Mathematical Optimization and the Synchronizing Properties of Encodings

LARRY E. STANFEL

Department of Quantitative Business Analysis, Louisiana State University, Baton Rouge, Louisiana 70803

Methods for constructing binary exhaustive prefix codes with certain synchronizing properties are given. In particular it is shown how to formulate the requirement that a code be anagrammatic and the requirement for minimal synchronizing delay as linear integer programming problems. The tree structure of the code is given and the program optimizes over all possible labelings of the tree. Several examples, solved by a commercially available linear/integer programming package, are included. © 1988 Academic Press, Inc.

I. PRELIMINARIES

If $E$ is a set of symbols then let $E^i$, $i = 0, 1, 2, \ldots$, denote the set of all symbol strings of length $i$ over $E$. The set of all strings over $E$, $E^1 \cup E^2 \cup \cdots$, is often denoted $E^*$. By an encoding over $E$ of a finite set $S$ is meant a mapping of $S$ into $E^*$, and the mapping may be represented just by the set of $E^*$ elements and the association with $S$ elements. For example, with $E = \{0, 1\}$ an encoding of $\{s_1, s_2, s_3, \ldots, s_6\}$ is $W = \{1, 11, 00, 1100, 10101, 110011\}$, where $s_1$ is represented by $1$, $s_2$ by $11$, etc.

Elements of $W$ are called code words and concatenations of $W$ elements are messages. A message from the example code above is $110011$, and $W$ is not uniquely decipherable (ud), since there are messages which may be decoded validly in more than one way. Other authors reserve the term "code" for the ud case.

If a string $e \in E^*$ may be expressed as the concatenation of two non-empty strings $\alpha, \beta$, $e = \alpha\beta$, then $\alpha$ is a prefix of $e$ and $\beta$ is a suffix of $e$. As is well known if an encoding $W$ has the property that none of its words is the prefix of another word, the encoding is uniquely decipherable.

A proper prefix of $W$ is a non-null prefix.

A convenient representation of a prefix code is available in the form of a tree, which may be drawn as an oriented edge-labeled graph, but where the code words are described by the sequence of labels along paths from a recognized root to all terminal (degree one) vertices.
FIG. 1. A prefix code.

For example, (000, 001, 10, 11) can be represented as shown in Fig. 1. An exhaustive prefix code (epc), $W$ over alphabet $E$ (of which Fig. 1 is not an example), is one with the property that any string of symbols from $E$ is either a message from $W$ or the prefix of a message from $W$. In other words, if $x$ is a string from $E^+$, $e \in E$, and $xe$ is the prefix of $w \in W$, then $xe'$ is the prefix of $w' \in W$ or $xe' \in W$. In terms of trees, an exhaustive prefix code will have the property that if a vertex has a successor $v_1$, then it will have a successor for each symbol of the alphabet.

If we add the word 01 to the example in Fig. 1 we obtain an exhaustive prefix code with the graph of Fig. 2.

The terminology of the foregoing definitions is that originally introduced when the concepts began being studied. Subsequently, there have been changes, and while the definitions given are best for our purposes, some related properties will be defined. A comprehensive reference in this case is [1], suggested by a referee. It is about the exhaustive property that elaboration is required. While it is the desired notion for these investigations it has lost currency. A maximal code is one that is not a proper subset of another code over the same alphabet. It is easy to establish that for prefix codes, a maximal code is exhaustive, so having restricted attention to prefix codes we could demand maximality and get the desired property.

The idea of a complete code is also related. A code $X$ is said to be complete if for every symbol string $m$, there are strings $u, v$ such that $umv$ is a message from $X$. Thus, a complete code is not necessarily exhaustive, whereas by taking $u$ always null we see that an exhaustive code is complete.

Among the set of exhaustive prefix encodings it is possible to discuss the synchronizing properties of the code. While the concept is not restricted to binary alphabets, our discussion will be.

FIG. 2. An exhaustive prefix code.
Messages are transmitted as unspaced strings of code words as Fig. 3 illustrates, where words from the code of Fig. 2 are used. Solid vertical bars indicate the intended words. Suppose, for some reason, the first symbol is not received. The small arrow indicates the perceived start of the message, and the dashed vertical lines, the words decoded. After discovering various erroneous words, decoding becomes synchronized with the intended message. Beyond the point where are found both dashed and solid vertical lines, the intended message would be received as intended.

