CODE AND PARSE TREES FOR LOSSLESS SOURCE ENCODING*

JULIA ABRAHAMS†

1. Introduction. This paper surveys the theoretical literature on fixed-to-variable-length lossless source code trees, called code trees, and on variable-length-to-fixed lossless source code trees, called parse trees. Huffman coding [1] is the most well known code tree problem, but there are a number of interesting variants of the problem formulation which lead to other combinatorial optimization problems. Huffman coding as an instance of combinatorial search has been highlighted in the books by Ahlswede and Wegener [2] and Aigner [3]. See also the papers of Hinderer and Stieglitz [4], Hassin and Henig [5], and Nagaraj [6] for overviews of the combinatorial search literature. Tunstall parsing [7] is the most well known parse tree problem for a probability-based source model, although parsing based directly on source data is very familiar as Lempel Ziv parsing [8-9], a family of techniques which is outside the scope of this survey. Similarly, adaptive, data-based variants of Huffman coding, e.g. [10-13], will not be treated here. Also not treated here is the theoretical computer science literature in which the source probability model is suppressed, as in the book by Berstel and Perrin [14], and see the recent survey by Brüyere and Latteux [15]. Rather, the assumption here is that the source model is given as a sequence of independent and identically distributed (i.i.d.) random variables for some known discrete distribution, although on occasion it is possible that only partial information about the source is available. These lossless source encoding techniques comprise a subset of data compression techniques, and broader surveys of the data compression literature are available [16-24].

In particular, the following code tree topics are outlined in Section 2: characteristics of the Huffman code tree; Huffman-type coding for infinite source alphabets and universal coding; the Huffman problem subject to a lexicographic constraint, or, that is, the Hu-Tucker problem; the Huffman problem subject to maximum codeword length constraints and other constrained coding problem variants; code trees which minimize other functions besides average codeword length; coding for unequal cost code symbols, or, that is, the Karp problem, and finite state channels; and variants of Huffman coding in which the assignment of 0's and 1's within codewords is significant, such as bidirectionality and synchronization. The literature on parse tree topics is less extensive. Treated in Section 3 are: variants of Tunstall parsing; dualities between parsing and coding; dual tree coding in which parsing and coding are combined to yield variable-length-to-variable-length codes; and parsing and random number gen-

†National Science Foundation, Computer & Information Sciences & Engineering, 4201 Wilson Blvd., Arlington, VA 22230 USA, Email: jabrahams@nsf.gov
eration. Finally, questions related to counting and representing code and parse trees are discussed in Section 4.

2. Code Trees. In fixed-to-variable-length binary source coding, a code tree is used to associate a string of binary code symbols with each of a set of $K$ source symbols. A code tree is a complete binary tree with $K$ leaf nodes and $K - 1$ internal nodes including the root at the top of the tree. Pairs of branches descend from each internal node. Left branches are labeled with the code symbol 0 and right branches are labeled with the code symbol 1. Each leaf node is labeled with one of the $K$ source symbols, $s_1, \ldots, s_K$. See Figure 1 for an example. A path through

![Figure 1. Example of a binary code or parse tree.](image)

the code tree from the root to a leaf describes the string of code symbols associated with the source symbol at the leaf. The problem is to find a code tree to optimize some performance criterion, possibly under the imposition of constraints on the form of the tree. For $r$-ary source coding, when $r > 2$, each internal node of the tree can have anywhere between 2 and $r$ branches descending from it, and this flexibility introduces some additional complexity to the $r$-ary code tree problem. Nevertheless, much of the literature concentrates on binary code trees with the $r$-ary generalization straightforward.

2.1. Huffman Code Trees. Huffman coding [1] can be found described in all information theory textbooks [25-28], as well as in many computer science [29-33] and discrete mathematics texts [34-35]. Zimmerman [36] also derived the Huffman algorithm independently. Huffman's algorithm solves the quintessential problem of finding the code tree to minimize average codeword length. For source symbols occurring with probabilities $P = \{ p(i), i = 1, \ldots, K \}$ we wish to find path lengths through the tree $L = \{ l(i), i = 1, \ldots, K \}$ to solve $\min_L \sum_{1 \leq i \leq K} p(i) l(i)$. There are no further constraints on the problem; however, it is known from the Kraft inequality that for a set of codeword lengths $L$ to be compatible with a binary code tree it is necessary and sufficient that $\sum 2^{-l(i)} \leq 1$ hold. Compatibility is in the sense that paths of lengths given by the codeword lengths reach unique leaf nodes of the tree. Because the re-
resulting minimum average codeword length \( l^* \) can be bounded in terms of entropy
\[ H = -\sum p(i) \log_2 p(i) \] according to \( H \leq l^* < H + 1 \) with \( l^* = H \) if and only if \( P \) is a
dyadic distribution, that is \( p(i) = 2^{-l(i)} \) for some integer \( l(i) \), minimum redundancy,
or \( l^* - H \), is bounded according to \( 0 \leq l^* - H < 1 \). An equivalent perspective on
Huffman coding is that it minimizes redundancy over \( L \) because \( H \) is a constant with
respect to the minimization. Another equivalent perspective on Huffman coding is
that the dyadic distribution induced by the code tree \( Q = \{ q(i) = 2^{-l(i)}, i = 1, \ldots, K \} \)
is the dyadic distribution closest to \( P \) in the minimum discrimination sense. That is
to say, Huffman coding solves \( \min_L \sum p(i) \log_2 p(i)/q(i) \).

Huffman’s algorithm is a bottom up merge type algorithm in which, at each stage
of the process, the two least probable source symbols are “merged” into a pair of
sibling nodes whose new parent node has probability given by the sum of the two
original probabilities. These combined nodes are treated as a single node in each
subsequent stage of the algorithm.

Note that the code tree generated by the Huffman algorithm is not unique, and
that there are code trees which share the same set of codeword lengths as a Huffman
code but which cannot be obtained from the Huffman algorithm. These distinctions
will not be important here.

