A New Algorithm for Generating all Binary Huffman Codes Based on Path-Length Extensions

Othmane Niyaoui
Telecommunication engineer of the National Institute of Telecommunications in Rabat (INPT), Morocco.

Oussama Mohamed Reda
Laboratoire de recherche en informatique (LRI)
Department of computer science, Faculty of Sciences of Rabat, Mohammed V University in Rabat, Morocco.

Abstract
Huffman codes (HC) constitute a crucial part of lossless data compression. Using methods such as Huffman's algorithm allows generation of optimal encoding schemes, in the sense that the average codeword length is minimum. Knowing all binary Huffman codes promises to be essential in developing many adaptive-based approaches in the field of universal lossless data compression. In this paper, we develop a recursive algorithm to generate all possible binary HC. We first define those binary Huffman codes, give some basic properties and develop the key concept named path-length extension. Based on this concept we then show that path-length sequences of binary HC can be generated from a trivial sequence in a determined number of steps. Motivated by this later result, we then address and prove our new algorithm.

INTRODUCTION
In the last decades, there has been a growing interest in data compression as the amount of data processed in applications and transmitted over the internet has reached a tsunami scale. For many kinds of applications where data compression is a key component, lossless compression becomes mandatory as data loss is not acceptable. The aim of data compression is to encode as efficiently as possible a source of data. A discrete data source is a set of randomly delivered source symbols from an alphabet. Discrete data sources are of two kinds, namely, memoryless sources and sources with memory which are usually modeled as markov chains and known as markovian sources.

In contrast to markovian sources, discrete memoryless sources (DMS) have the property that its outputs at different times do not depend on each other. For a discrete memoryless source with given probabilities on the source symbols, a simple way to achieve good compression is by mapping more probable symbols into shorter bit sequences, and less likely symbols into longer bit sequences called codewords. The necessity to concatenate successive variable-length codewords leads to the concept of unique decodability. In fact, any method of encoding symbols as variable-length bit sequences may be acceptable in practice if it is always possible to uniquely decode the source symbols from the encoded string of binary digits.

To determine where the code for one symbol ends and the code for the next one begins, a sufficient condition, is that each codeword must not to be a prefix of another codeword; such codes are called binary prefix free codes and they form a simple class of uniquely decodable codes. In 1948, Shannon has proved [1] that the expected length of uniquely decodable codes is lower bounded by its entropy which represents a measure of the least amount of bits necessary for a data source to encode its symbols without loss. Since this work, significant statistics-driven contributions on lossless data compression whose efficiency approaches Shannon's limit have been produced [2]. David Huffman has proposed a simple algorithm for constructing the optimal variable-length prefix free codes i.e Huffman codes [3]. Since its inception, Huffman algorithm has been the subject of intensive research in data compression. It has inspired a large literature with diverse theoretical and practical contributions [4]. Although the Huffman coding is robust, it was found to be far from optimal in real applications. Indeed, Huffman algorithm inputs the probabilities of source symbols to find the optimal prefix free code (in the sense of minimum expected length). However, this is impossible from a practical standpoint since the probability distribution of source symbols cannot be known in advance. In addition, the Huffman algorithm cannot take into consideration other constraints (variance, maximum length, cost, etc.). Finding all binary HC can be an alternative to this issue. In fact, having all possible binary HC, we can start with a random code to reach an optimal solution using an adaptive approach whichever the constraints of the application.

The present paper deals essentially with the generation of binary Huffman codes without addressing their use in lossless data compression. Research has previously been done on algorithms for generating all possible HC as well as trying to count there number. Indeed, in the early days, Norwood has proposed a recursive method to count the number of all possible binary HC [5]. She showed that this number increases in a predictable fashion as the code length increases. More recently, Elshotz and al. [6] have proved a precise asymptotic result on the counting function of the number of all binary HC.