Symbolically, what has occurred is that a suffix of a word \( \beta \) has been followed by a message \( m \), such that \( \beta m \) is also a message; \( m \) is a synchronizing message for \( \beta \). Since each word is transmitted with a non-zero probability at any time, if every suffix in the code has a synchronizing message and if the transmission has sufficient length, decoding will get synchronized with probability one. Such a code is called completely synchronizing or ergodic. If some suffixes but not all suffixes own synchronizing messages, the code is called partially synchronizing, because the possibility of getting synchronized depends upon the suffix encountered—for some, but not all, synchronization will be impossible. For other codes no suffix has a synchronizing message, so if the decoder gets out of synchronization it returns with probability zero. Such a code is called never synchronizing. It is also called anagrammatic or biprefix since, read backwards, its words must also comprise an exhaustive prefix code, although not necessarily the identical one, as the terminology might suggest. The adjective biprefix [1] is used for any code which is a prefix code in both directions. It is easy to prove that the only exhaustive prefix codes to be never synchronizing are those enjoying a suffix property; no word is the suffix of another word.

Of some interest is the frangibility of synchronizing properties. Two codes differing in small structural ways may have for example one member ergodic, the other anagrammatic. The codes \{01, 000, 100, 110, 111, 0010, 0011, 1010, 1011\}, \{01, 000, 101, 110, 111, 0010, 0011, 1000, 1001\}, whose tree representations differ slightly, provide an example. Ergodic codes are vastly more numerous [8] and the sparse populations of the other varieties add to their interest [10].

Tests are available to determine the synchronizing classification of an arbitrary code, Refs. [2, 5], for example, and in the ergodic case one may calculate the expected synchronizing delay, in number of symbols, for a given code [4]. This will be of interest to us later.
II. PROBLEM DESCRIPTION

What is of interest at present is to examine the possibility of designing codes whose gross structure is pre-specified, so as to influence the synchronizing properties of the result. Specifically, we shall consider unlabeled, exhaustive trees and explore the existence of labelings which achieve established goals of certain target synchronizing properties. And we shall rely more upon the numerical properties of such labelings than upon their symbolic properties.

To begin we may label a given tree with variables, as in Fig. 4, and be concerned about the values of those variables necessary to provide the desired results.

We also emphasize that if such a tree has $n_i$ words of length $\ell_i$, $i = 1, \ldots, m$, then at one time we admit a rather large number of different codes, although in general this will not be the set of all epcs describable by the set of pairs $\{(n_i, \ell_i)\}$. To appreciate this look ahead to Figs. 6 and 7. Both codes have one word of length 2, four of length 3, and four of length 4, yet all labelings of Fig. 6 will not generate all codes with these $n_i$ and $\ell_i$. For example, any labelings of Fig. 7 will produce all longest words beginning with the same symbol, but no labeling of Fig. 6 has this property.

By the very definition of synchronization and the most basic results thereto pertaining, we must be interested in whether or not some words suffix other words.

In Fig. 4, for example, if we ask whether $X_2X_4$ suffixes $X_2X_3X_6$, we ask whether $X_2 = X_5$ and $X_4 = X_6$. Since all the symbols are either 0 or 1 (albeit they are further constrained), we might say that if

$$(X_7 + X_2) \text{mod } 2 + (X_4 + X_6) \text{mod } 2 = 0,$$

where the middle addition is ordinary real addition, then, and only then, does $X_2X_4$ suffix $X_2X_5X_6$.

If we write $X_2 + X_5 = 2i_1 + f_1$, $i_1 = 0$ or 1, and $f_1 \leq 1$, then $f_1 = 1 \iff X_2, X_5$ are different.

Writing, in addition,

$$X_4 + X_6 = 2i_2 + f_2$$

fig. 4. A code with variable symbols.
defining values in the same fashion for $i_2$ and $f_2$, one finds that $f_1 + f_2 \geq 1 \Rightarrow X_2X_4$ does not suffix $X_2X_5X_6$.

To describe the problem of labeling the tree in Fig. 5 so as to obtain an anagrammatic encoding (which is not possible) over $E = \{0, 1\}$, we obtain

\[
\begin{align*}
X_2 + X_5 &= 2i_1 + f_1 \\
X_4 + X_8 &= 2i_2 + f_2 \\
X_4 + X_9 &= 2i_3 + f_3 \\
X_3 + X_5 &= 2i_4 + f_4 \\
X_6 + X_8 &= 2i_5 + f_5 \\
X_6 + X_9 &= 2i_6 + f_6 \\
X_7 + X_8 &= 2i_7 + f_7 \\
X_7 + X_9 &= 2i_8 + f_8 \\
\end{align*}
\]

\[f_1 + f_2 \geq 1\]
\[f_1 + f_3 \geq 1\]
\[f_4 + f_5 \geq 1\]
\[f_4 + f_6 \geq 1\]
\[f_4 + f_7 \geq 1\]
\[f_4 + f_8 \geq 1\]
\[X_2 + X_3 = 1\]
\[X_6 + X_7 = 1\]
\[X_4 + X_5 = 1\]
\[X_8 + X_9 = 1\]
\[f_1, \ldots, f_8 \leq 1\]
\[i_j = 0, 1; \quad x_j = 0, 1.\]

\[\text{FIG. 5. An example of formulation.}\]
In the Xif constraints, we have one constraint always involving two distinct X’s and a unique pair \((i_k, f_k)\), for each pair of symbols which enters into the examination for suffixing pairs.