2.2. Characterizations of Huffman Code Trees. A number of authors have
addressed the problem of characterizing the form of the resulting Huffman code tree
given partial or full information about the source probabilities. These are results about
the form of the Huffman code tree which are known without actually constructing the
code. Katona and Nemetz [37] upper bound the length of the codeword associated
with a source symbol of probability \( p(i) \) based on the extremal properties of the
Huffman code tree for a particular source distribution involving ratios of Fibonacci
numbers. They are motivated to discover a relationship between the self-information
of the source symbol, \( -\log_2 p(i) \), and its codeword length, \( l(i) \), in view of the entropy
bounds on average codeword length. Other work on individual Huffman codeword
lengths has also appeared [38-46]. Schack [47] continues Katona and Nemetz’s line
of inquiry by bounding the probability of source symbols whose self-information and
codeword lengths are far from each other. Cheng et al. [48] make use of Schack’s
machinery in their work on bounding the temporary expansion possible when Huffman
coding is used in lieu of block coding and low probability source symbols precede high
probability source symbols in encoding a sequence of source symbols. De Prisco
and De Santis [49-50] further address the data expansion problem by making use of
the literature on redundancy bounds for given partial information about the source
probabilities. Shen and Gill [51] provide the tightest bound on data expansion to
date, coming close to the surprisingly challenging conjecture of [48].

When the given source symbol is known to be the most likely symbol, with proba-
bility, say, \( p(1) \), its codeword length can be specified to within 1 [38-39,52]. Results
of this type have typically been employed to bound the redundancy of the Huffman
code in terms of known \( p(1) \) more tightly than to within \([0, 1) \). The most comprehen-
sive paper on redundancy bounds is Manstetten’s [53]. The problem of redundancy
bounds originated with Gallager [10], and a number of other authors contributed to this problem [54-61] including the problem variant in which the least likely probability, \( p(K) \), is known. A useful general approach to establishing redundancy bounds given a subset of the ordered source probabilities is Yeung’s [62] redundancy theorem, and several of the redundancy bounds are special cases of his result. Swasek and Wali [63] have conducted a Monte Carlo simulation experiment on redundancy values.

Golomb [64] considers finite source distributions for which multiple distinct code trees share a common minimum average codeword length.

Geckinli [65] gives necessary and sufficient conditions on the source probabilities \( P \) for the difference between the longest and shortest codeword lengths to be 0 or 1. Katona and Lee [38] give a sufficient condition such that the difference is less than or equal to a given integer. In a sense, Geckinli addresses the question of, when is the short, fat tree optimal? In contrast, Katona and Nemetz’s [37] work can be interpreted as addressing the question of, when is the long, thin tree (with codeword lengths \( \{l(i) = i, i = 1, \ldots, K - 1; l(K) = l(K - 1)\} \) optimal? Vinokur [66] addressed this question independently as well.

Some authors are concerned with the balance properties of Huffman trees, that is, with the relative size of the weights associated with the internal nodes of the tree [67-69].

For some parametric families of source distributions, the form of the resulting Huffman code tree is known explicitly. The binomial distribution arises in encoding fixed length blocks of binary source symbols from an independent and identically distributed sequence of biased coin tosses. Jakobsson [70] gives the Huffman code in explicit form for binomial parameters satisfying certain conditions. Stubley [71] considers the binomially distributed source for all parameter values and analyzes the redundancy of the Huffman code in this case obtaining redundancy bounds, and Szpankowski [72] is able to obtain a precise asymptotic analysis of redundancy for large block length. See also [73] for independent work on the binomial distribution. Stubley [74] also examines the multinomial distribution arising from fixed length blocks of \( r \)-ary source symbols. Other authors who have examined binomial sources have been interested in questions about the behavior of the resulting average codeword length as a function of block size, and in particular about its nonmonotonicity with increasing block size. See [52, 75-77].

Similar questions related to Huffman coding variable length strings rather than fixed length blocks can be thought of as arising within the framework of variable-to-variable-length or dual tree coding, treated in a later section.

Another distribution that has been studied in connection with Huffman coding is the finite geometric distribution. Huffman coding for this distribution arises in the context of group testing, a combinatorial search problem in which the goal is to fully classify all members of a set into one of two classes based on a series of tests, each of which can determine whether or not the subset submitted to it consists entirely of members of one of the classes. Hwang [78] finds the average codeword length for this distribution without explicitly identifying the codeword lengths. The codeword
lengths are given explicitly for some parameter values only in [79]. Other work on this problem appears in [5, 80].

There are also a few other parametric families of finite source distributions for which Huffman codes are available. The uniform distribution always leads to a short, fat Huffman tree as in Geckinli [65]. Sandelius [81] observed this explicitly. Günther and Schneider [82] and Campbell [83] discuss the “typical” source of K symbols and compare its entropy to that of the uniform source. The difference is quite small for large K suggesting that, for large, typical sources, the short, fat tree is nearly optimal. And of course the Huffman code trees for dyadic distributions are immediate by inspection. These Huffman codes have zero redundancy and the interesting property that 0’s and 1’s occur with equal probability on average in the string of code symbols, a property which does not hold in general [84]. The finite zeta-function distribution in which \( p(i) \) is proportional to \( 1/i \) is studied by Gutman [85] who shows that for certain finite source alphabet sizes the Huffman code tree can be found explicitly for this distribution. Tucker [86] examines probabilities proportional to \( i, i = 1, \ldots, K \).

Hwang [87] considers the case that the probabilities take on either of only two values. Wang [88] extends this work to certain cases in which the probabilities take on any of only three values. A distribution of particular form arising in a data structure application is explicitly Huffman coded in [89]. There are also a number of infinite source distributions for which explicit minimum average length codes are known, and these will be treated in the next section.

Gallager’s sibling property [10] states that a code tree is a Huffman tree if and only if the nodes of the tree can be listed in order of nonincreasing probability with each node being adjacent in the list to its sibling. This property can be used to verify whether or not a given tree is the Huffman tree for a fixed distribution.

Longo and Galasso [90] employ the minimum discrimination perspective for Huffman coding and find conditions on source probability distributions such that they share the same Huffman code as a particular dyadic distribution. See also [91]. Discrimination arises also in the problem of evaluating the mismatch when a source with one set of probabilities is mistakenly encoded assuming a different set of probabilities [92-94]. It may be of interest to examine mismatch problems in other coding and parsing contexts as well.