In [7] Even and Lempel have constructed an algorithm for generating and enumerating all binary HC by solving the characteristic sum condition. Authors in [8] have showed that
the construction of binary HC is a discrete convex optimization problem over a lattice of binary trees. Dean Huffman and al. have proposed an algorithm for generating all n-ary HC based on a numerical test for membership in the set of Kraft sequences [9]. Finally, Khosravifard and al. [10] have constructed a tree-based algorithm to generate all possible binary HC proving new constraints on the set of codeword lengths. This paper develops a new algorithm to generate all binary HC. Our motivation came from the analysis of the path length sequence associated with binary HC. Our objective is to give a structural characterization of binary full HC. The output of the presented algorithm is theoretically the same as the approaches referenced above. However, we present a much more intuitive approach based on a new characterization of the binary HC path-length sequences. We show how to generate all codes of a given order from a trivial sequence using a determined number of iterations.

The organization of the paper is as follows. Section II recalls HC definition and presents the necessary prerequisite to present and develop our work. Section III develops the main contribution of the paper, namely, the path-length extension concept and the main theorem on which our new algorithm is based. Section IV addresses the algorithm and finally the paper is concluded in section V.

**HUFFMAN CODES - A BRIEF REVIEW**

**A. Overview**

Let a discrete memoryless source with alphabet \( \{ a_1, ..., a_n \} \), a binary HC associates with each source alphabet \( a_i \) a codeword \( w_i \). The number of bits in \( w_i \) is called the length \( l_i \). As mentioned in the introduction, a HC has the property that each codeword cannot be a prefix of another codeword; for example, the binary code \( \{ "1", "00", "01" \} \) is a binary Huffman codes with lengths \( (1, 2, 2) \).

Binary HC can be represented as binary trees which grow from a root to leaves representing codewords. Each branch is labelled 0 or 1 and each final node (leaf node) represents the binary codeword. The tree is extended just enough to include each codeword (see Figure 1).

**Figure 1:** The binary tree corresponding to the code \( \{ "1", "00", "01" \} \)

For a certain set of probabilities \( \{ p_1, ..., p_n \} \) on the source symbols, the Huffman algorithm provides a simple procedure for finding a binary optimal variable-length full prefix tree codes. The algorithm can be described as follows:

**Step 1** Let \( L \) be the list of the probabilities of the source symbols;

**Step 2** Take the two symbols with the smallest probabilities in \( L \) and create a new node as being a parent of the nodes associated with them;

**Step 3** Replace the above two probabilities in \( L \) with their sum associated with the new intermediate node. If the new list contains one element, stop, otherwise return to **Step 2**.

The tree is then traversed to determine the codewords of the symbols. The Huffman process is best illustrated by an example.

**Figure 2:** Huffman algorithm illustration

Given five symbols with probabilities as shown in Figure 2, they are processed in the following order:

1. \( a_4 \) is combined with \( a_5 \) and both are replaced by the combined symbol \( a_{45} \) whose probability is 0.25;
2. There is now four symbols left, we select \( a_3 \) and \( a_{45} \) combine them, and replace them with the auxiliary symbol \( a_{345} \), whose probability is 0.45;
3. Three symbols are now left, \( a_1 \), \( a_2 \), and \( a_{345} \); the least probable symbols \( a_1 \) and \( a_2 \) are then combined and replaced by \( a_{12} \), whose probability is 0.55;
4. Finally, we combine the two remaining symbols, the tree is now complete;

The obtained HC codewords are: \( \{ 11, 10, 00, 011, 010 \} \)

**B. Path-length sequence associated with a binary HC**

It is well known that binary Huffman codes obey to Kraft equality [11]. In fact, If \( (l_1, ..., l_n) \) is the path-length sequence associated with a binary Huffman codes, then \( \sum_{i=1}^{n} 2^{-l_i} = 1 \). Conversely, for any set \( (l_1, ..., l_n) \) of lengths satisfying the Kraft equality, there is a simple procedure to construct a HC with those lengths [2]. Thus, binary trees, path-length sequences, and binary codewords are equivalent representations of binary HC. Consequently, path-length sequences will be used to represent binary HC for the rest of the paper.