In the f constraints, there is one for each possible pair (word, longer word) and the corresponding constraint has a number of f’s equal to the length of “word.”

The X constraints, finally, simply require that each non-terminal vertex give rise to both a 0 and a 1 edge.

The system of equations and inequalities may be made smaller and the number of variables decreased if several observations are applied.

1. \(X_6, X_7, X_8, \text{ and } X_9\) do not need to be considered variable. Whether we have \(X_6 = 0, X_7 = 1\) or \(X_6 = 1, X_7 = 0\) the code is the same. Similar comments pertain to \(X_8\) and \(X_9\). Certainly any subtree \(A\) (this symbol is used for the subtree of the same shape, that is, a vertex with branches leading to two distinct vertices below) that does not have descendants itself may be labeled in advance.

2. The number of X variables surviving may be halved and the X constraints eliminated just by noting that
\[
X_3 = 1 - X_2 \\
X_5 = 1 - X_4.
\]
In general that substitution is used in every subtree \(A\) that has a descendant. Utilizing observations 1 and 2 we may represent the code by the strings
\[
X_2 X_4 \\
(1 - X_2) 0 \\
(1 - X_2) 1 \\
X_2(1 - X_4) 0 \\
X_2(1 - X_4) 1
\]
with parentheses used to enclose what will be a single symbol.

The size of the system of constraints and the number of variables both decrease again as the result of two additional observations.

3. Whenever a 0 would be compared to a 0 or a 1 to a 1 there is no Xif equation and neither \(i\) nor \(f\) is defined. The two are equal and do not change the suffixing or non-suffixing of one word by another. For example if \((1 - X_2) 0\) suffixes \(X_2(1 - X_4) 0\) it is only a question of \((1 - X_2) vs (1 - X_4), and 0 vs 0 is negligible.
4. Whenever 0 would be compared to 1 the shorter word clearly does not suffix the longer word and that particular word pair is neglected. That pair of words can give rise to no new variables or constraints.

Applying the four observations to the code of Fig. 5, the conditions (1) may be reduced to

\[
\begin{align*}
X_2 + (1 - X_4) &= 2i_1 + f_1 \\
X_4 + 0 &= 2i_2 + f_2 \\
X_4 + 1 &= 2i_3 + f_3 \\
(1 - X_2) + (1 - X_4) &= 2i_4 + f_4 \\
f_1 + f_2 &\geq 1 \\
f_1 + f_3 &\geq 1 \\
f_4 &\geq 1 \\
f_1, f_2, f_3, f_4 &\leq 1 \\
X_2, X_4 &\geq 0 \text{ or } 1 \\
i_1, i_2, i_3, i_4 &= 0 \text{ or } 1.
\end{align*}
\]

(2)

To reiterate, the system (2) is a set of conditions whose satisfaction is equivalent to the ability to select symbols for the code of Fig. 5 so that no word is the suffix of a different word. The given code structure, as exemplified by Fig. 5, for example, may be optimal in the sense of average word length. That is, a procedure such as Huffman's [3] may have produced the structure. Such algorithms, of course, can only be tied to synchronizing properties in a most indirect and limited way, as noted in [9] as a function of top or bottom merging. Given a best code in terms of average word length, one could proceed to ask for the best synchronizing code realizable from the given structure. That is precisely one of the problems addressed in Section III.

As mentioned, we hope to design codes with certain synchronizing capabilities or to decide the feasibility of certain designs (as in the foregoing example). To do this we shall solve optimization problems with systems similar to (2) as constraints. Accordingly, it would be highly desirable to eliminate the integrality requirements on the \(X\)'s and the \(i\)'s and exploit the efficiencies of the simplex method (since all the constraints are linear), hoping to obtain always integer answers.

Unfortunately, that hope is not rewarded. Typical matrices lack the property of total unimodularity [6], which guarantees integrality of linear programming solutions. If we neglect the integer requirements we obtain,
in general, fractional results that make no sense as code symbols. In the
system (2), for example, let all the \( f \)'s = 1, all the \( i \)'s = 0, and all the \( X \)'s = \( \frac{1}{2} \). On the other hand, with the integer requirements in force, (2) has no solution. It is clear that by requiring the \( f \)'s to be \( \leq 1 \), and if we additionally demand the \( X \)'s to be \( \leq 1 \), then it is sufficient to ask that the \( X \)'s and the \( i \)'s be integers, for no value greater than unity could occur.