Hu and Tucker [95] and Hwang [96] consider the question of computing the average codeword lengths of Huffman codes for different source probability distributions without actually constructing the codes. They give inequalities on functionals of the source probabilities sufficient for inequalities on the Huffman average codeword lengths. Their approaches are related to the notion of majorization in the theory of inequalities.

Hwang’s [96] paper is interesting in another respect. He deals with Huffman code trees more general than r-ary. The number of branches descending from each internal node is a fixed integer varying from node to node. See also Chu and Gill [97] as well as Yeung [98] for trees with variable internal node degrees for the Huffman problem. Both Hwang and Chu and Gill also address the Hu-Tucker problem discussed later.
2.3. Infinite Source Alphabets and Universal Coding. Since the Huffman algorithm is bottom up, beginning with the merge of the two least likely source symbols, infinite source alphabets are not immediately amenable to Huffman coding. However, Gallager and Van Voorhis [99] were able to make use of the Huffman code for a finite distribution obtained from the infinite distribution together with a limiting argument and obtain the minimum average codeword length code for the infinite geometric distribution. Their work extends the earlier work of Golomb [100] on this problem. Note that the geometric distribution arises in describing the run lengths generated by a sequence of underlying i.i.d. binary random variables. Similar methods have been applied to other parametric families of infinite source distributions or to establishing sufficient conditions on the source probabilities for a particular structured infinite tree to be optimal for that distribution [101-104]. Linder et al. [105] prove that this general approach will find the optimal code whenever the source entropy is finite, however it is not truly a constructive method. Kato [106] has extended the proof to infinite entropy distributions. Several authors, for example [107-112], use the codeword sets of Golomb and Gallager and Van Voorhis, or a modification of these sets, for other infinite source distributions besides geometric, not necessarily in an optimal fashion.

Gallager and Van Voorhis’s infinite code trees, parameterized by an integer that depends on the parameter of the geometric distribution, are not universal in the sense of Elias [113]. Universal codes for infinite alphabets have the property that when the shortest codewords are used to represent the most likely source symbols, the resulting average codeword length is upper bounded by a constant multiple of the source entropy for all infinite source distributions.

A number of authors discuss infinite code trees in the context of universal codes including [114-127]. Many of the universal code trees discussed in the literature fall into a common framework as described in [21, Sect. 3.3] based on systems of numeration [128]. Some specific examples of universal codes based on numeration systems are identified in [129-131] and independently in [132]. The general idea is that each positive integer \( x \) is assigned a binary codeword composed of two components consistent with the existence of a binary code tree. An infinite sequence \( V = \{ V_i, i = 1, 2, \ldots \} \) is specified and \( x \) is represented relative to \( V \) by the pair \( (j, r) \) where \( \sum_{i < j} v_i < j < \sum_{i < j+1} v_i \) and \( r = x - \sum_{1 \leq i \leq j} v_i - 1 \). The concatenation of a string of \( j \) bits for \( j \) and a string of \( \lfloor \log v_j \rfloor \) bits for \( r \) makes up the codeword. Of course nonuniversal infinite alphabet code trees can also be interpreted in terms of the two components of a numeration system, particularly when the convention for the representation of \( r \) is loosened to include other variable length strings. There are a variety of other approaches to universal coding that do not fall into the infinite code tree perspective and are not discussed here.

2.4. Lexicographic Constraints on the Huffman Coding Problem: The Hu-Tucker Problem. Suppose that the source symbols to be encoded must be assigned to the leaves of the code tree in a fixed left-to-right order. Equivalently the binary codewords must exhibit a particular lexicographic order. It may happen
for a particular set of source probabilities and a particular linear order constraint on the corresponding symbols that the Huffman code tree (or a tree with the same average codeword length in the case that the minimum average codeword length tree is not unique) exhibits the desired linear order. This is the case, for example, if the source symbols are linearly ordered according to a monotonically increasing or decreasing order on their probabilities. But, in general, the Huffman code tree does not exhibit the desired linear order, and the code tree which minimizes average codeword length subject to a linear order constraint on the source symbols has a resulting average codeword length greater than the Huffman minimum. Such alphabetic or lexicographic or linearly ordered code trees can be obtained using the algorithm of Hu and Tucker [133-138], and their average codeword lengths fall within \([H, H + 2]\).

The Hu-Tucker algorithm has as its central step a merge operation like the Huffman merge but restricted to pairs of symbols which are adjacent in the linear order in a certain generalized sense. Repeated application of this merge operation yields a tree that can then be rearranged into a tree with the same set of codeword lengths which also satisfies the linear order constraint. It is a difficult algorithm to understand in contrast with the straightforwardness of the Huffman algorithm.

Several authors have proved a version of the Kraft inequality for Hu-Tucker code trees, that is, have characterized the set of codeword lengths consistent with the existence of a lexicographically constrained tree \([140-143]\). With this result in hand, a number of results refining the \([0,2]\) Hu-Tucker code redundancy bound given partial information about the source probabilities and their linear order have been obtained, along with bounds on the Hu-Tucker average codeword length given in terms of the Huffman average codeword length for the same (unordered) probabilities \([140-143]\). Kleitman and Saks’s \([144]\) result, that the worst case ordering of source probabilities for the Hu-Tucker problem is the “sawtooth order” also serves to upper bound the Hu-Tucker average codeword length. Hu and Tan \([145]\) take this kind of approach to performance bounds in a more general ordered search tree context that includes the Hu-Tucker problem as a special case.

One other set of results characterizing the form of the Hu-Tucker code tree is due to Ramanan \([146]\) who gives inequalities on the source probabilities sufficient for a particular Hu-Tucker tree to be optimal. Ramanan’s motivation is in finding means to test whether a given tree is the Hu-Tucker tree for a fixed set of source probabilities in a way which is computationally less complex than the construction of the optimal tree itself. This may be a potentially promising perspective to take, as well, towards other constrained coding problems, to be described later in this survey, for which the optimal code tree is difficult to find algorithmically.

