A Context-Tree Branch-Weighting Algorithm

Paul A.J. Volf and Frans M.J. Willems
Eindhoven University of Technology
Information and Communication Theory Group

Abstract

The context-tree weighting algorithm [4] is a universal source coding algorithm for binary tree sources. In [2] the algorithm is modified for byte-oriented tree sources. This paper describes the context-tree branch-weighting algorithm, which can reduce the number of parameters for such sources, without increasing the complexity significantly.

1 Introduction

Contemporary universal source coding algorithms process a source sequence symbol by symbol. An adaptive modeling algorithm estimates the probability distribution of the new symbol, based on statistical information which it inferred from the previous symbols of the source sequence. The encoder and decoder use the same modeling algorithm. In the encoder, an arithmetic encoder uses the probability distribution and the actual value of the new symbol to form the code word. The decoder uses the probability distribution to decode the new symbol from the code word. Then they both update their statistics and process the next symbol. The arithmetic encoder and decoder perform nearly optimal, thus the performance of these universal source coding algorithms depends fully on the modeling algorithm.

To implement the modeling algorithm efficiently, it assumes that the source belongs to a specific model class, and it wants to find the model within that model class, that matches the source sequence best. But if the source sequence has not been generated by a source from that model class, then the actual model has to be approximated by a model in the model class. This often results in some extra redundancy.

Section 2 describes an efficient modeling algorithm for the class of tree sources, called the context-tree weighting algorithm [4]. Our main goal is to apply this algorithm to texts. But texts are not generated by tree sources. Therefore, the next section discusses an extension of the model class, and an efficient weighting algorithm for the new model class, called the context-tree branch-weighting algorithm. Next, the new weighting algorithm and the normal context-tree weighting algorithm are combined into a single algorithm. Finally, this combined algorithm is tested on some text files.

2 The context-tree weighting algorithm

The context-tree weighting algorithm assumes that the actual model is from the class of tree sources. A tree source consists of a suffix tree (the model) and a parameter vector. The suffix tree determines the state of the source, based on the context of the new symbol. The context of the new symbol is the suffix (of at most D symbols) of the source sequence seen so far. The first symbol of the context, the most recent symbol of the source sequence, will be used in the root of the suffix tree to choose a child (if it has any). Then the next context symbol is used in the child node. This recursion continues until it encounters a leaf. This leaf is the state corresponding to that context. Every leaf has its own parameter. It defines the probability distribution with which all symbols following that context are generated. Thus those symbols form a memoryless subsequence.

The context-tree weighting algorithm uses a context tree to collect the statistics of the tree source. A context tree is a full and complete tree of depth D. In every node and leaf s of this tree the algorithm counts for every symbol how often it followed context s. Furthermore, it stores two probabilities in every node and leaf s: the "estimated" probability $P_e^s(x_1^n)$ and the weighted probability $P_w^s(x_1^n)$. The estimated probability is the probability of those symbols of x_1^n which follow context s, assuming that this node is a leaf of the actual model. In that case these symbols form a memoryless subsequence. The block probability of this subsequence can be computed sequentially. If the subsequence contains a_s zeros and b_s ones so far, then the probability that the next symbol is 0, is defined as $\frac{a_s+\alpha}{a_s+b_s+\alpha}$, in which $\alpha > 0$. The block probability of the subsequence extended with a 0 can then be written as (similar for a 1):

$$P_e^s(x_1^n, x_{n+1} = 0) = P_e^s(x_1^n) \cdot \frac{a_s + \alpha}{a_s + b_s + \alpha}. \tag{1}$$

With $\alpha = 1$ (1) reduces to the Laplace-estimator, and with $\alpha = \frac{1}{2}$ it reduces to the Krichevsky-Trofimov estimator. If the probability of a memoryless sequence x_1^N is estimated with the Krichevsky-Trofimov estimator, then the individual redundancy can be upper bounded by [4]: $\frac{1}{2}\log_2 N + 1$.