Therefore the system (2) may be modified slightly and written as

\[
\begin{align*}
X_2 - X_4 - 2i_1 - f_1 &= -1 \\
X_4 - 2i_2 - f_2 &= 0 \\
X_4 - 2i_3 - f_3 &= -1 \\
-X_2 - X_4 - 2i_4 - f_4 &= -2 \\
f_1 + f_2 &\geq 1 \\
f_1 + f_3 &\geq 1 \\
f_4 &\geq 1 \\
f_1, f_2, f_3, f_4 &\leq 1 \\
X_2, X_4 &\leq 1 \\
i_1, i_2, i_3, i_4, X_2, X_4 &\geq 0 \text{ and integer.}
\end{align*}
\]

(3)

Having provided examples of the formulational scheme, we turn now to a generalized description of it and the embodiment of the result in the form of a theorem.

Formulation Rules

Let \( T \) be the binary tree of an exhaustive prefix code and label its edges in the following way.

(a) If neither successor node of a given node has a successor itself, label one of the two edges 0 and the other, 1, arbitrarily.

(b) For every other pair of edge descendants from nodes, label one edge \( x_i \), the other \( (1 - x_i) \), and use a different index \( i \) for each such pair. Each code word then is represented as a string of symbols, where a symbol is an \( x \) variable, the difference \( 1 - x \) an \( x \) variable, a 0, or a 1.

(c) For each pair of code words \( w_r, w_s \), where \( l(w_r) < l(w_s) \), \( l \) denoting the length of the word, a collection of variables is defined and a number of algebraic equations are written.

These result from symbol-by-symbol comparisons between \( w_r \) and \( w_s \), specifically the first symbol of \( w_r \) is compared to the \((l(w_s) - l(w_r) + 1)\)st
symbol of $w_s$; the second symbol of $w_r$ to the $(l(w_r) - l(w_s) + 2)$nd symbol of $w_s$; up to the last symbol of $w_r$ to the last symbol of $w_s$.

For each comparison of symbols $Z_u, Z_v$, new variables $i_c$ and $f_c$ are defined and an equation

$$Z_u + Z_v = 2t_d + f_d$$

can be written. Symbols that are variables are constrained to be 0 or 1, $i_c$ is constrained to be 0 or 1, and $f_c$, which is necessarily an integer then, is constrained to be no larger than 1.

For each comparison of word pairs, let $D = \{d | f_d$ was used in the equations from this comparison$\}$ and write an inequality

$$\sum_{D} f_d \geq 1.$$

Several circumstances that may arise in step (c) reduce the number of variables.

(i) If neither $Z_u$ nor $Z_v$ is a variable and $Z_u = Z_v$, then no equation is written and no new $i$ and $f$ variables are defined.

(ii) If neither $Z_u$ nor $Z_v$ is a variable and $Z_u \neq Z_v$, then omit all variables, equations, and the inequality that would normally be produced for this word pair.

(iii) If the comparison of a particular pair of symbols has arisen in the comparison of some other word pair, then use the $i$ and $f$ variables previously defined in lieu of introducing new ones.

**Theorem 1.** The tree $T$ admits a labeling that provides an anagrammatic code if and only if the linear system of equations and inequalities in the Formulation Rules together with the variable constraints enumerated there has a solution.

**Proof.** By definition, such a $T$ gives rise to an anagrammatic code if and only if no word is the proper suffix of another. Further, a solution to the system in the Formulation Rules is equivalent, by construction, to the absence of any word $w_r$ suffixing any longer word $w_s$.

### III. Data Processing, Specific Problems, and Results

1. **Generating the Problems to Solve**

It is evident from the example of the last section that writing the relationships for even trees of very modest size is a significant task.
Accordingly, it is accomplished by computer, using a PASCAL program. The inputs to this program are strings which convey both the variable and the fixed symbols of the code. For example, applying observations 1 and 2, the structure of Fig. 5 gives the words

\[
\begin{align*}
2 & \quad 4 \\
2 & \quad -4 \quad 0 \\
2 & \quad -4 \quad 1 \\
-2 & \quad 0 \\
-2 & \quad 1.
\end{align*}
\]

The program recognizes 0 and 1 as fixed code symbols, and for \( n > 1 \) associates \( x_n \) with \( n \) and \( 1 - x_n \) with \( -n \). It writes a system as (3) into a data set, from which it may be read for the process of problem solution.

The optimization program used is LINDO [7], a commercially available, interactive package for solving linear programming problems with the simplex method, integer programming problems by a branch and bound [6] algorithm, and quadratic programming problems.

2. Creating Anagrammatic Codes

In some optimization problems where certain variables must be integers it is possible to ignore at least some of those requirements and invent an objective to nudge the process toward an integer solution. If that could be done with a linear objective here, we would need have recourse only to the simplex method, which is much more efficient than any integer programming routine.

But, as illustrated in the previous section, finding a labeling that provides a never-synchronizing code is just a question of satisfying a set of linear relationships, that is, of finding a feasible solution to a problem. Given the scarcity of these codes, we have always just a search for a particular sort of feasible solution, and no linear objective will change that.