A stricter set of constraints on the Huffman problem than the linear order of Hu and Tucker is Van Voorhis’s \([147]\) problem in which the codewords must in addition satisfy a monotonic length constraint. As expected, the resulting minimum average codeword length code obtained from Van Voorhis’s algorithm has an average codeword length which can be upper bounded in terms of the Hu-Tucker codeword length \([148]\).
One combinatorial search problem besides the minimum average codeword length coding problem in which Huffman and Hu-Tucker code trees arise is the problem of testing for a single distinguished item on the basis of a series of tests each of which can identify whether the distinguished item is in a given subset of items presented to the test or not. We are given prior probabilities on each item as to how likely it is to be the distinguished item. If the items are components of some system arranged in a fixed left-to-right order such as in an "oil pipeline," and the tests identify whether or not the distinguished or "faulty" component is to the left of the tested point, the test tree which minimizes the average number of tests to identify the distinguished component is the Hu-Tucker tree. The problem variant in which the components are arranged according to a fully connected, partially-ordered structure with a single final element in the partial order has been examined [149] but not fully resolved. It would be of interest to have a generalization of the Hu-Tucker algorithm that applied to a partial-order constraint of this form on the source symbols. Similarly one can imagine imposing partial orders of other forms on the source symbols in a minimum average codeword length coding problem. For example, consider a set of several disjoint linear orders, representing a source alphabet of letters, numbers, and punctuation marks, and an algorithm and bounds on the resulting codeword length would be of interest in this case as well. For constrained trees in these and other search problems, the appropriate Kraft-type inequality would be of interest although in most cases it is not available.

For a parallel search of linearly ordered items, or for a parallel version of the unordered Huffman problem, an optimal binary forest is found by stopping the Hu-Tucker or Huffman merge algorithms after $K - m$ merge steps [133]. Then each of the $m$ trees in the forest can be searched simultaneously, and the average time for any of the searchers to find the single distinguished time is minimized. Entropy bounds on the resulting minimum average path lengths are given in [150] for these problems.

In the nonbinary case, Huffman forests are found by Tomescu [151]. This merge-based algorithm is more appealing than the dynamic programming approach taken by Itai [152] to nonbinary Hu-Tucker forests. Note that a nonbinary version of the Hu-Tucker algorithm remains essentially unavailable [153]. It would be interesting if such an algorithm could be obtained by combining Hu and Tucker's original perspective with Tomescu's approach.

Another potentially interesting variant of the Hu-Tucker problem may be the "parallel product" problem described in [4]. Imagine tests of the Hu-Tucker type used to identify a single distinguished item, however each item is characterized by $D > 1$ attributes, each one of which can only be tested separately. For example, the single distinguished item may be thought of as faulty, but to be faulty it must be defective in all of $D$ dependent but separately testable modes. We are given the prior probability that the item for which attribute $d$ takes on the value $i(d)$, $i(d) = 1, \ldots, K(d)$, $d = 1, \ldots, D$, is the distinguished item. Thus each node in the code or test tree distinguishes between the subset of the items for which $i(d)$ is less than some value and its complement. In essence this problem introduces probabilities
into the standard multidimensional search problem in which the attributes are tested cyclically, \( d = 1, \ldots, D \), in each level of the tree. A Huffman “parallel product” problem or one that is Huffman in some attributes and Hu-Tucker in the others might also be interesting.

In any of these Huffman problem variants with order constraints, the average codeword length obtained from the Huffman algorithm applied to the unconstrained probabilities is always a lower bound on the constrained minimum average codeword length. If the Huffman tree satisfies the constraints of the problem, or if it can be rearranged into a tree with the same codeword lengths which satisfies the constraints of the problem, then that tree solves the constrained problem. The characterization of constraint “profiles” for which the unconstrained, Huffman, algorithm applies may be useful.

2.5. Maximum Codeword Length Constraints on Huffman and Hu-Tucker Coding and Other Constrained Coding Problem Variants. Both the Huffman and Hu-Tucker problems have been addressed under the additional constraint that no codeword be longer than a certain maximum value. Clearly, as that permitted maximum value increases, eventually the unconstrained Huffman or Hu-Tucker tree results. Also, in general, the imposition of the maximum codeword length constraint will lead to larger resulting average codeword lengths than in the unconstrained Huffman and Hu-Tucker problems.

Currently the best algorithmic approach to Huffman coding under the maximum codeword length constraint is due to Moffat et al. [154-156] based on the algorithm of Larmore and Hirschberg [157]. However, a number of earlier approaches to the same problem have appeared [152, 158-164]. And see [165-166] for a Hu-Tucker coding algorithm under the maximum codeword length constraint.

Some performance analysis results are available for maximum codeword length constrained coding problems, notably Capocelli and De Santis’s [167] redundancy bounds for the length constrained Huffman problem, including the case of \( p(1) \) or \( p(K) \) given. The corresponding questions for the length constrained Hu-Tucker problem have not been addressed. The worst case ordering of source probabilities for the length constrained Hu-Tucker problem remains the “sawtooth order” as in the unconstrained problem [168].

The work of Anily and Hassin [169] is relevant to constraints such as maximum length constraints or other constrained problems which one might propose, such as maximum constraints on the difference between longest and shortest codeword lengths, or fringe, for example. They compute the best, second best, . . . , \( s \)-th best trees in terms of minimum average codeword length for fixed source probabilities from which the best tree satisfying the constraints can be identified by examining each candidate tree in turn. They deal with both the Huffman and Hu-Tucker versions of this problem. Entropy bounds on the average codeword length of the \( s \)-th best tree are not available and may be of interest to obtain.

Another interesting constrained Huffman and Hu-Tucker problem is addressed by Mandoin [170]. He supposes that some \( m \) codewords of known length are given
and that the Huffman or Hu-Tucker tree is to be constructed for \( k \) additional source symbols occurring with known probabilities. Thus the overall tree has \( m+k \) leaf nodes, but the minimum average codeword length is determined only by the \( k \) unspecified codewords.

