Variations on Theme by Huffman: A Literature Review

Cy Chan
cychan@mit.edu
May 10, 2006

1 Introduction

In this project report, the author reviews Gallager’s 1978 work showing four (then new) properties of Huffman codes [1]. The first property, the **sibling property**, is the property that each of the nodes in the Huffman code tree can be listed in non-increasing order of probability such that sibling nodes are adjacent. The second property established an upper bound on the redundancy of the Huffman code based on the probability of the most likely source symbol. This allows us in some cases to use a tighter upper bound on the expected code length when compared to the commonly known $H(X) + 1$ (where $X$ represents the random source). The third property shows that we can reserve a code word of length 2 and still have redundancy of at most 1. Finally, he describes an adaptive approach to Huffman coding which requires only a single pass and so can be done as data is provided rather than having to make a first pass to determine symbol frequency.

In addition to reviewing Gallager’s original paper, we give additional insight regarding the performance of adaptive Huffman coding in relation to another universal coding algorithm developed since: the LZ dictionary coding scheme. While Huffman coding is a fixed-to-variable length coding technique, LZ coding is a variable-to-fixed length coding technique.

2 The Sibling Property

Let the source symbols be $\{1, 2, \ldots, k\}$, and the probability of symbol $i$ be $P_i$ for each $1 \leq i \leq k$. Without loss of generality, assume $P_1 \geq P_2 \geq \ldots \geq P_k$. Let a binary code tree be a binary tree where each of the leaf nodes is assigned a probability and all of those probabilities sum to 1. Also, each interior node is assigned a probability equal to the sum of its children’s probabilities.

Gallager defines the **sibling property** to be the property where the nodes (terminal and interior, but excluding the root) of a binary code tree can be listed in non-increasing order of probability such that, in the list, all siblings are adjacent. He then argues that a code tree is a Huffman code tree iff it has the sibling property.

The forward direction can be seen by building an auxiliary stack as the Huffman tree is constructed. Every time two nodes are paired together in the Huffman algorithm, we add them to the top of the stack, the less probable node first. From this we see that all siblings will be adjacent because we push them onto the stack as a pair, and the resulting stack is in non-increasing order because after each push, the remaining nodes on the Huffman list are all larger than the two nodes that were just pushed.

The backward direction can be seen by using the sibling property list during construction of the Huffman tree. At any particular step, the bottom two nodes in the sibling list can be chosen by the Huffman algorithm because the list is in non-increasing order, and those nodes must be leaves since there are no nodes underneath them in the list. When the Huffman algorithm removes from its list and replaces them with their parent, we simply remove them from our sibling list because the parent (now a leaf) is already on the sibling list. Since the bottom two nodes of the new list are now leaves we can repeat this algorithm for all steps of the Huffman algorithm resulting in a true Huffman code tree.
3 Redundancy of Huffman Codes

The redundancy of the Huffman code is defined by Gallager to be the expected length of the code per source symbol minus the entropy of the source. The commonly known upper bound on the expected length is $H(X) + 1$ (upper bound on redundancy is 1), which can be shown by assigning each source symbol $i$ a length of $\lceil \log \frac{1}{P_i} \rceil$. Since these lengths satisfy the Kraft Inequality, there exists a uniquely decodable prefix code with those lengths, and since Huffman codes have been shown to be optimal, the Huffman code must also satisfy this inequality.

Gallager shows that in cases where the probability of the most probable source symbol $P_1$ is small, the redundancy can be much smaller than 1. The essence of his argument lies in two stages. First, he expresses the expected length and the entropy of the source as a sum of local functions over all interior nodes. He then splits the code tree into two parts: the first part contains all interior nodes down to the lowest level $l$ that is “full” (contains $2^l$ nodes), and second part contains all nodes below that. He bounds the contribution to the redundancy of the lower part by $P_1$ using simple inequality arguments, and he bounds the upper part by maximizing over all possible legal configurations of probabilities over those nodes. From this analysis, he concludes that the redundancy is at most $P_1 + \sigma$, where $\sigma = 1 - \log_2 e + \log_2(\log_2 e) \approx 0.086$. Further, for $P_1 \geq \frac{1}{2}$, he shows that the redundancy is at most $P_1$.

Using an intermediate result from the above proof, Gallager examines the case where we would like to reserve a length two codeword for protocol purposes. He does this by taking the original Huffman code tree and inserting a new node between the root and the less probable node in level 1. This lengthens each encoding for all symbols that branch off this node by 1, but it opens a new level 2 node for protocol use. Using the intermediate result mentioned above, he shows that this new code still has redundancy less than 1.

4 Adaptive Huffman Codes

4.1 Gallager’s Work

Static Huffman codes have been proven to be optimal over choices of fixed codes which do not change as the source produces symbols. However, they leave open the possibility that codes that change over the length of the data stream may be able to achieve better compression. Moreover, if the source statistics are not known in advance, the generation of the optimal static Huffman encoding of a given finite sequence requires a pre-pass to count the frequency of each symbol. Gallager’s idea of adaptive Huffman coding addresses this issue as well as provides a mechanism to adapt the code if the statistics of the source change over time.

First, in order to determine empirical source symbol probabilities, a counter is maintained for each symbol. Every time a source symbol is transmitted, its associated counter is incremented. Also, after every $N$ symbols all counters are multiplied by a scalar factor $\alpha < 1$, so that the frequency counts have “fading memory”. In other words, the symbol frequencies observed most recently have the most influence on the resulting Huffman code tree.

