6.975 Week 5: Grammar-Based Codes

Emin Martinian

October 7, 2002

Abstract

This week's paper [1] discusses a method of lossless data compression called grammar-based codes. These codes compress a data string, $\mathbf{x}$, by first transforming it into a context-free grammar $\mathbf{G}$, and then compressing $\mathbf{G}$. The main result of the paper is that this compression technique will be universal (*i.e.*, asymptotically achieve optimal compression without knowing the source model) for the class of finite state sources provided that the grammar $\mathbf{G}$ satisfies some mild conditions.

Grammar based codes can be viewed as a generalization of the well known Lempel-Ziv lossless compression algorithm. The main advantage of the grammar framework is that it provides a way to construct families of universal compression algorithms and thus optimize their components. A companion paper [2] illustrates simulation results demonstrating that grammar based codes can significantly outperform the Lempel-Ziv algorithm.

1 Introduction

Transform coding is a well-known source coding technique for lossy compression applications such as image compression. The idea behind transform coding is to find a basis where the components of the data vector become independent or at least uncorrelated and then use scalar quantization to compress each component separately. Grammar based codes operate in an analogous manner. The data string is represented using symbols of a context-free grammar and then the symbols of the grammar are compressed using a memoryless lossless source code such as an arithmetic code.

Before presenting the definition of a context-free grammar, we illustrate this idea with an example. Suppose we wish to compress the data string

$$\mathbf{x} = cababcccababcccab. \tag{1}$$

Notice that $\mathbf{x}$ consists mostly of the patterns ab and ccc. This observations suggests that we could define a variable A_1 to represent ab and another variable A_2 to represent ccc. Using these definitions we can represent $\mathbf{x}$ as

$$\mathbf{x} = cA_1A_1A_2A_1A_1A_2A_1.$$

Once again we observe that the data contains the pattern $A_1A_1A_2$, thus we define the variable A_3 to represent $A_1A_1A_2$ to yield

$$\mathbf{x} = cA_3A_3A_1.$$

Thus we have transformed the data string in (1) into the grammar $\mathbf{G_x}$ specified by the following production rules:

$$A_0 \rightarrow cA_3A_3A_1 \tag{2}$$

$$A_1 \rightarrow ab \tag{3}$$

$$A_2 \rightarrow ccc \tag{4}$$

$$A_3 \rightarrow A_1A_1A_2 \tag{5}$$

with the understanding that $\mathbf{x}$ is derived by starting with the variable A_0 and applying the relevant production rules. Intuitively, $\mathbf{G_x}$ is a representation of $\mathbf{x}$ in a basis which captures the relationship between letters by parsing commonly occurring letters together into variables. With inter-letter redundancy accounted for by the parsing, the redundancy due to the different frequencies of variables can be accounted for using memoryless entropy coding (*e.g.*, Huffman coding or arithmetic coding). In the following sections, we present various theorems to justify this intuition.

2 Grammars Yielding Universal Codes

The main focus [1] is on classes of grammars which result in universal source codes. A grammar transform $\mathbf{x} \to \mathbf{G}$ is defined to be asymptotically compact if

$$\text{For each } \mathbf{x}\text{, the grammar } \mathbf{G}_{\mathbf{x}} \text{ representing } \mathbf{x} \text{ belongs to } \mathcal{G}^*(\mathcal{A}) \tag{6}$$

$$\lim_{n\to\infty} \max_{\mathbf{x}\in\mathcal{A}^n} \frac{|\mathbf{G}_{\mathbf{x}}|}{|\mathbf{x}|} = 0 \tag{7}$$

where $\mathcal{G}^*(\mathcal{A})$ will be defined shortly and $|\mathbf{x}|$ and $|\mathbf{G}_{\mathbf{x}}|$ represent the length of data string $\mathbf{x}$ and the sum of the lengths of the production rules in the grammar $\mathbf{G}_{\mathbf{x}}$ respectively.

Asymptotically compact grammars are interesting because [1, Theorem 7] provides a bound on the maximal pointwise redundancy of a grammar based code as a function of

$$\max_{\mathbf{x}\in\mathcal{A}^n} \frac{|\mathbf{G}_{\mathbf{x}}|}{|\mathbf{x}|}.$$

This result essentially means that in order to design good universal lossless source codes we can focus on the more concrete problem of designing asymptotically compact grammars.

Before summarizing the conditions for a grammar to be in $\mathcal{G}^*(\mathcal{A})$ we recall the formal definitions of a context-free grammar.

2.1 Context-Free Grammars

A context-free grammar, $\mathbf{G}$, designed to represent strings in the alphabet T using variables in the set V where $T \cap V = \varnothing$, is a quadruple (V, T, P, S). The set P contains production rules of the form

$$A \to \alpha$$

indicating that any occurrence of the variable A can be replaced with the string α. Left members of rules must be elements of V while right members must be strings of elements in $V \cup T$. The set of strings which can be derived by applying the rules in P to the start symbol $S \in V$ are referred to as the language of $\mathbf{G}$, $L(\mathbf{G})$. For example, the grammar in Section 1 is the quadruple corresponding to

$$\begin{aligned}
V =& \{A_0, A_1, A_2, A_3\} \\
T =& \{a, b, c\} \\
P =& \{A_0 \to cA_3A_3A_1, A_1 \to ab, A_2 \to ccc, A_3 \to A_1A_1A_2\} \\
S =& A_0
\end{aligned}$$

and the language of this grammar is the data string in (1).