One way of thinking about the \( m \) given codewords is as escape characters for an application in which only \( k \) source symbols from among some larger original source alphabet can be Huffman encoded due to restrictions on the number of nodes which can be maintained in the code tree. Mandoiu’s approach resolves this problem in the case that the overflow source symbols and the escape characters are predetermined independently of the coding problem. The more general problem of jointly optimizing the design of one or more escape characters and a residual Huffman code is essentially open. Smirnov and Yunakov [171] identify a special case for \( m = 1 \) in which the hybrid escape/Huffman problem can be resolved short of exhaustive search.

In recent work, Doyle and Carlson [172] have formulated the Huffman problem of minimizing average codeword length constrained by the Kraft inequality as a special case within a larger class of problems in which average codeword length is minimized subject to a resource constraint of fairly general form. They are not concerned with the integer constraint on codeword lengths, however the integer-lengths version of their resource constrained generalized coding problem appears to be of additional interest. Other forms of resource constraints may also be of interest, particularly in conjunction with the Kraft inequality in the case that a tree-structured solution is desirable.

### 2.6. Minimizing Other Codeword Length Functionals

Both the Huffman and Hu-Tucker problems have been addressed under the variant that a functional of codeword length other than average codeword length is to be minimized. Of particular interest are cases in which a merge-type algorithm like Huffman or Hu-Tucker serves to construct the optimal tree. Parker [173] has unified a large earlier literature on generalized Huffman problems into a common comprehensive framework. In particular, Parker works with a class of “quasilinear” merge functions, \( F(x, y) \), whose arguments are the probabilities weighting the nodes to be merged. Then \( F(x, y) \) will be the weight of the newly created parent node. In the ordinary Huffman algorithm, \( F_s(x, y) = x + y \). Another example of merge function which is included in Parker’s framework is \( F_m(x, y) = \max(x, y) + c \). Then Parker’s generalized merge algorithm creates a tree by merging, at each stage of the process, the two least probable source symbols, with probabilities \( x \) and \( y \), into a pair of sibling nodes whose new parent node has probability, or weight, given by \( F(x, y) \). If \( G \) is a “Schur concave” cost functional of the weights on the interior nodes of the tree, then the generalized merge algorithm forms a tree for which \( G \) is minimized. In the ordinary Huffman algorithm, \( G_s = \) sum, and the resulting tree which minimizes the sum of the weights of the interior nodes of the tree is exactly the tree which minimizes average codeword length \( \sum p(i) h(i) \). Another example of cost functional which is included in Parker’s framework is \( G_m = \max \), and the tree resulting from the generalized merge algorithm incorporating \( F_m \) which minimizes the maximum of the weights of the interior nodes of the
tree is exactly the tree which minimizes \( \max(p(i) + c(i)) \). Generalized entropy-type bounds on the resulting minimum codeword length functional are included in [173]. See also [174] for \( p(i) \) given. Among the special cases included in Parker’s framework are those addressed in [175-178] and see the references in [173]. Knuth [179] revisited Parker’s problem from an elegant abstract perspective which also incorporates Huffman problems with secondary optimality criteria such as, of all Huffman trees for given source probabilities, select the one with minimum sum of codeword lengths, a problem originated by Schwartz [180], or minimum variance, a problem originated by Kou [181]. See [182] for more on the Schwartz and Kou problems. Markowsky [183] also examines Huffman problems with secondary optimality criteria. Forst and Thorup [184] also discuss these kinds of problems. Chang and Thomas [185] further examined Knuth’s algebraic perspective on code trees, particularly in the case that random variables are substituted for the source probabilities.

The generalized merge version of the Hu-Tucker problem was addressed by Hu, Kleitman, and Tamaki [153] who work with a class of merge functions and cost functionals satisfying a set of specified properties. It is not immediately clear exactly what is the correspondence between the class of \( F, G \) pairs for which Parker’s generalized Huffman algorithm holds and the class of \( F, G \) pairs for which Hu, Kleitman, and Tamaki’s generalized Hu-Tucker algorithm holds. It may be a useful contribution to align these two apparently independent generalized merge formalisms. It may also be of interest to obtain generalized entropy-type bounds on the minimum codeword length functional for alphabetic codes in the context of the Hu, Kleitman and Tamaki algorithm. Zhang [186] proves that a generalized merge Hu-Tucker type algorithm finds the optimal tree for a particular \( F, G \) pair, \( F_m \) for \( c = 1 \) and \( G_s \), which is claimed to be outside the original class of \( F \) and \( G \) treated by Hu, Kleitman, and Tamaki. Kirkpatrick and Kløve [187] carry out the details of the \( F_m, G_m \) problem for \( c = 1 \) for alphabetic codes in the nonbinary case thus addressing in part an issue raised in [153]. They also provide a generalized entropy-type upper bound for this \( F, G \) pair.

Occasionally the Huffman problem is considered under some other optimization criterion variant besides those in Parker and Knuth for which a generalized merge algorithm finds the optimal code tree. One such problem is to find Cover’s [188] competitively optimal code tree. The Huffman tree finds \( L \) such that \( \sum p(i)(l(i) - l'(i)) < 0 \) for all \( L' \). Cover asks for the tree such that \( \sum p(i) \cdot \text{sgn}(l(i) - l'(i)) < 0 \). Feder [189] and Yamamoto and Itoh [190] also examine competitive optimality, and Yamamoto and Itoh show that if the competitively optimal code exists for given source probabilities then it is also the Huffman code tree, and they provide conditions on \( P \) for its existence. Perhaps it may be of interest to consider the tree such that \( \sum p(i)f(l(i) - l'(i)) < 0 \) for some general class of nonlinear functions \( f \).

Larmore [191] is interested in minimizing a particular nonlinear function of average codeword length and codeword variance that arises in modeling delay in communicating coded strings. He resolves the Huffman version of his problem but leaves the Hu-Tucker version open. One can imagine a broader class of nonlinear functions
of average codeword length and other codeword length functionals being of interest as performance criteria as well. It would be of interest to determine how broadly Larmore’s method applies. Since the approach consists of two stages, one which minimizes all linear combinations of average codeword length and codeword variance, and one which searches over all candidate trees within a constrained set determined by the result of the first stage together with the form of the particular nonlinear function in the problem formulation, it seems reasonable to expect that the general approach is more widely applicable. Nevertheless the algorithm itself is not as elegant as the merge-based Huffman algorithm.

