $\{n', m\}$ is reached, with m being a leaf. Then the edge $\{n, \varepsilon\} \to \{n', \varepsilon\}$ is added to the graph with the number of steps from $\{n, \varepsilon\}$ to $\{n', m\}$ as its weight. If such an edge has been added before, only the weight is updated to be the minimum of the previous weight and the new one. Finally, the mark M of the edge is set appropriately.

The cost of processing each pair $\{n, \varepsilon\}$ during the construction of the graph G is proportional to the size of the subtree rooted at n, because the DFS search is limited to the subtree of n. It follows that the construction of G uses the time proportional to the sum of sizes of the subtrees of T. By Lemma 6 this is $O(\sum |w_i|)$, where w_i are the codewords given by the tree T. The number of vertices in the graph is $|V| = N - 1$. The number of edges is bounded by the sum of sizes of all the subtrees of the tree, that is $|E| = O(\sum |w_i|)$.

The shortest merging string for a set $\{n, \varepsilon\}$ is given by the lightest path from $\{n, \varepsilon\}$ to $\{\varepsilon\}$. The tree of the lightest paths from any node to $\{\varepsilon\}$ can be constructed using the Dijkstra's algorithm in $O(|E| + |V| \log |V|)$. Since $|V| = O(N)$, $|E| = O(\sum |w_i|)$ and $\sum |w_i| \geq N \log N$, the lightest paths' tree can be computed in $O(\sum |w_i|)$.

Theorem 11. *Let $\mathcal{T}$ be a Huffman automaton. The algorithm for computing the shortest merging string for a set $\{n, \varepsilon\}$, where n is any state of $\mathcal{T}$, requires preprocessing time $O(\sum_i |w_i|)$. Then the shortest merging string for each pair $\{n, \varepsilon\}$ can be found in the time proportional to the length of the merging string.*

4 Length of a Synchronizing String

In this section we give an upper bound on the length of the shortest synchronizing string for any synchronizing Huffman code. We begin with a lemma that helps to prove the main theorem of this section (Theorem 13).

Lemma 12. *Let T be a complete binary tree with N leaves. There exists a string w of length at most $\lceil \log N \rceil$ such that for each node n of T some prefix of w labels a path from n to a leaf.*

Theorem 13. *For any synchronizing Huffman code of size N the length of the shortest synchronizing string s is at most*

$$|s| \leq \lceil \log N \rceil + (\lceil \log N \rceil - 1)X = O(Nh \log N) \tag{9}$$

where h is the length of the longest codeword,

$$X = \sum_{t \in \mathscr{T}(T) \setminus \{T\}} h_t, \tag{10}$$

$\mathscr{T}(\mathcal{T})$ is the set all different subtrees of T and h_t is the height of the subtree t.

Proof (sketch). We first find a string w given by Lemma 12 and apply it to coins on all states of $\mathcal{T}$. It reduces the number of coins to at most $\lceil \log N \rceil$, because the final position of each coin is determined by some proper suffix of w. We may assume that one of the coins is on the root of $\mathcal{T}$ (otherwise w could be shortened). Then, we pick a node n with a coin and we construct the shortest string s_n that merges n and the root (to get shorter synchronizing strings it is better to pick a node n with the shortest merging string for $\{n, \varepsilon\}$ among the nodes with a coin). By Corollary 8, $|s_n| \leq X$. This string applied the current set reduces the number of coins by at least one. Repeating this procedure additional $(\lceil \log N \rceil - 2)$ times leaves us with just one coin. The length of the merging string does not exceed $\lceil \log N \rceil + (\lceil \log N \rceil - 1)X$. Finally, $X < Nh$ gives the asymptotic bound $O(Nh \log N)$. □

The proof of Theorem 13 is constructive and gives an algorithm for the construction of a synchronizing string for a Huffman code. The algorithm works as follows.

First a string w from Lemma 12 is found. It is done by checking all the $O(N)$ strings of length less or equal $\lceil \log N \rceil$ in the following way. From each node n of T we traverse the subtree of n with DFS. Each time we are in a node m that is $l \leq \lceil \log N \rceil$ steps below n, we mark the string w on the path from n to m as *bad*. This means that no prefix of w brings n to a leaf.