The second probability in a node s is the weighted probability of the symbols of x_1^n with context s. The context-tree weighting algorithm weights in every node the probabilities of two possibilities: the actual model has a leaf here and the actual model has a node here. In the first case the subsequence is memoryless and the algorithm uses the estimated probability $P_e^s(x_1^n)$. In the latter case the subsequence can be split in two disjoint subsequences: one consisting of those symbols following context $0s$ ($0s$ denotes context s expanded with the preceding symbol 0) and one for those symbols following context $1s$. In this case the probability is the product of the probabilities of both disjoint parts. Thus the weighted probability in a node or leaf s of the context tree, where s has depth $l(s)$ is defined as:

$$P_w^s(x_1^n) = \begin{cases} P_e^s(x_1^n) & \text{if } l(s) = D, \\ \frac{1}{2}P_e^s(x_1^n) + \frac{1}{2}P_w^{0s}(x_1^n)P_w^{1s}(x_1^n) & \text{if } l(s) < D, \end{cases} \tag{2}$$

The weighted probability in the root node of the context tree λ is the weighted probability over the entire sequence so far, and with this probability the probability distribution for the arithmetic coder can be computed. The individual redundancy of a sequence x_1^N, compressed with the context-tree weighting algorithm compared to the situation in which the actual source (with model $\mathcal{S}$ and parameter vector Θ) is known, satisfies:

$$\log_2 \frac{P_a(x_1^N|\mathcal{S},\Theta)}{P_w^\lambda(x_1^N)} \le (2|\mathcal{S}| - 1) + (\frac{|\mathcal{S}|}{2}\log_2 \frac{N}{|\mathcal{S}|} + |\mathcal{S}|), \tag{3}$$

in which $P_a(x_1^N|\mathcal{S},\Theta)$ is the probability of the sequence given the actual model. The first term on the right hand side is the model cost (redundancy for not knowing the model), and the second term the parameter redundancy of the $|\mathcal{S}|$ Krichevsky-Trofimov estimators. For sequences of increasing length, the model cost will amount to a constant number of bits (1 bit per node and per leaf), but the parameter redundancy will continue to increase. Thus it is important that the number of parameters is as small as possible, even if this results in higher model cost.

3 The context-tree branch-weighting algorithm

3.1 The decomposition

The context-tree weighting algorithm gives excellent performance for binary sources. But in practice most sources will be non-binary (for example text-sources). Suppose that the symbol alphabet $\mathcal{X}$ is non-binary, thus $|\mathcal{X}| > 2$. One way to extend the context-tree weighting algorithm for such sources is to use a binary decomposition (see [1] and [2]). A binary decomposition is an arbitrary binary tree with $|\mathcal{X}|$ leaves. Every leaf corresponds to one of the source symbols. To every node of the binary decomposition belongs a context tree. A source symbol is encoded by tracing the path in the decomposition tree to the leaf corresponding to that symbol. The decoder is able to decode the symbol correctly if it can find the same path. Therefore the encoder has to encode for every node on the path which branch it chooses. To do that, the encoder updates the context tree corresponding to that node and applies the context-tree weighting algorithm to find a probability distribution over both branches. Finally, this probability distribution is used by an arithmetic encoder to encode the correct branch. The decoder is then able to find the same branch, and thus to trace the path in the tree towards the correct leaf.

Every node in the decomposition tree has two branches. Thus the context-tree weighting algorithm can use binary estimators. But the context symbols (the previous symbols in the source sequence) are non-binary. As a result, the $|\mathcal{X}| - 1$ context trees (the decomposition tree has $|\mathcal{X}| - 1$ nodes) consist of $|\mathcal{X}|$-ary nodes. The context-tree weighting algorithm for context trees with non-binary nodes is defined as (we omit the trivial case $l(s) = D$, for which $P_w^s = P_e^s$ from now on):

$$P_w^s(x_1^n) = \beta P_e^s(x_1^n) + (1-\beta) \prod_{t \in \mathcal{X}} P_w^{ts}(x_1^n), \quad \text{if } l(s) < D, \tag{4}$$

in which we also replaced the weights $\frac{1}{2}$ with weights β and $(1-\beta)$, with $0 < \beta < 1$.

3.2 Weighting per branch

Formula (4) weights between two possibilities: the actual model has a leaf here, or it has a node here with all its children. Thus this algorithm can be called the context-tree node-weighting algorithm. But this choice may be to strict. Consider a text-source, often a small part of the context will already be sufficient to estimate the probability distribution of the new symbol. But in some special cases, the part preceding this short context may give some valuable additional information. Thus we want to extend the model class in such a way that a "node" of the source may act as a leaf for most contexts, while it acts as an internal node for some specific contexts.