Second, Gallager describes a set of data structures that allows implementation of the adaptive Huffman coding method. He leverages the sibling property described in Section 2 in the design of his data structures, placing sibling pairs in a single unit to be moved around the tree together. He then describes how to update the pointers in the data structures whenever source symbol frequencies drift to the point that the code tree needs modification. Using the methods he proposes, the computational complexity of the encoding and decoding algorithms remain linear in the encoding length.

4.2 Analysis

4.2.1 Performance

It is important to realize that adaptive Huffman coding could result in a shorter overall encoding length than the optimal static Huffman encoding. This is because the adaptive nature allows the coding scheme to react
to changes in the source and adapt to use a coding scheme tailored to current statistics. An easy example
to illustrate this is a source that has two phases: during the first half of the sequence (assumed to be long
compared to the decay length $N$ of the frequency counts) the source outputs ‘0’ and ‘1’ with $\frac{1}{2}$ probability
each, and during the second half it outputs ‘2’ and ‘3’ with $\frac{1}{2}$ probability each. Clearly the most optimal
static code is one that assigns each of the four symbols a two bit encoding. If we use an adaptive approach,
the majority of the source sequence can be encoded at an expected 1.5 bits per input symbol, thus achieving
greater compression than the static approach.

4.2.2 Tree Initialization

One area that Gallager does not address is how the Huffman code tree should be initialized or built at the
beginning of transmission; he rather assumes that a tree already exists and describes how to update it as
relative source symbol frequencies change. One way to initialize the code tree is to take a page from the static
algorithm and do a partial scan: count the first $N$ source symbols, build a tree based on those frequencies,
and then transmit the tree and associated frequencies to the receiver. Transmission can then commence in
the usual fashion. A method proposed later by Vitter in 1987 [2] allows the construction of the Huffman
code tree as symbols are transmitted, thus providing a true one-pass algorithm.

4.2.3 Universal Algorithms: Comparison with LZ

The adaptive Huffman coding method (in Vitter’s form) is an example of a universal code because it is a
prefix code that does not require the sender or receiver to know the source statistics before transmission
of the actual source data can begin. Further, Vitter showed that the resulting encoding remains within a
constant factor of the optimal static Huffman encoding.

Other universal encoding schemes exist today, including the LZ family of dictionary algorithms. In a
way, we can see adaptive Huffman coding as a variation on Huffman coding in a similar way that LZ coding
is a variation on Tunstall coding. In addition, we note that Huffman coding can be seen as the optimal
fixed-to-variable length coding scheme while Tunstall coding can be seen as the optimal variable-to-fixed
length coding scheme.

The way that the adaptive Huffman algorithm chooses its code tree is remarkably different from the
approach that the LZ algorithm takes. If we set $\alpha = 1$ in the adaptive algorithm, the $n$-th source symbol is
encoded using, essentially, the optimal static Huffman tree built on the frequencies given by the first $n − 1$
symbols. On the other hand, in the LZ algorithm, the $n$-th transmission is not determined by the optimal
Tunstall code built by all symbols previously transmitted. Instead, it is based on a dictionary that possibly
contains numerous entries that are completely unnecessary. For example, if (assuming $\Sigma = \{a, b\}$) the strings
$aba$ and $abb$ are added to the dictionary, the string $ab$ becomes useless, but there is no effort made by the
algorithm to remove it from the dictionary, freeing up a code slot for use by another phrase.

Despite this suboptimality, in practice it is generally accepted that LZ coding achieves better compression
than adaptive Huffman coding. One reason often cited for this is that LZ coding can easily take advantage
of higher-order correlations between letters and words that often appear together by building a dictionary
with those words and phrases in it. At first glance, it might appear that Huffman coding is ill-equipped to
take similar advantage; however, if we increase the size of the source strings we encode to grow, Huffman
coding gains that ability by assigning a short code to blocks of symbols that appear frequently in words and
phrases.

For example, if we wish to take advantage of the correlation that occurs in sources with memory $M$, we
can use adaptive Huffman coding on strings of length $M$. The drawback is that there is a high computational
cost in managing a Huffman code tree with $2^M$ leaves. Vitter’s algorithm addresses this point to some extent
by using a single leaf for all strings that have not yet been transmitted. If we were encoding English text,
certain strings of length $M$ would just never come up while other strings are highly likely. When using
Vitter’s approach we would see that the code tree would fill with the limited subset of length-$M$ strings that
are also parts of English words and phrases.
Although we have shown that adaptive Huffman coding can take advantage of sources with memory, this analysis brings to light that LZ coding has the additional benefit of not having to guess $M$ beforehand. It simply builds its dictionary to the point where correlated words and phrases that appear frequently are all in its dictionary. Regardless of the memory length, the dictionary would rapidly acquire sequences that belong to the typical set of the source.

5 Conclusion

In this literature review, we have described the groundbreaking results Gallager presented in his 1978 work. The sibling property has been referenced in several subsequent works on Huffman coding and compression. Additionally, the idea of adaptive Huffman coding opened a new approach to using Huffman codes as a universal coding scheme. This scheme was later improved upon by several scientists including Vitter in 1987, who also provided a bound on the difference in performance between static and dynamic Huffman encoding. We have analyzed Gallager’s work on adaptive Huffman coding and compared it to the LZ family of algorithms, describing some of the costs and benefits of each algorithm type.

References