After the traversal, the strings that have not been marked as bad bring any node through a leaf. By Lemma 12, there is at least one such a string of length $\lceil \log N \rceil$ or less. The cost of this algorithm is proportional to the sum of sizes of all subtrees of T, which is $O(\sum_i |w_i|)$.

After finding the string w we may apply it to the set of all internal nodes of T. This will take $O(N \log N)$ time. Then, at most $\log N$ merging strings for $\{n, \varepsilon\}$,

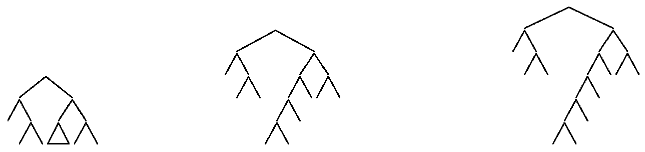


Fig. 2. The class of trees with the longest synchronizing string for a given number of nodes n. s denotes the synchronizing string for each tree. The triangle denotes a code $\{1, 01, 001, \dots, 0^i1, 0^i0\}$, $i \geq 0$.

with some n, suffice to build a synchronizing string. Computing the merging strings require preprocessing time $O(\sum_i |w_i|)$ and then any string can be read in the time proportional to its length. The length of each merging string is bounded by X and there are at most $\log N$ vertices that have to be moved using each such string. Thus the total cost of the algorithm is $O(X \log^2 N + \sum_i |w_i|)$.

5 Experimental Tests

Tests were performed to find the worst-case trees for the length of the shortest synchronizing string and the worst-case trees for the length of the shortest merging strings for a pair $\{n, \varepsilon\}$, where n is an internal node of the tree. All trees of sizes, N, from 3 to 20 were analyzed first. Then the procedure was repeated for all trees of heights, h, from 2 to 5.

5.1 Long Synchronizing String

In most of the tested cases the worst-case trees for fixed size, N, were unique up to the reflection across the y axis (relabeling 0-edges to 1-edges and 1-edges to 0-edges). The exceptions were the trees with 7 nodes — three nonequivalent trees, 10 nodes — 5 trees, and 12 nodes — 2 trees. For trees with 9, 11 and 13-20 nodes the unique worst-case tree corresponds to one of the codes C_k, given below. The codes C_k also form one of the worst-case trees with 7, 10 and 12 nodes.

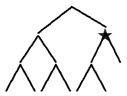
$$C_k = \{00, 010, 011, 110, 111\} \cup \{10^i1 | i = 1, 2, \dots, k-1\} \cup \{10^k\}, \quad k \geq 1. \quad (11)$$

The size of the code C_k is $k+5$. The structure of these trees is shown in Fig. 2(a) and examples can be found in Figs. 2(b) and 2(c).

Theorem 14. *The shortest synchronizing string for the tree C_k, $k \geq 1$, is $s_0 = 0^k10^k$ for odd k (even number of codewords) and $s_1 = 0^k1010^k$ or $s_2 = 0^k1110^k$ for even k (odd number of codewords). The length of the shortest synchronizing string is $2N-9$ for even code size, N, and $2N-7$ for odd code size.*

The worst-case trees for fixed height h, with $h = 2, 3, 4$ and 5, are the trees given by the set of codewords

$$D_h = \left(\{0,1\}^h \setminus \{1^{h-1}1, 1^{h-1}0\}\right) \cup \{1^{h-1}\}, \quad h \geq 2. \quad (12)$$


(a) h=3 , s=$1^2 0^3 1^2$

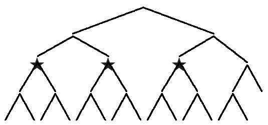

(b) h=4 , s=$1^3 0^4 1^3 0^4 1^3$

Fig. 3. Trees with the worst-case length of a synchronizing and merging string among trees of fixed height h, for $h = 2, 3, 4, 5$ and a scheme of these trees. The nodes n with the longest merging string for $\{n, \varepsilon\}$ are marked with a star.