At first glance it may not seem necessary to develop a new weighting algorithm for such situations, because the context-tree node-weighting algorithm can "simulate" these situations. For a node with only a few of its branches, it will simply find a node with all its children. For the contexts for which this node should have been a leaf, the algorithm will find a leaf one level deeper in the tree and for the other contexts, the model can just grow further. But this has two distinct disadvantages. First of all, a lot of extra leaves will have to be created which could result in higher model cost. But more importantly, the counts which were originally collected in one "leaf" are now spread over many leaves. Thus the individual estimates will be less accurate than necessary which results in additional parameter redundancy. Therefore, by extending the model class this way, we may reduce the number of parameters, and thus the parameter redundancy.

3.3 An optimal solution

First we introduce some notations. We denote a branch by the value t of the context symbol for which that branch will be used. Consequently, the set of all possible branches is equal to the set of possible context symbols $\mathcal{X}$. This is reasonable because a branch t from a node s corresponds to its child node which has context ts with t from $\mathcal{X}$. In the same way a subset T of branches is equal to the set of contexts ts for which $t \in T$. We denote such a subset of contexts with Ts (thus $Ts \triangleq \{ts | t \in T\}$).

The optimal weighting algorithm for this model class has to weight for every node s over every possible subset T of its *branches*. Take a subset of branches $T \subseteq \mathcal{X}$. For all contexts ts for which $t \in T$ this node of the context tree will act as a node, while for all other contexts $t's$ with $t' \in \mathcal{X} \setminus T$ this node acts as an leaf. A single estimator will be used for all symbols following a context in the subset $T's$. For these symbols the probability in this node is: $P_e^{T's}(x_1^n)$. For all other symbols, those following a context ts in subset Ts, a branch exists and the weighted probabilities in their children will be used. For these symbols the probability in this node is: $\prod_{t \in T} P_{w,ts}^b(x_1^n)$, in which $P_{w,ts}^b(x_1^n)$ is the weighted probability in child node ts. Thus the probability of all symbols in this node s, given a subset T of branches, is defined as: $P_e^{T's}(x_1^n) \cdot \prod_{t \in T} P_{w,ts}^b(x_1^n)$. The new context-tree *branch-weighting* algorithm is obtained by weighting in this node over all possible subsets T. The new weighted probability in a node s over a sequence x_1^n is defined as:

$$P_{w,s}^b(x_1^n) \triangleq \sum_{T \subseteq \mathcal{X}, T' = \mathcal{X} \setminus T} P(T)(P_e^{T's}(x_1^n) \cdot \prod_{t \in T} P_{w,ts}^b(x_1^n)), \quad \text{if } l(s) < D, \tag{5}$$

in which $P(T)$ is an a priori probability distribution over the subsets T. For example, if all subsets of branches are equally likely, $P(T) = 2^{-|\mathcal{X}|}$.

3.4 Two examples

Let us examine the exact form of (5) for two examples: binary context trees and ternary context trees.

Example 1. First we examine the new weighted probability for binary nodes in case all subsets are equally probable. We will only examine the non-trivial case: nodes s for which $l(s) < D$. The weighted probability now reduces to:

$$P_{w,s}^b(x_1^n) = \frac{1}{4}\left(P_e^s(x_1^n) + P_e^{0s}(x_1^n)P_{w,1s}^b(x_1^n) + P_e^{1s}(x_1^n)P_{w,0s}^b(x_1^n) + P_{w,0s}^b(x_1^n)P_{w,1s}^b(x_1^n)\right)$$

$$\begin{aligned}= \frac{1}{4}\left(P_e^s(x_1^n) - P_e^{0s}(x_1^n)P_e^{1s}(x_1^n)\right) + \\ + \frac{1}{4}(P_e^{0s}(x_1^n) + P_{w,0s}^b(x_1^n))(P_e^{1s}(x_1^n) + P_{w,1s}^b(x_1^n)).\end{aligned} \tag{6}$$

