Summary 1: Identifying Hierarchical Structure in Sequences: A linear-time algorithm (by Nevill-Manning and Witten)

The Sequitur algorithm determines the hierarchical structure from a sequence of discrete symbols by replacing repeated phrases with a grammatical rule that generates the phrase, and repeats this process recursively. The beauty of the algorithms is that it offers insights into the lexical structure of the original sequence, based on the hierarchical representation of the original sequence. This scheme has many applications, including compression and determination of structure.

The basic insight of the Sequitur algorithm comes from the fact that many different sequences of discrete symbols exist in nature. Examples of such sequences include text of a given language, which is broken down into paragraphs, sentences, phrases, and words, and music, which can be broken down into major sections, motifs, bars, and notes. Many areas of research search for such hierarchical structure, but one prime area for application of such techniques is compression, as such structure can be used to predict upcoming symbols so that they can be encoded efficiently. In other words, Sequitur factors out repetition in the sequence. As Sequitur can deal with long sequences, it can also be used in other applications, such as hierarchically organizing libraries of text, which may offer a novel method of browsing. Thus, we see that the Sequitur algorithm is good both for compression and determining structure.

The general Sequitur algorithm is based on the formulation of a grammar from a sequence on repeated phrases in that sequence. Furthermore, every time a phrase is repeated in the sequence, the grammar can be used to generate the phrase (and thus another grammar not needed for the repeated phrase). In general, the grammars protect two properties: (1) no pair of adjacent symbols appears more than once in the grammar; (2) every rule is used more than once. For instance, in the sequence “hollywolly”, the entire string will be represented as a symbol, and then the repeated substrings will also be represented as a symbol: A → “hollywolly”, B → “olly”, etc. The second constraint, called the “rule utility” constraint, indirectly provides a mechanism for forming long rules, which can be performed efficiently with the appropriate data structures.

After describing the algorithm in detail, Nevill-Manning and Witten discuss actual structures that can be inferred from realistic sequences. I thought it was nice to see the applicability of their approach in a general sense, and really enjoyed this section of the paper. I particularly liked the examples they showed of the Bach chorale scores.

In the next section, the authors described the implementation of Sequitur. I won’t go into much detail on this section, but it’s important to note that the data structures they chose had to do a few following operations efficiently. Specifically, they used a doubly linked list in order to efficiently append a rule to S so S can be lengthened and shortened.
quickly. They also used a hash table to store the diagram index, which permits fast access and minimal work for adding and deleting entries. The authors also showed that the computational complexity of the algorithm is linear in both space and time. I found their proof to be a bit awkward, although I hope we have a chance to prove something similar in the homework so I can get a better grasp on it.

Finally, I found the author’s inclusion of the “Exploring the Extremes” section particularly refreshing, as I think authors sometimes like to sweep problem cases “under the carpet.” I also found their “Behavior in Practice” section interesting, although I didn’t appreciate the fact that they mention their algorithm performs well “up to 40MB”, as many compression candidates are much larger than that. I am interested in seeing an extension of this work that is better suited to very large data sets.

Overall, I thought the paper was interesting, but it certainly wasn’t on my favorites list for the semester. I also thought that the authors sometime used language that seemed to be trying to convince the reader that their method was “great”, whereas I generally prefer it when the author leaves it to the reader to say, “wow, that’s a great idea” (just personal preference).
Summary 2: The Entropy of English Using PPM-Based Models (by Teahan and Clearly)

In this paper, Teahan and Clearly develop a new estimate for the entropy on the English language that closes the gap between human estimates and machine estimates done in previous studies over the last 45 years. The authors present a method that uses PPM models along with a number of enhancements that shows that it is difficult to obtain large improvements in machine models, but the best machine models can perform in the range achieved by humans.

The authors motivate their problem by discussing the work that has been done on the subject over the last five decades, starting with the eminent Shannon. Shannon used a human-based model to give an estimate of 1 bit per character (bpc). Other researchers such as Cover and King used a gambling approach to estimate 1.25 bpc, while researchers in the cryptographic community used n-gram analysis to achieve a 1.5bpc asymptotic limit for 26-letter English. Note that the “entropy in a language” refers to the information content of that language – it is the amount of information one needs to use to encode the language such that, after compressed, the original text can be obtained. Of course, this is possible due to the large amount of redundancy in English. As a very simple example, we as humans can probably guess the letter currently denoted by * in the following sentence: “*he boy went to *he store”. Thus, our prior experiences as humans with the English language allow us to use context and our personal “dictionary” to “fill in the gaps.” The inherent structure and regularity in the English language point to the fact that its information content, or entropy, can indeed be bounded.

Teahan and Cleary staunchly believe that machine models should be able to achieve results similar to human models. In the text, they essentially show the necessary enhancements necessary to make PPM perform much better over the original or baseline implementations. The first notable enhancement is the use of training text to improve PPM models. The reason this enhancement works well is because a lot of the poor performance in the baseline PPM implementation comes from very poor performance during the start of PPM, since it hasn’t yet built up the counts for the higher order context models, and thus must resort to lower order models. They claim that the training text should be either by the same author, or on a similar topic to achieve the most improvement through training. The also attack the question of how much training is necessary, and claim that although more is generally better, using around than 1% of a large body of text (roughly on the order of 1M words) provides a marked improvement.

The next enhancement they show is to enlarge the alphabet, based on the premise that some pairs of characters (bigrams) occur more frequently than individual letters. For example, they would replace the pair “th” with a new symbol. Such replacements fall into three categories: bigram-based methods, such as the “th” example above; digram-based methods, which replace a group of two successive letters whose phonetic value is a single sound (e.g., the “ng” in “bring”); and vowel/consonant-based methods, which uses vowels and consonants to define bigrams that have distinct phonetic sounds.
Overall, they show that bigram-based techniques perform best of the encoding techniques when using trained text. In the best case, for a large sample of text, they achieve a 1.488 bpc estimate of entropy in English. In sum, they claim that machine models can be improved most markedly by using a large amount of training text that is closely related to the body of text to be assessed, replacing frequent bigrams with their own symbols, and using texts with few typographical errors.

Overall, I found this paper quite interesting. However, I found the fact that the upper bound on the entropy of English is actually smaller than I would have guessed. It’s also refreshing (from a theoretical and academic standpoint) to see that we can theoretically build machines to do this sort of thing about as good as humans.