Without loss of generality, the path-length sequence of a binary HC will be set in an ascending order i.e. \( l_1 \leq l_2 \leq ... \leq l_n \).
\[
\ldots l_{n-1} \leq l_n. \] This leads us to the following definition of binary Huffman codes' path-length sequences:

Without loss of generality, the path-length sequence of a binary HC will be set in an ascending order i.e. \( l_1 \leq l_2 \leq \ldots \leq l_{n-1} \leq l_n. \) This leads us to the following definition of binary Huffman codes' path-length sequences:

**Definition 1 (path-length sequence of binary Huffman codes)** Let \( n \geq 2 \) be a positive integer, we say that \( (l_1, \ldots, l_n) \in \mathbb{N}^n \) is a path-length sequence of a binary HC of order \( n \) if the following two conditions are satisfied:

1. \( l_1 \leq l_2 \leq \ldots \leq l_{n-1} \leq l_n \)
2. \( \sum_{i=1}^{n} 2^{-l_i} = 1 \)

C. Properties of path-length sequences associated with binary Huffman codes

In order to simplify the proof given in section IV, we first develop and prove the properties below which constitute the needed results to prove our theorem.

**Properties**

1. Every path-length sequence of a full binary HC has an even number of its largest elements
2. If \( (l_1, \ldots, l_n) \) is a path-length sequence of a binary HC then \( l_n = l_{n-1}; \)
3. For any path-length sequence of a binary HC \( (l_1, \ldots, l_n) \) we have \( 2^{l_1} \leq n; \)
4. The only path-length sequence of a binary HC of order 3 is \((1, 2, 2); \)
5. Two trivial \( n \)-path-length sequences of a binary HC are: \((1, 2, \ldots, n - 2, n - 1, n - 1)\) and \((n, \ldots, n).\)
6. Let \( l_1 \in \mathbb{N}^n \), the only \( 2^{l_1} \) path-length sequences of a binary HC is the trivial sequence \((l_1, \ldots, l_1)\)

**Proofs:**

1. See [4] theorem 3.1;
2. A direct result of 1;
3. We have \( \forall i \in \{1, \ldots, n\}; \quad 2^{-l_i} < 2^{-l_1}, \) so \( \sum_{i=1}^{n} 2^{-l_i} \leq n \) then \( n > 2^{l_1}; \)
4. We have by definition 1 \( \left\{ 0 \leq l_1 \leq l_2 = l_3 \right\} \)
   \[ 2^{-l_1} + 2^{-l_2} + 2^{-l_3} = 1 \]
   Since by Property 3 \( 2^{l_1} < 3, \) then \( l_1 = 1 \), therefore \( l_2 = l_3 = 2; \)
5. Easily proven and it’s omitted;
6. Let \( l_1 \in \mathbb{N}^n \), suppose that \( (l_1, \ldots, l_n) \) is a path-length sequence of a binary HC of length \( n = 2^{l_1}; \) then \( \sum_{i=2}^{n} 2^{-l_i} = \frac{n-1}{n}\). Suppose that \( l_1 \neq l_2 \), therefore \( l_1 < l_2 \leq \ldots \leq l_n; \)
   i.e. \( \forall i \in \{2, \ldots, n\}; \quad 2^{-l_i} < 2^{-l_1}, \) then \( \sum_{i=2}^{n} 2^{-l_i} < \frac{n-1}{n}, \) and this is impossible, so we conclude that \( l_1 = l_2 \). The same demonstration can be done to show that \( l_2 = l_3 = \ldots = l_n = l_1 \) and so forth.

**PATH-LENGTH EXTENSION**

Let \( (l_1, \ldots, l_n) \in \mathbb{N}^n \) be a path-length sequence of a binary HC of order \( n \). The path-length extension arises from the conservation of the Kraft-condition sum when we split one element \( l_k \) into two elements of the same value \( l_k + 1 \) as illustrated in Figure 3 i.e. \( \sum_{i=1}^{n} 2^{-l_i} = \sum_{i=1}^{n-1} 2^{-l_i} = 1. \)

\[
\begin{align*}
(l_1', \ldots, l_{k-1}', l_k, l_{k+1}', \ldots, l_n) \\
(l_1', \ldots, l_{n+1}') = (l_1', \ldots, l_{k-1}', l_k + 1, l_k + 1, l_{k+1}', \ldots, l_n)
\end{align*}
\]

*Figure 3*: Kraft sum conservation after extension

Moreover, the obtained path-length sequence \((l_1', \ldots, l_{n+1}')\) verifies the order condition only if the path-length extension is applied on the right elements (i.e. indices) of the path-length sequence. This require that \( l_{k+1} \leq l_{k+2} \), hence \( l_{k+1} + 1 \leq l_{k+1} \) i.e. \( l_k < l_{k+1} \). This leads us to first define the extension set presented below.