In case a leaf is assigned probability $\frac{1}{2}$, and all other subsets are equally probable (thus they have probability $\frac{1}{6}$), the weighted probability reduces to a similar form:

$$\begin{aligned}P_{w,s}^b(x_1^n) = \left(\frac{1}{2}P_e^s(x_1^n) - \frac{1}{6}P_e^{0s}(x_1^n)P_e^{1s}(x_1^n)\right) + \\ + \frac{1}{6}\left(P_e^{0s}(x_1^n) + P_{w,0s}^b(x_1^n)\right)\left(P_e^{1s}(x_1^n) + P_{w,1s}^b(x_1^n)\right).\end{aligned} \tag{7}$$

The results in (6) and (7) are similar to the binary node-weighting algorithm. The first term is a term with only estimated probabilities, and only one of both parts of the last term has to be updated for each symbol. Furthermore, in both cases this branch weighting algorithm does not need any additional terms P_e.

For binary context trees the branch-weighting algorithm will not result in significant increase in complexity. But for binary context trees this branch weighting is not necessary. The new model class now also contains models in which some nodes have only one child. In that case an estimator is used to estimate the probability of the symbols for the missing child. Thus the parameter which in the old model class would have been one level deeper in the tree is now in this node. But the number of parameters will not decrease. There will only be a difference in the way the model cost are computed and this can also be achieved in the node-weighting algorithm by changing the weights β in (4). Thus for binary context trees the complexity will not increase, but the model class has not been extended either. The non-binary branch-weighting algorithm is more interesting.

Example 2. Now we examine the weighted probability for ternary nodes in the context tree. We assume that all subsets T are equally probable. Then the weighted probability for a node s with $l(s) < D$ is:

$$\begin{aligned}P_{w,s}^b(x_1^n) = \frac{1}{8}\big(P_e^s(x_1^n) + P_e^{0s}(x_1^n)P_{w,1s}^b(x_1^n)P_{2s}^b(x_1^n) + \\ + P_e^{1s}(x_1^n)P_{w,0s}(x_1^n)P_{w,2s}(x_1^n) + P_e^{2s}(x_1^n)P_{w,0s}(x_1^n)P_{w,1s}(x_1^n) \\ + P_e^{\{0,1\}s}(x_1^n)P_{w,2s}^b(x_1^n) + P_e^{\{0,2\}s}(x_1^n)P_{w,1s}^b(x_1^n) + \\ + P_e^{\{1,2\}s}(x_1^n)P_{w,0s}^b(x_1^n) + P_{w,0s}^b(x_1^n)P_{w,1s}^b(x_1^n)P_{w,2s}^b(x_1^n)\big),\end{aligned} \tag{8}$$

in which $P_e^{\{0,1\}s}$ is the estimated probability of the symbols following contexts $0s$ *and* $1s$, estimated by a single estimator. But (8) cannot be written in a more compact form due to the many additional estimated probabilities. In this case the branch-weighting algorithm has to update all 8 terms for each symbol, and it needs many additional estimated probabilities compared to the node-weighting algorithm. Thus for non-binary context trees, the optimal branch-weighting algorithm is considerably more complex (in computational time and memory) than the node-weighting algorithm.

3.5 An approximation

For non-binary sources the optimal branch-weighting algorithm seems too complex to use. The main reason is the large number of different estimated probabilities that have to be computed and stored in every node of the context tree. An estimated probability in a node s can be written as:

$$P_e^s(x_1^n) \triangleq \prod_{t \in \mathcal{X}} P_e^{(t)s}(x_1^n), \tag{9}$$

in which $P_e^{(t)s}$ is the probability of all symbols following context ts, estimated with the estimator in node s each time they occurred. Thus if a new symbol x_{n+1} has context ts probability $P_e^{(t)s}$ will be updated with the estimate of the estimator in node s: $P_e^{(t)s}(x_1^{n+1}) = P_e^{(t)s}(x_1^n)P_e^s(X_{n+1} = x_{n+1}|x_1^n)$. The other probabilities $P_e^{(y)s}$ with $y \neq t$ will not change. Splitting the probability like this has been done before by Weinberger et al. in [3]. Now we can approximate the various estimated probabilities with:

$$P_e^{0s}(x_1^n) \approx P_e^{(0)s}(x_1^n), \tag{10}$$

$$P_e^{\{0,1\}s}(x_1^n) \approx P_e^{(0)s}(x_1^n)P_e^{(1)s}(x_1^n), \tag{11}$$

$$P_e^{T's}(x_1^n) \approx \prod_{t \in T'} P_e^{(t)s}(x_1^n), \tag{12}$$

in which $T' \subseteq \mathcal{X}$. These approximations reduce the optimal branch-weighting algorithm of (5) to the following approximated branch-weighting algorithm (note the similarity with the binomial theorem), assuming that all subsets are equally probable:

$$P_{w,s}^b(x_1^n) = \prod_{t \in \mathcal{X}} (\frac{1}{2}P_e^{(t)s}(x_1^n) + \frac{1}{2}P_{w,ts}^b(x_1^n)), \quad \text{for } l(s) < D. \tag{13}$$

The algorithm of (13) can also be explained in an intuitive way. For every *branch* t the algorithm weights between two probabilities: the probability in case the actual model has a leaf for that context and the probability in case it has a branch for that context. In the first case the estimated probability of the symbols with context ts, estimated with the estimator in this node s ($P_e^{(t)s}$) is used, and in the latter case the weighted probability of the child node ($P_{w,ts}^b$) is used. Also compare (13) with examples (6) and (7), the examples have similar product terms. In this paper the approximated form (13) will be termed as *the* context-tree branch-weighting algorithm.

The advantage of the approximated algorithm is, that for every update only one of the terms of the product changes (only the term corresponding to the context of the new symbol). Thus the computational complexity is similar to that of the node-weighting algorithm. Moreover, it also uses only estimated probabilities of the form $P_e^{(t)s}$ (one per child node), thus also the memory complexity will be similar to the node-weighting algorithm. Furthermore, the approximation may seem very crude, but it still has the property that the estimated probability in case the node of the context tree is a leaf of the actual model with no branches at all is equal to the estimated probability in the node-weighting algorithm. This follows immediately from (9).

4 A general weighting algorithm

A disadvantage of the branch weighting algorithm written as (13) is the small probability assigned to a leaf (the empty subset T). If a node is visited by $|T|$ different contexts, the probability of a leaf will be $2^{-|T|}$. This can be corrected by changing the weights $\frac{1}{2}$ within each term of the product to weights γ and $1-\gamma$. But then it is still difficult to assign such weights that a leaf has a priori probability $\frac{1}{2}$, because the actual number of different contexts in each node is not known in advance. Actually by defining only the γ's not every desirable distribution over the models can be realized. Consider example 1. In the first case (6) the first term, will cancel out once the approximations are used. Thus with (13), this case (all models have an equal a priori probability) can be realized. But in the second case (7), the first term will *not* cancel out by using the approximations. The same holds for all context trees with nodes with a higher degree: it is not possible to assign probability $\frac{1}{2}$ to a leaf and equal probabilities to all other subsets of branches. Therefore we weight the branch-weighting probability again with the estimated probability, resulting in a term much like (7). The more *general branch-weighting* algorithm is defined as:

$$P^g_{w,s}(x_1^n) = \beta P_e^s(x_1^n) + (1-\beta)\prod_{t\in\mathcal{X}}(\gamma P_e^{(t)s}(x_1^n) + (1-\gamma)P^g_{w,ts}(x_1^n)), \quad \text{for } l(s) < D. \quad (14)$$

In (14) the estimated probability (P_e^s) is weighted with the weighted probability of the branch-weighting algorithm (which also contains this term). This may seem like doing things twice, but it gives us more freedom in defining the a priori probability distribution over the subsets of branches, and thus over the models. Furthermore, with $\beta = 0$ (14) reduces to the branch-weighting algorithm, while with $\gamma = 0$ it reduces to the node-weighting algorithm. In this way we captured both weighting algorithms in one formula, which allows us to mix them. That is why we termed (14) as a more general weighting algorithm.