2.2 Requirements For $\mathcal{G}^*(\mathcal{A})$

Essentially $\mathcal{G}^*(\mathcal{A})$ is a set of "reasonable" grammars which are constructed to avoid obviously inefficient production rules, rules which make the transformation $\mathbf{x} \to \mathbf{G}$ not invertible or rules which make the mapping from $\mathbf{G}$ to a data string one-to-many. The requirements for a grammar to be in $\mathcal{G}^*(\mathcal{A})$ are the following:

1. For each variable A in the set of variables $V(\mathbf{G})$, there is exactly one production rule in $P(\mathbf{G})$ whose left member is A. (This requirement insures that $\mathbf{G}$ maps to a unique $\mathbf{x}$).

2. The empty string is not the right member of any production rule. (This requirement insures that $\mathbf{G}$ does not contain useless rules which expand to nothing.)

3. $L(\mathbf{G})$ is non-empty. (This requirement insures that $\mathbf{G}$ represents some string.)

4. $\mathbf{G}$ has no useless symbols, *i.e.*, symbols which can not be reached from the start symbol.

5. The variable naming for $\mathbf{G}$ follows a certain canonical convention. (This is required to specify an efficient encoding of the rules of $\mathbf{G}$. Since any variable naming scheme can easily be transformed to satisfy the canonical naming convention this requirement is quite mild).

6. Whenever A and B are distinct variables $f^{\infty}_{\mathbf{G}}(A) \neq f^{\infty}_{\mathbf{G}}(B)$ where $f^{1}_{\mathbf{G}}(\alpha)$ is the result of replacing every variable in the string α with the corresponding production rule and $f^{i}_{\mathbf{G}}(\alpha) = f^{1}_{\mathbf{G}}(f^{i-1}_{\mathbf{G}}(\alpha))$. (This rule states that no two distinct variables expand to the same result. If such an A and B did exist in $\mathbf{G}_{\mathbf{x}}$, then every occurrence of B could be replaced with A without changing the string $\mathbf{x}$ represented by $\mathbf{G}_{\mathbf{x}}$.)

In [1], the first four conditions are presented as requirements of an admissible grammar while the last two are presented as requirements of $\mathcal{G}^*(\mathcal{A})$ which is a subset of admissible grammars.

2.3 Irreducible Grammars

The authors introduce the class of irreducible grammar transforms which, according to [1, Theorem 8], are guaranteed to be asymptotically compact (*i.e.*, yield a universal code). Specifically, a grammar $\mathbf{G}$ is irreducible if the following four properties are satisfied:

1. $\mathbf{G}$ is admissible (*i.e.*, $\mathbf{G}$ satisfies conditions 1-4 in Section 2.2).
2. If v_1 and v_2 are distinct variables in $V(\mathbf{G})$, then $f^{\infty}_{\mathbf{G}}(v_1) \neq f^{\infty}_{\mathbf{G}}(v_2)$.
3. Every variable in $V(\mathbf{G})$ other than the start symbol appears at least twice as an entry in the right members of the production rules of the grammar $\mathbf{G}$.
4. There does not exist any pair Y_1, Y_2 of symbols in $V(\mathbf{G}) \cup T(\mathbf{G})$ such that the string Y_1Y_2 appears more than once in non-overlapping positions as a substring of the right members of the production rules for $\mathbf{G}$.

Essentially, an irreducible grammar is a grammar that is well designed in the sense that it does not contain redundant variables (condition 2), it does not contain unnecessary variables which are only used once (condition 3), and no further patterns exist in multiple rules which could be compressed with an additional rule.

In [1, Section VI], the authors provide five reduction rules which can always be applied to produce an irreducible grammar transform. These rules provide a way to construct a family of universal lossless compression algorithms. By choosing the order these rules are applied (or by adding additional rules), a system designer can optimize the performance of the resulting algorithm or tailor it for specific applications while still guaranteeing universality.

3 Conclusion

The key points of this week's paper are the introduction of grammar based codes, a list of grammar properties which are sufficient to yield a universal lossless source code, and a set of reduction rules which are guaranteed to produce a grammar satisfying these properties. These results are a step in moving universal lossless compression from an art to a science since they provide a systematic method of designing lossless compression algorithms. In the presentation on Wednesday we will discuss some more examples and explore the intuition behind the proofs for these results. Some examples of the superior performance obtained by optimizing this construction are discussed in [2], [3], [4].

References

[1] J. C. Kieffer and E.-H. Yang. Grammar-based codes: A new class of universal lossless source codes. *IEEE Trans. Inform. Theory*, 46(3):737–754, May 2000.

[2] E.-H. Yang and J. C. Kieffer. Efficient universal lossless data compression algorithms based on a greedy sequential grammar transform–part one: Without context models. *IEEE Trans. Inform. Theory*, 46(3):755–777, May 2000.

[3] E.-H. Yang, A. Kaltchenko, and J.C. Kieffer. Universal lossless data compression with side information by using a conditional mpm grammar transform. *IEEE Trans. Inform. Theory*, 47(6):2130–2150, Sep 2001.

[4] J.C. Kieffer, E.-H. Yang, G.J. Nelson, and P. Cosman. Universal lossless compression via multilevel pattern matching. *IEEE Trans. Inform. Theory*, 46(4):1227–1245, July 2000.