These are full binary trees with two edges in the lower-right corner removed. The number of codewords in the code D_h is $2^h - 1$. They are unique worst-case trees up to the reflection across the y axis. The trees D_3 and D_4 are shown in Fig. 3(a) and Fig. 3(b).

Theorem 15. *The shortest synchronizing string for the tree D_h, $h \geq 2$, is $s = (1^{h-1}0^h)^{h-2}1^{h-1}$ with $|s| = 2h^2 - 4h + 1$ (however, the shortest synchronizing string is not unique).*

The minimized Huffman automaton for D_h has $K = 2(h-1)$ nodes. Even though it contains a letter that reduces the number of coins by $h - 2 = \frac{K}{2} - 1$ (a letter of deficiency $\frac{K}{2} - 1$), its shortest synchronizing string is of length $2h^2 - 4h + 1 = \frac{K^2}{2} - 1$, which is quadratic in K. This makes the automata D_h interesting in themselves.

The results of the search allow us to state the following conjecture.

Conjecture 16. The length of the shortest synchronizing string s for a code of N codewords, $N \geq 9$, with h being the length of the longest codeword, is at most:

$$|s| \leq \min(2N - a, 2h^2 - 4h + 1), \tag{13}$$

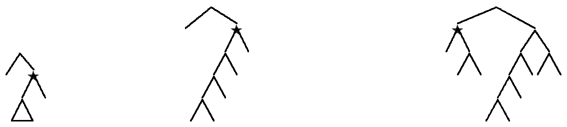
where a is 7 for odd N and 9 for even N.

5.2 Long Merging String

For trees of fixed size N the length of the shortest merging string in the worst case is equal $N - 2$, for $N = 3, \ldots, 20$, apart from $N = 6$. For $N = 6$ the worst-case length is equal $N - 1 = 5$. Two families of trees have the worst-case shortest merging strings. The first one corresponds to the code

$$G_k = \{0, 10^k\} \cup \{10^i 1 | i < k\}, \quad k \geq 1, \tag{14}$$

and gives the worst-case trees for N from 3 to 20, apart from $N = 6$. The size of the tree G_k is $N = k + 2$. The merging string of the set $\{1, \varepsilon\}$ is of length $N - 2$. The structure of these trees is shown in Fig. 4(a) and the tree G_4 is shown in Fig. 4(b). The latter figure also shows the node whose merging string with ε is the longest.



(a) A tree G_k (b) G_4, $s = 0^4$ (c) C_4, $s = 1110^4$ or $s = 1010^4$

Fig. 4. The nodes with the longest merging string for the two families C_k and G_k

The other family of trees is the family C_k (see (11) and Fig. 2(a)) with even k (odd number of codewords). The merging string for $\{0, \varepsilon\}$ is of length $N-2$ and this is the worst case for $N = 7, 9, 11, \ldots 19$. The node with the longest merging string and the merging string itself for the tree C_4 are shown in Fig. 4(c).

There were also additional worst-case trees found for $N = 5, 6, 7, 9, 10, 12$. These do not correspond to neither the trees C_k nor G_k.

The worst-case trees among trees of fixed height are the trees D_h (Equation (12) and Fig. 3).

Theorem 17. *The upper bound on the length of the shortest merging string for any pair $\{n, \varepsilon\}$, where n is a state of D_h, is $\lceil h^2 - \frac{3}{2}h \rceil$. For odd h it is achieved by the pair $\{0^{(h-1)/2}, \varepsilon\}$. For even h it is achieved by pairs $\{x, \varepsilon\}$, where x is any binary string of length $\frac{h}{2}$ containing at least one 0.*

The results of the search allow us to state the following conjecture.

Conjecture 18. For any Huffman automaton $\mathcal{T}$ corresponding to a code with N codewords, with h being the length of the longest codeword, the length of the shortest merging string s_n for a set $\{n, \varepsilon\}$, where n is any state of $\mathcal{T}$ is at most:

$$|s| \leq \min(2N - 2, \lceil h^2 - \tfrac{3}{2}h \rceil), \tag{15}$$

if $N \neq 6$, and $|s| \leq 5$ for $N = 6$.