**Definition 2 (Extension set)** the extension set associated with a binary HC path-length sequence of order \( n \) is given by \( \{i \in \{1, \ldots, n\}; l_i < l_{i+1}\} \cup \{n\} \)

Note that the extension set can never be empty and it always contain the last element index \( n \). Note also, that if the extension set contain just one element, then \( l_1 \leq l_2 = \ldots = l_n \).

**Proposition and Definition (Path-length extension)** let \( (l_1, \ldots, l_n) \in \mathbb{N}^n \) be a path-length sequence of a binary HC of order \( n \) and \( k \) be any element of the extension set, then the sequence \((l_1', \ldots, l_{n+1}') = E_k(l_1, \ldots, l_n)\) defined below is a path-length sequence of a binary HC of order \( n + 1; \)

\[
\begin{align*}
(l_1', \ldots, l_{k-1}') &= (l_1, \ldots, l_{k-1}) \\
l_k' &= l_{k+1}' = l_k + 1 \\
l_{k+1}', \ldots, l_{n+1}' &= (l_{k+1}, \ldots, l_n)
\end{align*}
\]

*Note* that \( E_k \) is defined as the path-length extension applied in the \( k^{th} \) index.

**Proof**: Easily proven and it’s omitted

The path-length extension operation can be illustrated as follows. Starting with the first sequence \((1, 1)\), the associated extension set contains only one element, so by applying the path-length extension on the underlined element, we obtain the sequence \((1, 2, 2)\). The same operation can be repeated for the new sequence to obtain the sequence \((1, 2, 3, 3)\). Now, the
extension set contains two elements, and the extension operation can be applied to each element of the extension set (underlined elements), so we obtain two sequences, \((1,3,3,3,3)\) by extending the first element, and \((1,2,3,4,4)\) by extending the second one.

So forth, by repeating indefinitely the extension process, we will obtain all binary HC path-length sequences whose first element is \(l_1\). The next section develops and proves this result. The choice to not include the first index element in the extension set definition is justified by the fact of keeping a print of the starting sequence so we know from where the sequence was generated, for instance the sequence \((1,1)\) .

The same thing can be done, if we start with the next trivial path-length sequence \((2,2,2,2)\) . Figure 4 illustrate these two examples.

![Figure 4: Path-length extension applied to the sequences \((1,1)\) and \((2,2,2,2)\)](image)

That said, the path-length extension concept is the building block of our theorem detailed in the next section, which is along with its derived algorithm are the core of the paper’s contribution.

**ALGORITHM**

The idea is the following. Starting from an initial trivial path-length sequence \((l_1, \ldots, l_1)\) of length \(2^{l_1}\), we can generate all binary Huffman codes' path-length sequences of length \(n \geq 2^{l_1}\). We prove that the number of necessary iterations is \(K = n - 2^{l_1}\). At each \(k^{th}\) step, we apply the path-length extension on the obtained sequences from the \(k - 1\) step on the elements of the extension set.

**Theorem (Binary HC generation)** Any path-length sequence of binary Huffman code \((l_1, \ldots, l_n)\) of length \(n \geq 3\) can be obtained from the trivial sequence \((l_1, \ldots, l_1)\) of length \(2^{l_1}\) in \(K = n - 2^{l_1}\) steps.