Furthermore, until an initial integer-feasible solution is discovered, the branch and bound logic cannot exploit objective function value to sharpen the search. Since an initial integer feasible solution is the object of interest here, the choice of objective function is irrelevant. Hence, a simple one will be employed.

It should also be mentioned that whereas no linear objective function will induce integrality of the \( i \)'s and \( x \)'s, a quadratic one will. Suppose we take as an objective

\[
\max \sum (x_k - \frac{1}{n})^2 + \sum (i_j - \frac{1}{n})^2.
\]
Any integer solution to (3) will give a larger value to the function in (4) than will any non-integer solution. Therefore, suppose we solve the quadratic programming problem (4) subject to the constraints (3) but neglecting integrality. If an integer optimum is obtained it determines an anagrammatic code. A non-integer optimum implies that no such code corresponds to the tree which yields (3). But this may be an unreliable way to solve the problem, because it requires maximizing a convex function over the convex feasible set. A QP algorithm could converge to the minimum, a solution of no interest.

The IP approach is entirely reliable, of course, but large code structures beget large IPs and the computation time required to determine a code or that none corresponds to an input structure may be significant. The time to generate the IP, on the other hand, is of no consequence. It is small. Let us describe the results of two experiments.

First we take a familiar structure which does admit a never-synchronizing labeling.

The set of constraints corresponding to Fig. 6 is omitted to conserve space. The goal is to locate a feasible solution to the system with all variables $x$ and $i$ required to be integers, so the objective function is immaterial. In the example it was taken to be $\max x_2$.

The solution discovered was the one in which the left branches in Fig. 6 are labeled 0, and the right branches, 1. The code then is $\{01, 000, 100, 110, 111, 0010, 0011, 1010, 1011\}$, in which it is apparent that no word suffixes another word.

It should be noted that the question, “Is there an anagrammatic binary encoding with one word of length 2, four of length 3, and four of length 4?” is broader than the one we are able to answer. For example, Fig. 7 belongs to that class, but one cannot label its branches so as to yield a never-synchronizing code.

Under the present scheme a collection of IPs might be required to answer the query about (number, length) pairs. Whether this can be accomplished within a single optimization problem remains to be seen.

As a second example the structure of Fig. 7 was the input and LINDO reported the absence of a feasible solution to the integer problem.

![Fig. 6. A structure that yields anagrammatic codes.](image-url)
3. Minimizing Synchronizing Delay

While the search for anagrammatic codes is interesting the "yes, no" nature of the conclusions is unexciting. We proceed now to describe a different and more challenging optimization problem.

By taking the view that the ergodic property is desirable, one concludes that it must be desirable to regain synchronization once it is lost, as rapidly as possible in some sense. Assuming we have a measure for that appropriate sense it is reasonable to seek a "best" ergodic code derivable from a given structure.

To introduce the concept of synchronizing delay, the requisite computations, and some possible performance measures, reference is made to [1].

As Fig. 3 illustrates, when synchronization is lost, one encounters suffixes of intended words, and since codes of interest to us are exhaustive each of these strings is also a prefix. Encountering the null prefix, $\phi$, means that synchronization is regained. Assuming $W$ elements have stationary probabilities of occurrence, the process of decoding erroneously until eventually getting back in step with the intended message has a convenient Markov chain representation: the states are the proper prefixes of $W$ along with $\phi$, and transitions occur as follows. If in state $c$ and $w$ is the intended word following, decode $aw$ and if $p$ is the remaining string, then the process is in state $p$. Given word probabilities, all transition probabilities are known.

If $\pi_1, \pi_2, \ldots, \pi_k$ are the proper prefixes of $W$, then the transition matrix $M$ for $W$ would appear as in Fig. 8, and $M_{ij}$ is the probability of a transition from state (prefix) $i$ to state (prefix) $j$ at any time. If $P_\ell$ is the stationary probability of word $\ell$, then

$$M_{ij} = \sum_{\ell \in L} P_\ell,$$

where $L = \{\ell | \text{decoding } \pi_i w_\ell \text{ leaves the prefix } \pi_j\}$.

If $T$ denotes the submatrix corresponding to proper prefixes alone, then $(I - T)^{-1}$ is the average number of transitions to reach $\pi_i$ from $\pi_i$. $r_i = \sum_j (I - T)^{-1}_{ij}$ is the average number of transitions from state $i$ to other non-
synchronizing states. What, then, represents a measure of synchronizing performance of the $n$-word code? Is it the expected value of the $r_i$, $\sum_{i=1}^{n} p_i r_i$? The arithmetic average of them, $\frac{1}{n} \sum_{i=1}^{n} r_i$? The largest of them, $\max\{r_1, r_2, \ldots, r_n\}$? Some other function? The quantity in the heading of this subsection, "synchronizing delay," it is seen, may be measured reasonably in a variety of ways.