Cohen and Fredman [192] address the minimization of a performance criterion that arises when \( m \) independent processors use a single test tree in parallel. Each is to identify its own distinguished item from among the \( K \) items. In particular, \( m \) items are selected independently according to \( P \). Assuming each processor’s search must be completed before the next phase of the computation begins, the cost to be minimized is the average length of the maximum of the \( m \) path lengths through the tree. For \( m = 1 \), this is the ordinary Huffman problem or the ordinary Hu-Tucker problem if an alphabetic constraint is imposed. The approach of their suboptimal algorithm is to derive a set of transformed probabilities to which the \( m = 1 \) Huffman algorithm is applied. Clearly it would be of interest to resolve this intriguing problem optimally. Similarly one can introduce other search problems based on Cohen and Fredman’s scenario of \( m \) independent processors, and these may also be of interest.

In the binary unequal costs coding problem, using the code symbol 0 costs \( c(0) \geq 1 \) and using the code symbol 1 costs \( c(1) \geq 1 \). Thus a path through the code tree which consists of \( M(i) \) left branches and \( N(i) \) right branches, or, that is, a codeword consisting of \( M(i) \) 0’s and \( N(i) \) 1’s, costs \( l(i) = c(0)M(i) + c(1)N(i) \). When \( c(0) = c(1) = 1 \), \( l(i) \) is codeword length in the usual sense. Karp [193] first gave an algorithm to find the optimal code tree in the sense of minimum average codeword cost, that is to solve \( \min_L \sum_{i \leq i < K} p(i)l(i) \), and this problem will be referred to as the Karp problem. Actually, Karp addressed the case of \( r \)-ary code alphabets and distinguished between exhaustive codes, in which each interior node of the tree has exactly \( r \) descendants, and nonexhaustive codes, in which each interior node has at least 2 and no more than \( r \) descendants. Generally in unequal costs coding problems, the algorithms to find optimal exhaustive codes are simpler but the nonexhaustive codes yield lower average cost. For binary code alphabets the distinction does not arise. For \( c(0) = c(1) = 1 \), Karp’s problem is Huffman coding.

Karp’s method is an integer programming method that makes use of a useful structure function method for representing code trees algebraically. Golin and Rote [194] provide a dynamic programming algorithm to solve the Karp problem that is much more computationally efficient than Karp’s approach. Both Karp and Golin and Rote address the unordered or Huffman version of the unequal costs coding problem. See Itai [152] for the ordered or Hu-Tucker version. Because no bottom up algorithms are known for optimal unequal costs coding problems, optimal code trees are not
obtained from optimal forests. The problem of obtaining minimum average codeword cost forests may be of interest.

Entropy bounds for the Karp problem are given by Krause [195]. See also [196]. The Kraft inequality for unequal costs on which the bounds are based is found in [193]. See also [197]. Recently De Prisco and Persiano [198] extended the Kraft-inequality-like results of [140] for the Hu-Tucker problem to the ordered Karp problem. Hwang’s [96] majorization inequalities for comparing minimum average codeword lengths across source probability distribution carry over straightforwardly to unequal costs coding problems.

A number of authors have dealt with techniques for finding good but not optimal unequal costs codes for arbitrary source probabilities [199-201] or have developed techniques to find optimal codes for uniform source probabilities and special cost structures [202-205].

Finding optimal codes for uniform source probabilities and arbitrary cost structures is a problem of interest for which a substantial body of results is known. Varn [206] first provided algorithms to find the optimal exhaustive and nonexhaustive code trees in the minimum average codeword cost sense. Perl, Garey, and Even [207] introduced a nonexhaustive algorithm which is computationally better than Varn’s and provided a simpler algorithm to find the optimal nonexhaustive code tree in the minimax codeword cost sense. In the exhaustive case the same code trees are optimal according to both minimum average and minimax cost performance criteria [208]. Other nonexhaustive algorithms are in [209-210]. Choi and Golin’s [211] algorithm for the nonexhaustive Varn problem is the best computationally. Savari [212] provides performance bounds on the resulting average codeword cost of Varn codes. See also [213] for the binary case.

Varn’s algorithm for exhaustive code trees is a top down splitting algorithm in which a least cost node is replaced by two descendant nodes at each stage. The costs of the descendants are the cost of the parent plus the cost of the associated branch. The sequence of Varn code trees has an underlying Fibonacci tree structure first identified by Horibe [214] in the case that $c(0) = 1$ and $c(1) = 2$ and since generalized for arbitrary integer costs [215-216]. A similar generalized Fibonacci structure is present in the sequence of nonexhaustive minimax Varn trees [217]. Hinderer [218] also addresses the binary Varn problem independently.

Horibe has written a number of papers on the properties, particularly the balance properties, of the Fibonacci trees arising in the solution to the Varn problem [219-222]. One property of particular interest is the fact that the generalized Fibonacci tree that solves the Varn problem for arbitrary costs both minimizes average codeword cost and maximizes the entropy of the probability distribution induced by the tree [223]. The induced distribution is of the form $P = \{ p(i) = t(i), i = 1, \ldots, K \}$ where $t(0) + t(1) = 1$. The idea of induced distribution will reappear in the following in the discussion of parse trees.

Besides uniform sources, Karp’s problem is also easily solved for distributions whose relationship to the code symbol costs generalizes the relationship between
dyadic sources and binary equal code symbol costs [224]. These distributions can be coded with zero redundancy.

Consider now the generalized unequal costs coding problem in which the costs of the code symbols vary with the position of the code symbol in the codeword. This model of code symbol costs is Shannon’s original discrete noiseless channel model in which a finite state diagram describes the cost of each code symbol when it appears following some particular string of code symbols in the codeword [225]. Finite state channel models arise frequently in the literature on magnetic and optical recording [226], however this literature is not typically concerned with either variable-length-to-block codes or with minimum average cost as a performance criterion. Csiszár [227] proposed a nonoptimal approach to variable length coding for finite state code symbol costs and provided entropy-type performance bounds. The particular case of costs that grow rapidly with the position of the code symbol in the codeword was addressed in [228-229], and bounds of Csiszár’s form are shown not to hold. It would be of interest to derive algorithms for minimum average codeword cost codes for finite state cost models, for both arbitrary source probabilities and for the special case of uniform source probabilities. The author [230] was able to show that a Varn-like algorithm applies to certain finite state cost models for uniform source probabilities. In essence \( c(0) + c(1) \) must be nearly constant at every interior node of the code tree for the Varn-like algorithm to apply. Recall that for Varn codes, \( c(0) \) and \( c(1) \) are fixed at every interior node of the code tree.