6 Summary

We presented a constructive upper bound on the length of the shortest merging string and the shortest synchronizing string for a Huffman code.

We tested the lengths of the shortest merging and synchronizing string on all codes of size from 3 to 20 and on all codes with the length of the longest codeword from 2 to 5. Three classes of worst-case codes were found. The length of the shortest synchronizing strings for these classes of codes is far from the bound proven before. This allowed us to formulate conjectures, which remain open.

Acknowledgement

The author wishes to thank dr Wojciech Plandowski of Institute of Informatics, University of Warsaw, Poland for useful discussions. Dr Plandowski improved the bounds in Theorems 5 and 13, originally found by the author, to the ones presented here.

References

1. Capocelli, R.M., Gargano, L., Vaccaro, U.: On the characterization of statistically synchronizable variable-length codes. IEEE Trans. Inform. Theory 34(4), 817–825 (1988)
2. Freiling, C.F., Jungreis, D.S., Theberge, F., Zeger, K.: Almost all complete binary prefix codes have a self-synchronizing string. IEEE Trans. Inform. Theory 49(9), 2219–2225 (2003)
3. Schützenberger, M.P.: On synchronizing prefix codes. Information and Control 11(4), 396–401 (1967)
4. Rudner, B.: Construction of minimum-redundancy codes with an optimum synchronizing property. IEEE Trans. Inform. Theory 17(4), 478–487 (1971)
5. Perkins, S., Escott, A.E.: Synchronizing codewords of q-ary Huffman codes. Discrete Math. 197-198, 637–655 (1999)
6. Huang, Y.M., Wu, S.C.: Shortest synchronizing codewords of a binary Huffman equivalent code. In: ITCC 2003: Proceedings of the International Conference on Information Technology: Computers and Communications, Washington, DC, USA, p. 226. IEEE Computer Society Press, Los Alamitos (2003)
7. Capocelli, R.M., Santis, A.D., Gargano, L., Vaccaro, U.: On the construction of statistically synchronizable codes. IEEE Trans. Inform. Theory 38(2), 407–414 (1992)
8. Ferguson, T.J., Rabinowitz, J.H.: Self-synchronizing Huffman codes. IEEE Trans. Inform. Theory 30(4), 687–693 (1984)
9. Maxted, J.C., Robinson, J.P.: Error recovery for variable length codes. IEEE Trans. Inform. Theory 31(6), 794–801 (1985)
10. Černý, J.: Poznámka k. homogénnym experimentom s konecnými automatmi. Mat. fyz.cas SAV 14, 208–215 (1964)
11. Ananichev, D.S., Volkov, M.V.: Synchronizing generalized monotonic automata. Theor. Comput. Sci. 330(1), 3–13 (2005)
12. Kari, J.: Synchronizing finite automata on eulerian digraphs. Theor. Comput. Sci. 295(1-3), 223–232 (2003)
13. Pin, J.E.: On two combinatorial problems arising from automata theory. Annals of Discrete Mathematics 17, 535–548 (1983)
14. Ananichev, D.S., Volkov, M.V., Zaks, Y.I.: Synchronizing automata with a letter of deficiency 2. Theor. Comput. Sci. 376(1-2), 30–41 (2007)
15. Trahtman, A.N.: An efficient algorithm finds noticeable trends and examples concerning the Černý conjecture. In: Kralovic, R., Urzyczyn, P. (eds.) MFCS 2006. LNCS, vol. 4162, pp. 789–800. Springer, Heidelberg (2006)
16. Eppstein, D.: Reset sequences for monotonic automata. SIAM J. Comput. 19(3), 500–510 (1990)
17. Sandberg, S.: Homing and Synchronizing Sequences. In: Broy, M., Jonsson, B., Katoen, J.P., Leucker, M., Pretschner, A. (eds.) Model-Based Testing of Reactive Systems. LNCS, vol. 3472, pp. 5–33. Springer, Heidelberg (2004)