The choice must reflect the objectives of the particular application, but it is apparent that our design vehicle, an algorithm for solving linear optimization problems in the presence of integer constraints, cannot accommodate $(I-T)^{-1}$ and its various by-products. Our design tool is capable only of characterizing the composition of each word and, by a formulation trick, the suffixing properties of the code. But since synchronization is intimately connected with suffixing, perhaps that is sufficient. The approach is to invent an objective function, the product of heuristic considerations, and to obtain codes optimal with respect to that measure. The results will be evaluated by enumerating or sampling all possible codes for a number of examples and comparing properties of the optimization-provided optimal code with the best properties obtained from the enumeration.

The Markov model provides an alternative method for solving our problem: one can produce all possible labelings of the given structure, perform the above computation for each possible code, and select the one giving the best result. The computation is lengthy, though, for each instance, and the combinatorial number of possible labelings makes this strategy undesirable.

Synchronization is closely related to suffixing, and the variables of our formulation characterize completely all instances of suffixing. It is easy to generate reasonable heuristics and as the sample results will establish, these achieve the desired code results in a most gratifying fashion.

If out of synchronization, what returns one rapidly to that state? It is encountering, as the suffix presented to the decoder, a string that is itself a word. We want, therefore, to promote the number of suffixing instances.
Moreover, it is desirable that higher probability words be those with other words as suffixes.

Now the variables \( f_j \) inform one exactly of the matching or mismatching of symbols. Consider, for example, words \( X_1X_2 \) and \( X_3X_4X_5X_6 \). Comparing these two produces the equations

\[
X_1 + X_5 = 2i_1 + f_1
\]
\[
X_2 + X_6 = 2i_2 + f_2.
\]

It is recalled that with \( f < 1 \) we get \( f = 0 \) if and only if the two symbols match. But suffixing is just this matching: in the example above \( X_1X_2 \) is a suffix of \( X_3X_4X_5X_6 \) if and only if \( f_1 = f_2 = 0 \).

Let us see how we might contrive an objective function to provide incentive toward a quickly resynchronizing code.

First, a word suffixes another when and only when all the \( f \)'s corresponding to the comparison are zero. If \( \{ f_i | i \in I \} \) is such a set, then demanding \( y \geq f_i, i \in I \), and inducing \( y = 0 \) would cause a word to suffix another. Minimizing a function of the \( y \)'s would be a strategy for making some \( y \)'s zero, then, and forcing some words to suffix others.

But the probability that the longer word is transmitted is also crucial—if it is seldom sent it is of lesser importance than having a more probable word possessing a word suffix. Therefore, \( y \) should be weighted (assuming the objective is to minimize) with an increasing function of the probability of the longer word.

To generate a sample objective function we return to the code of Fig. 5 and imagine the word probabilities to be \( \{0.25, 0.25, 0.25, 0.13, 0.12\} \) with the length two words having probability 0.25. Here, as in all the examples, the probabilities used will give the structure assumed if Huffman's procedure is used. The resulting system (2) indicates the objective function's formation. We show all possible comparisons and the \( f \)'s corresponding to each.

<table>
<thead>
<tr>
<th>Shorter word</th>
<th>Longer word</th>
<th>Corresponding ( f )'s</th>
</tr>
</thead>
<tbody>
<tr>
<td>( X_2X_4 )</td>
<td>( X_2(1 - X_4) )</td>
<td>( f_1, f_2 )</td>
</tr>
<tr>
<td>( X_2X_4 )</td>
<td>( X_2(1 - X_4) )</td>
<td>( f_1, f_3 )</td>
</tr>
<tr>
<td>( 1 - X_2 )</td>
<td>( X_2(1 - X_4) )</td>
<td>( f_4 )</td>
</tr>
<tr>
<td>( 1 - X_2 )</td>
<td>( X_2(1 - X_4) )</td>
<td>No suffix possible</td>
</tr>
<tr>
<td>( 1 - X_2 )</td>
<td>( X_2(1 - X_4) )</td>
<td>No suffix possible</td>
</tr>
<tr>
<td>( 1 - X_2 )</td>
<td>( X_2(1 - X_4) )</td>
<td>( f_4 )</td>
</tr>
</tbody>
</table>
Now, of course, the anagrammatic inequalities upon the $f$'s disappear from the problem, which is shown formulated in (5),

$$\text{Min } 0.13y_1 + 0.12y_2 + 0.13y_3 + 0.12y_4$$

ST

$$x_2 - x_4 - 2i_1 - f_1 = -1$$

$$x_4 - 2i_2 - f_2 = 0$$

$$x_4 - 2i_3 - f_3 = -1$$

$$-x_2 - x_4 - 2i_4 - f_4 = -2$$

$$y_1 - f_1 \geq 0$$

$$y_1 - f_2 \geq 0$$

$$y_2 - f_1 \geq 0$$

$$y_2 - f_3 \geq 0$$

$$y_3 - f_4 \geq 0$$

$$y_4 - f_4 \geq 0$$

$$f_1, \ldots, f_4 \leq 1$$

$$x_2, x_4 \leq 1$$

Whereas problem (5) and the structure of Fig. 5 do not offer many different choices of coding symbols, the formulation proves effective: the solution $X_2 = X_4 = 0$ was discovered to be optimal. The corresponding code, then, is $\{11, 10, 00, 010, 011\}$. This code (along with its symmetric mate, $X_2 = X_4 = 1$) provides the maximum number of suffixing instances and is superior to the other possibilities.