A somewhat different unequal costs coding problem is to minimize average codeword length subject to a constraint on maximum codeword cost. This problem arises in [231] where it is resolved for the special case \( c(0) = 0 \) and \( c(1) = 1 \). It would be of interest to address this problem for \( c(0) = 1 \) and arbitrary \( c(1) > 1 \).

2.8. Coding for Bidirectionality and Synchronization. In some coding applications it is of interest to select the assignment of patterns of 0’s and 1’s within the codewords in order to obtain good performance with respect to some secondary pattern-based criterion, most typically bidirectionality or synchronization. Ideally the minimum average codeword length property of the underlying code tree is not affected by the choice among codewords with the specified codeword lengths, but sometimes average codeword length is sacrificed in order to gain improved performance with respect to the secondary criterion. Work in the theoretical computer science community in the spirit of Berstel and Perrin [14] emphasizes the characterization of codeword lengths compatible with the required patterns but independently of a source probability model. See [232] and [233] for pointers into this literature for bidirectionality and synchronization respectively.

Bidirectionality is one such secondary criterion. Several authors have considered this problem [234-238]. If the set of codewords also has the property that when read backwards the codewords form a code tree, then the code is bidirectional. Questions of interest include characterizing the sets of codeword lengths that are consistent with bidirectionality, finding codeword string assignments with the bidirectionality property when they exist, bounding the increase in average codeword length required
to ensure bidirectionality when the Huffman codeword lengths are inconsistent with bidirectionality.

Due to the difficulties associated with resolving basic questions about bidirectional codes, Ye and Yeung [239] have developed an interesting and powerful new method that they apply to bidirectionality but which may also be useful in other coding problems. In particular, they describe a random procedure for constructing a code with a given set of codeword lengths and then check to see if the codewords form a bidirectional code, evaluating a lower bound on the probability that a bidirectional code has been found. As long as that bound is strictly greater than zero, there exists at least one bidirectional code with the given codeword lengths. This method yields Kraft-inequality-like results for bidirectional codes and, in fact, provides a novel rederivations of the classical Kraft inequality for unrestricted binary code trees. In addition, Ye and Yeung use their Kraft-like results to obtain bounds on the redundancy for bidirectional codes given partial information about the source probabilities.

Synchronization is another such secondary criterion. A number of authors [240-249] have examined conditions on the set of codeword lengths sufficient for the existence of one or more synchronizing codewords, a codeword whose pattern of code symbols, relative to the other codewords in the code, is such that the next received symbol is always correctly interpreted as the beginning of a new codeword, whether or not the synchronizing codeword was itself correctly received and/or decoded. Other authors [250-251] require the synchronizing codeword to have the stronger property that it is itself always correctly decoded when correctly received. In each case there is interest in constructing the codeword set and in evaluating both its synchronizing performance and its loss of average codeword length relative to the Huffman code if that loss occurs.

An alternative perspective on synchronization is to design a set of variable length codes for a given source alphabet size with good overall synchronization properties and then to select the best code from within that set, best in the sense of minimum average codeword length for the source distribution. Neumann [252-253] took this approach early on, incorporating distinctive synchronizing patterns in each codeword. More recently, a diffused concept of synchronizing patterns appears in the "T-codes" approach taken by Titchener [254-256] and described most tutorially by Günther [257], and further examined by Higgin [258-260] and by Swashek and Willett [261]. Unfortunately a good analytical evaluation of the tradeoff between synchronization performance and average codeword length for T-codes is not available, nor is there an algorithm other than exhaustive search for the identification of the best T-code in some combined synchronization/average-codeword-length sense.

Several authors have addressed the issue of synchronization recovery in variable-length codes beginning with Maxted and Robinson [262]. See Swashek and DiCicco [263] and the references therein, as well as, more recently [264].

Another application in which the patterns of code symbols within the codewords matter is in using Huffman codes cryptographically [265]. One of the issues here is to identify conditions on the Huffman codeword lengths such that different strings of
source symbols lead to an identical string of code symbols when the assignment of code symbols within the codewords differs. This kind of ambiguity is of cryptographic significance.

Immutable codes, as discussed by Capocelli, Gargano, and Vaccaro [266], are another interesting area in which patterns of code symbols are central. In this problem, allowable rewrites of code symbols are not permitted to transform a message of source symbols into another legitimate message, thus preserving data integrity. The problem of designing optimal variable length immutable codes remains open.

3. Parse Trees. In variable-length-to-fixed binary source coding, a parse tree is used to associate a string of binary source symbols with each of a set of $K$ code symbols. A parse tree is a complete binary tree with $K$ leaf nodes and $K - 1$ internal nodes including the root at the top of the tree. Pairs of branches descend from each internal node. Left branches are labeled with the source symbol 0 and right branches are labeled with the source symbol 1. Each leaf node is labeled with one of the $K$ code symbols, $s_1, \ldots, s_K$. See Figure 1 for an example. A path through the parse tree from the root to a leaf describes the string of source symbols associated with the code symbol at the leaf. The problem is to find a parse tree to optimize some performance criterion, possibly under the imposition of constraints on the form of the tree, when the source symbol is 0 with probability $r_0$ and 1 with probability $r_1$. Thus a path through the parse tree which consists of $M(i)$ left branches and $N(i)$ right branches, or, that is, a parse string consisting of $M(i)$ 0’s and $N(i)$ 1’s, corresponds to a string of source symbols which occurs with probability $R = \{r(i) = r_0^{M(i)} r_1^{N(i)}, i = 1, \ldots, K\}$. Refer to such an $R$ as the probability distribution induced by the parse tree.