5 Results

The context-tree node-weighting, the context-tree branch-weighting, and the general weighting algorithm each compressed 4 files of the Calgary Corpus: the text-files `book2` (610856 characters), `paper2` (82199 characters), and `progl` (71646 characters) and the object file `obj2` (246814 bytes). The algorithms used depth D=8 (symbols), and for the text files the decomposition from [2] was used. The algorithms use a decomposition based on the binary representation of the symbols for the object file `obj2`. For each file and each algorithm we searched for the best set-up. The value of α was decreased with a factor two each time (thus only even denominators were used), the β and γ were increased from 0 with steps of 0.025. The best results have been collected in table 1.

The numbers in table 1 have been obtained with a simulation program. The algorithms are quite robust, small variations in the parameters will only result in small variations in the results. The difference between the best solution and the best-but-one, is sometimes only a few bits.

From table 1 it is clear that the node-weighting algorithm gives better performance than the branch-weighting algorithm. This can be due to the approximation, but also due to the restrictions on the probability distribution over the models. The general

File	Algorithm	α	β	γ	bps	bytes
`book2`	Node-weighting	$\frac{1}{8}$	0.45	–	1.8688	142698
	Branch-weighting	$\frac{1}{20}$	–	0.475	1.8931	144549
	General weighting	$\frac{1}{12}$	0.35	0.20	1.8499	141254
`obj2`	Node-weighting	$\frac{1}{16}$	0.4	–	2.4332	75070
	Branch-weighting	$\frac{1}{66}$	–	0.475	2.4985	77082
	General weighting	$\frac{1}{38}$	0.275	0.225	2.3850	73581
`paper2`	Node-weighting	$\frac{1}{10}$	0.45	–	2.1886	22488
	Branch-weighting	$\frac{1}{36}$	–	0.45	2.2487	23105
	General weighting	$\frac{1}{16}$	0.375	0.15	2.1747	22345
`progl`	Node-weighting	$\frac{1}{16}$	0.375	–	1.5745	14101
	Branch-weighting	$\frac{1}{44}$	–	0.375	1.6277	14577
	General weighting	$\frac{1}{26}$	0.3	0.125	1.5561	13936

Table 1: The best set-ups for each algorithm and each file.

weighting algorithm has more freedom in choosing the probability distribution, and it always chooses a weight for γ greater than zero. Thus the algorithm indeed finds nodes for which it is best that for some contexts it acts as a node, while it acts as a leaf for others. The gain compared to the node-weighting algorithm is about 1 %. For comparison: the state-of-the-art compressor ACB 1.29a achieves 1.94 bps, 2.20 bps, 2.34 bps, and 1.50 bps on `book2`, `obj2`, `paper2`, and `progl` respectively.

6 Conclusions

The context-tree weighting algorithm with decomposition gives a good performance on object- and text files. This paper presented a new extension on the model class, and 3 new weighting algorithms for this model class. These algorithms should improve on the performance of the context-tree weighting algorithm on text files and object files by reducing the number of parameters.

First an optimal algorithm for the new model class has been described. But it is too complex to use. Then we used an approximation in the optimal algorithm to find the context-tree branch-weighting algorithm, which has the same complexity as the context-tree weighting algorithm. But the new algorithm does not perform as well as the context-tree weighting algorithm, because the probability distribution over the models is too restricted. Finally the more general weighting algorithm was presented, which gave us more freedom in determining a good probability distribution over the models. This algorithm can choose whether to use the branch-weighting or not. In practice it indeed uses the extended model class. Thus the extension of the model class is useful, even though we use an approximation. The performance of the more general algorithm is slightly better (1 %) than the context-tree weighting algorithm.

References

[1] R. Koster. Compressie van teksten met behulp van het contextweegalgorithme. Stage verslag, Eindhoven University of Technology, June 1994. In Dutch.

[2] P.A.J. Volf. Text compression methods based on context weighting. Technical report, Stan Ackermans Institute, Eindhoven University of Technology, June 1996.

[3] M.J. Weinberger, J.J. Rissanen, and R.B. Arps. Applications of universal context modeling to lossless compression of gray-scale images. *IEEE Trans. Image Processing*, 5(4):575–586, 1996.

[4] F.M.J. Willems, Yu.M. Shtarkov, and Tj.J. Tjalkens. The context-tree weighting method: Basic properties. *IEEE Trans. on Information Theory*, 41(3):653–664, 1995.