If accomplished manually, creating the objective function would be a terrific task for a large code. It is easy, however, to let the computer do this, too, at the time the constraint system is being generated and the $f$'s are being defined.

For the next examples it will be natural to ask how good our “optimal” code is relative to some of those measures which might be used as reasonable indications of synchronizing performance.

It is proposed to make this determination empirically, by enumerating the labelings of a particular structure, discovering a best code relative to reasonable measures, and comparing it to the code found by solving the optimization problem. In the experiments to be described those measures
are the maximum expected synchronizing delay over all prefixes and the average of the expected synchronizing delays.

In a formulation with \( m \) of the \( X \) variables there are \( 2^m \) (not necessarily producing distinct encodings) labelings and a computer program allows one to examine every \( k \)th code for \( k \geq 1 \), so that the enumeration may be exhaustive or a random sample if \( m \) prohibits an exhaustive search. Labelings are generated in the following order where the \( j \)th symbol represents the value of \( X_j \) in the labeling

\[
111 \ldots 1 \\
011 \ldots 1 \\
101 \ldots 1 \\
001 \ldots 1 \\
110 \ldots 1 \\
010 \ldots 1 \\
100 \ldots 1 \\
000 \ldots 1 \\
001 \ldots 1, \text{ etc.}
\]

Surely the \( k \)th item from an ordered list hardly constitutes a random sample, but as synchronizing parameters cannot be related directly to labeling, the sample is sufficiently random. For our examples exhaustive enumerations have been used in place of samples. Actually, only half the list need be enumerated since the first half is just the symmetric image (1 for 0 and 0 for 1) of the second half, code by code, and the synchronizing characteristics must be identical. Note that our two optimal codes from the first example comprise a symmetric pair.

For the structure of Fig. 6 every labeling was generated and the probabilities are \( \{0.28, 0.13, 0.13, 0.13, 0.05, 0.05, 0.05, 0.05, 0.00\} \) for the nine words in ascending order on length. For non-ergodic codes the matrix \((I-T)\) is singular, and both our example measures would be infinitely large. Subject to round-off error, they have inverses which occasion very large values for the two numbers (about \(10^{16}\) in this example). These, of course, are useful flags for the non-ergodic codes generated. Over all 32 labelings 1.9991 words was the minimum value of the max expected synchronization delay. This value was assumed for the symmetric pair of codes \( \{10, 110, 000, 010, 011, 1110, 1111, 0010, 0011\} \), \( \{01, 001, 111, 100, 101, 0000, 0001, 1100, 1101\} \). The largest value for this measure was 7.0948 words, which illustrates the extensive variation of synchronizing properties of codes resulting from one structure. The minimum average synch delay discovered was 1.8138 words. It occurred for the same two codes which minimized the
maximum delay. The largest value for this parameter was 6.4155, and it occurred for the same code found to be worst in the maximum delay sense. In fact if we put the pairs of our sample trial measures in order on the maximum synch delay, only two of the average synch delay values are out of natural order. Thus there is a strong connection between the two measures—at least for this example.

LINDO solved our integer programming problem and discovered the second of the symmetric pair above as its optimal solution. It was the third feasible integer solution reported. Relative to our two reasonable measures, then, our formulation found a best synchronizing encoding with very little computational effort.

The final example is derived from the structure of Fig. 9. It has 8 variables, 13 words, and 4 subtrees A where we arbitrarily take left = 0, right = 1, without affecting the results. The 8 variables were assigned in order $X_2 \cdots X_9$ to subtrees $A$ from top to bottom and left to right. The formulation yields 67 integer stipulations for there are 59 $i$ variables and 8 $X$'s. All $2^8 = 256$ possible codes were generated and, again, their maximum and average synchronizing delays calculated. The smallest values are 2.3904 and 2.1785, respectively. They occur for the same code and, except for the symmetric image, are unique. To portray the spectrum of values possible, Table I presents the distribution for the maximum synchronizing delay. (The average is distributed comparably and is not tabulated here.) In the table is found the number of codes whose maximum synchronizing delay is $<$ the corresponding bound and $>$ the bound for the preceding class. The best code discovered was the fourth integer solution produced by LINDO. This was after 1 min of computing. Following 4 min of computing, no bet-

![Fig. 9. A 13-word structure with corresponding word probabilities.](image-url)
TABLE I
Distribution of Max Synchronizing Delay from Fig. 9

