The Unique Decypherability Problem

Basic Definitions: A *code* C is just a finite set of finite length strings, called *codewords*. A *message* M is a string created by concatenating strings from C, with repetitions allowed. So there is no bound on the length of a message. A message M is called *uniquely decypherable (UD)* if there is only one way to write M as the concatenation of strings in C. For example, if $C = \{aba, a, abb, ab, ba\}$ then the message $abbab$ is UD, while the message aba is not UD. A code C is called UD if and only if *every* message that can be created from C is UD. So, the code C in the example is not UD, but if we remove the strings ba and ab from C, the resulting code would be UD. Note that the definition of a code being UD is over an infinite set of messages.

The **UD problem**, given a code C, is to determine if C is UD. At first, it may seem that the UD problem would be undecidable (it seems similar to the famous undecidable Post Correspondence Problem). But, in fact it has a polynomial-time solution, and it is even conjectured to have a linear-time solution. There are several algorithms for the UD problems, but they are all similar and can be thought of as a path problem on a graph.

Given C, we build a graph G with one node for each string that occurs as a suffix of any codeword in C, including a node for each complete codeword (the trivial suffix). Note that the same string might occur as a suffix of more than one codeword, but it is only represented by a single node in G. We label each node by the suffix (string) it represents. In graph G, there is an edge from node u to node v is there is a codeword c such that $c = uv$ (this is an L1 edge), or if $u = cv$ (this is an L2 edge).

Then, C is not UD if and only if there is a path with at least one edge from a node representing a codeword to a node representing a codeword.

Your problem is to give a proof of this result. Try to make the proof contain an intuitive explanation.

OPTIONAL PROBLEMS:

A. With the use of a suffix tree, all the L1 and L2 edges can be found in $O(nm)$ time, where n is the total length of the codewords, and m is the number of codewords. It is not an assigned problem, but you might think about how to do this.

B. An open question is to find a solution to the UD problem that runs in $O(n)$ time.

C. Here is another open problem related to the UD problem. It is known that a necessary condition for a code to be UD is the following:

Let Σ denote the alphabet of characters used in the codewords. In the above example, $\Sigma = \{a, b\}$, but in general the alphabet does not need to be binary. Now chose any probability distribution p on the characters in the alphabet, for example $p(a) = 1/3, p(b) = 2/3$, and then compute the probability of each codeword in the code, using the distribution p. For example $p(aba) = 1/3 \times 2/3 \times 1/3$. Then sum those probabilities over all the codewords in the code. Let $S(C, p)$ denote the sum, where C is the code and p is a chosen probability distribution on the alphabet of characters used in the codewords.

A necessary, but not sufficient condition for a code C to be UD is that $S(C, p) \leq 1$ for any p. This is a generalization of what is called the MacMillin condition, which is really just the special case where p is the uniform distribution.

Open Problem: Given C, what is the p that maximizes $S(C, p)$? More realisticly, what is an (efficient) algorithm to find the best p?