**Proof:** For \(n = 3\), the only binary Huffman path-length sequence of length 3 is \((1,2,2)\). Since we have \((1,2,2) = E_2(1,1)\), then the proposition is true for \(n = 3\). Let \(n > 3\), and assume that the proposition holds for all lengths less or equal than \(n\), and let show that this is true for length \(n + 1\). Let \((l_1, \ldots, l_{n+1})\) be a non-trivial sequence, then by property 2) we have \(l_n = l_{n+1}\). So \((l_1, \ldots, l_{n+1}) = E_n(l_1, \ldots, l_n - 1)\), the shrunked sequence \((l_1, \ldots, l_n - 1)\) is obviously a path-length sequence of a binary HC of length \(n\) so by our assumption it can be generated from a trivial sequence \((l_1, \ldots, l_1)\) in \(K\) steps, then \((l_1, \ldots, l_{n+1})\) is obtained in \(K + 1\) steps from \((l_1, \ldots, l_1)\).

For example the sequence \((1,2,3,4,5,6,6)\) is generated from the initial sequence \((1,1)\) in 5 steps as shown in Figure 5 below.

![Figure 5: Generation process of the sequence \((1,2,3,4,5,6,6)\)](image)

Note that according to the theorem above, if is a path-length sequence of a binary HC we have then twocases:
1. If \(n = 2^{l_1}\), in this case \((l_1, \ldots, l_n) = (l_1, \ldots, l_1)\) (according to property 6)
2. If \(n > 2^{l_1}\), the path-length sequence \((l_1, \ldots, l_n)\) is generated from the trivial sequence \((l_1, \ldots, l_1)\) in \(K = n - 2^{l_1}\) iteration.

The theorem above will be exploited to generate all possible path-length sequences of order \(n\). Starting from an initial sequence \((l_1, \ldots, l_1)\), we generate the sequence \((l_1, \ldots, l_1, l_1 + 1, l_1 + 1)\) of length \((2^{l_1} + 1)\). Note that the last element is the only element that is possible to extend in this case (according to the definition of the extension set). The operation of extending the obtained sequences is repeated until we reach the desired length \(n = 2^{l_1} + K\). However, this operation assumes that we know the sequence's first element \(l_1\), since the only input of our algorithm is \(n\) (the order of the path-length sequence).

The values of the sequences’ first elements are obtained from property 2), which imposes that \(2^{l_1} \leq n\). This restricts the first element to be contained in the set \(\{1, \ldots, \lfloor \log_2 n \rfloor\}\). Therefore, applying the already described method on each possible \(l\) value we obtain all path-length sequences of order \(n\).
Figure 6 illustrates the process and details different steps to obtain, respectively, all path-length sequences of order $n = 6$.

![Figure 6: Generation of all path-length sequences of binary HC of order 6](image)

We note that at the end of an iteration we can get redundant sequences. To avoid this we introduce a unicity test to our method (eliminated sequences in Fig 6 are dashed). The final algorithm is then addressed below.

Algorithm (Generating all binary HC)

Require $n > 3$

For $l_1 = 1$: $[\log_2(n)]$

1. Initializing the list $L = \{(l_1, \ldots, l_1)\}$
2. Determine the extension set $I$ of each sequence of $L$;
3. Apply the extension operator on the extension set elements, one element at a time;
4. Eliminate redundant sequences;
5. if the length of obtained sequences is equal to $n$ stop, other, replace $L$ by the obtained sequences and return to step 2
End

CONCLUSION

In this paper, a new recursive algorithm is formally developed to generate all binary Huffman codes. We give a new characterization based on their first element and their length. Indeed, we show that every path-length sequence of binary HC of length $n \geq 3$ starting with $l_1$ can be generated from the trivial sequence $(l_1, \ldots, l_1)$ in $K = n - 2^{l_1}$ steps using the path-length extension concept. The proposed algorithm has room for improvement in two directions. First, we can reduce the exponential growth of the size of the extension set since our algorithm is based on a naive and direct computing approach. Second, generation of redundant path-length sequences can be avoided by adopting a binary tree representation based-approach. Finally, the paper's results gives perspective of a practical application on universal lossless data compression and lays ground for development of a new adaptive coding scheme.

REFERENCES

[8] D. S. parker, P. Ram, *The construction of Huffman codes is a submodular optimization problem over a