<table>
<thead>
<tr>
<th>No. codes</th>
<th>Bound</th>
<th>No. codes</th>
<th>Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>2.59</td>
<td>4</td>
<td>4.99</td>
</tr>
<tr>
<td>24</td>
<td>2.79</td>
<td>24</td>
<td>5.19</td>
</tr>
<tr>
<td>16</td>
<td>2.99</td>
<td>8</td>
<td>5.39</td>
</tr>
<tr>
<td>74</td>
<td>3.19</td>
<td>14</td>
<td>5.59</td>
</tr>
<tr>
<td>10</td>
<td>3.39</td>
<td>6</td>
<td>5.79</td>
</tr>
<tr>
<td>4</td>
<td>3.59</td>
<td>2</td>
<td>5.99</td>
</tr>
<tr>
<td>8</td>
<td>3.79</td>
<td>4</td>
<td>6.19</td>
</tr>
<tr>
<td>10</td>
<td>3.99</td>
<td>2</td>
<td>6.39</td>
</tr>
<tr>
<td>12</td>
<td>4.19</td>
<td>0</td>
<td>6.59</td>
</tr>
<tr>
<td>12</td>
<td>4.39</td>
<td>2</td>
<td>6.79</td>
</tr>
<tr>
<td>18</td>
<td>4.59</td>
<td>0</td>
<td>6.99</td>
</tr>
<tr>
<td>12</td>
<td>4.79</td>
<td>30</td>
<td>18.0925</td>
</tr>
</tbody>
</table>

A reviewer has suggested the presentation of results in theorem form, and whereas the search for a best code has had to proceed heuristically, it is still possible to state an exact result for a general sort of problem.

**Theorem 2.** Assume that the synchronizing performance of an exhaustive prefix code may be expressed by a function of variables which tell exactly which words are suffixes of which other words. Then a best synchronizing code for a given tree T may be obtained by solving an integer programming problem whose objective function is the known performance measure, and whose constraints are given by the Formulation Rules, excluding those of the form $\sum_D f_d \geq 1$.

**Proof.** The suffixing information mentioned is exactly what the $f$ variables give, and the constraints taken exclude those that prohibit suffixing.

The difficulty with Theorem 2, of course, is that to obtain a solvable problem one requires a friendly objective function, which is unlikely to correspond to any conventional measures of synchronizing goodness.
IV. Conclusions

The optimization approach to code design with synchronizing capabilities has worked well. Since the anagrammatic property is simply a yes–no proposition, one can only credit the formulation with the ability to represent codes mathematically and accurately and therefore discriminate on the basis of the existence or absence of a feasible solution. It is hoped that further study of the formulations obtained might provide a new and better characterization of these scarce but interesting examples.

The problem of constructing codes with superior synchronizing capabilities is clearly more challenging and the examples provide strong evidence for the effectiveness of the procedure shown. Earlier we cited the objective of finding a best code in terms of synch properties once a tree structure (possibly representing a best code in terms of average work length) is given. Our method has been heuristic and, as such, cannot guarantee meeting that objective. Nevertheless, examples show it to come very close when it does not succeed, and it is the only scheme other than the total enumeration of possibilities that has been devised for the problem.

The sole disadvantage is problem size, and it will be worthwhile to seek alternative incentive functions that obviate the need for the large number of \( f, y \) constraints. A significantly more compact formulation worked as well for the first two examples, but failed badly for the larger third example. It weighted \( f_i \) by the number of occurrences multiplied by the probabilities of the longer word, but without the ability to bind at 0 all the \( f_i \)'s corresponding to a two-word comparison; a good objective function value can still correspond to a small number of suffixings. It is the \( f, y \) constraints that achieve the concerted binding. Still, easier formulations may be available. Certainly a reduction in the number of integer variables would make for faster solutions.

As mentioned previously, the strongest limitation of results is that a specific structure only may be treated. It is well known that a set of probabilities may have several sets of word lengths corresponding to optimal (in the sense of average word length) codes. Whether or not word-length optimality is a concern it would be much more valuable and interesting to seek the best synchronizing code that has one word of length 2, four of length 3, three of length 4, one of length 5, and four of length 7, for example, rather than to settle for Fig. 9, which represents some but not all the codes answering that description. This more general problem is a logical successor to the present work.
ACKNOWLEDGMENTS

The author is indebted to Shawn DePierri for her diligence in programming the complex string manipulations necessary to generate the optimization problems and to compute synchronizing delays for purposes of comparison. Furthermore, a referee is due considerable credit for his constructive suggestions.

REFERENCES

8. SCHUTZENBERGER, M. (1956), On an application of semigroup methods to some problems in coding, IRE Trans. IT-2, 47-60.
9. SCHWARTZ, E. (1964), An optimum encoding with minimum longest code and total number of digits, Inform. and Control 7, 1, 37-44.