3.1. Tunstall Parse Trees and Variants. Tunstall parsing [7, 267], also derived independently by Verhoeff [268], solves the problem of finding the parse tree to maximize average parse string length, or, that is, to solve $\max_{M,N} \sum_{1 \leq i \leq K} r(i)(M(i) + N(i))$ for $M = \{M(i), i = 1, \ldots, K\}, N = \{N(i), i = 1, \ldots, K\}$. Here $l(i) = M(i) + N(i)$ is the length of the $i$th parse string, and it arises with probability $r(i)$. There are no further constraints on the problem however $\sum 2^{-l(i)} \leq 1$ must hold in order that the binary parse tree exists, from the Kraft inequality. Another way of thinking about Tunstall parsing is that it finds the parse tree whose induced distribution $R$ has maximum entropy, or, that is, it solves $\max_{M,N} -\sum r(i) \log_2 r(i)$. This follows from the fact that [267] $H_t(R) = E[L] H_D(r)$, where $H_t(R) = -\sum r(i) \log_2 r(i)$ is the entropy of the distribution induced by a parse tree, $H_D(r) = -r_0 \log_2 r_0 - r_1 \log_2 r_1$ is the entropy of the source distribution, and $E[L]$ is the expected parse string length of that parse tree. A minimum discrimination perspective on Tunstall parsing is due to Stubley and Blake [269] who examined a broader class of parsing problems. In their framework, the distribution induced by the parse tree, $R$, is the closest induced distribution to the uniform distribution $S = \{s(i) = 1/K, i = 1, \ldots, K\}$ in the minimum discrimination sense. That is to say, Tunstall parsing solves $\min_{M,N} \sum r(i) \log_2 r(i)/s(i)$ for $S$ uniform. Since $S$ is a constant with respect to the minimization, this is exactly maximum entropy parsing. More generally, Stubley and Blake consider the minimum
discrimination parsing problem in which $S$ is a given distribution, not necessarily uniform, to be imposed on the leaves of the parse tree, and the goal is for the probability distribution induced by the parse tree to be as close as possible to a desired code symbol distribution $S$. They do not have an algorithm to find the optimal parse tree for the problem in its full generality.

Tunstall’s algorithm is a top down splitting algorithm in which a node of greatest probability is replaced by two descendant nodes at each stage. The weights of the descendants are the weight of the parent multiplied by the probability of the associated branch. Tunstall’s algorithm is exactly isomorphic to Varn’s algorithm for exhaustive unequal costs code trees for the uniform source distribution under the mapping $r_0 = t^{[0]}$, $r_1 = t^{[1]}$ as pointed out by Savari [212]. Both algorithms maximize the entropy of the induced distribution.

Petry and Schalkwijk [270-271] identify a generalized Fibonacci structure in binary Tunstall parse trees and exploit it in the implementation process. Actually their parsing problem was formulated independently of Tunstall but can be identified as equivalent.

A recent paper by Fabris, Sgarro and Paletti [272] addresses some characterization results for Tunstall codes, such as the relationship between source probabilities and the resulting minimum and maximum parse string lengths, motivated by applications to adaptive Tunstall parsing. They also use a discrimination measure of mismatch that arises in Huffman mismatch problems.

Nonexhaustive parse trees are not usually of interest, because every possible string of source symbols must be encoded, however Algra [273] describes a particular nonoptimal nonexhaustive parse tree variant of Tunstall parsing which incorporates special escape strings pointing to codewords for selected unlikely source strings. Strictly speaking Algra’s algorithm is not a variable-length-to-fixed algorithm but rather a special type of variable-to-variable-length coding algorithm in which the variable length parse strings are represented by either single code symbols or pairs of code symbols in a particularly highly structured way well suited to implementation. However, in general, the compression performance can be improved by using dual tree codes of less restricted form as described in a later section.

Nonproper parse trees in which some interior nodes correspond to code symbols are investigated by Yamamoto and Yokoo [274]. They design approaches that improve on the performance of the Tunstall algorithm for nonbinary source alphabets. Earlier work by Gilbert and Monma [275] poses related problems.

Lempel, Even, and Cohn [276] examine a parsing problem for the special case that the binary source symbols are equally probable and thus the induced distribution $R$ is dyadic. One can imagine a generalization of Lempel, Even, and Cohn’s problem in which $r_0$ and $r_1$ are not necessarily equal to $1/2$, and Tunstall parsing would be a special case of this generalization. The particular parsing problem of Lempel et al. is to solve $\min_{M,N} \sum r(i) \log_2 s(i) / \sum r(i) \log_2 r(i)$ where $R$ is the dyadic distribution induced by the parse tree and $S$ is a given distribution imposed on the leaves of the tree. This is in a sense a minimum distance parsing problem for a notion of distance
different from discrimination. Lempel et al. provide an algorithm to find the optimal parse tree for their problem which is an iterative algorithm which converges to the optimal tree, and, in each stage of the algorithm, a Huffman-type merge is performed. In their original problem statement, Lempel et al. associate costs with the individual code symbols and minimize the ratio of average code symbol cost to parse string length.

Jelinek and Schneider [277] address a parse problem in which the performance criterion is to minimize the average value of a constant \( c \) raised to the parse string length. But, because the average is taken with respect to the distribution induced by the parse tree in which \( r(i) = r_0^{M(i)}r_1^{N(i)} \), it is immediate that the Tunstall splitting algorithm applies to this problem as well, just by replacing the source probabilities \( r_0 \) and \( r_1 \) with values \( cr_0 \) and \( cr_1 \) respectively.

In the case that Tunstall’s splitting algorithm is applied to “probabilities” \( r_0 \) and \( r_1 \) where \( r_0 + r_1 < 1 \), the resulting tree also minimizes \( \sum r(i) \), a quantity identically equal to 1 for all parse trees when \( r_0 + r_1 = 1 \) [278].

Parse problems have been much less extensively studied than code problems. It may be that interesting parse problem variants are amenable to analysis.


The isomorphism between binary Tunstall parse trees and binary Varn code trees is a special case of a more general duality between parse trees and unequal costs code trees [279]. In particular, the general parse problem of Stubley and Blake can be formulated as finding the tree to solve \( \min R D(R, S) = \min_R \sum r(i) \log_2 r(i)/s(i) \) where \( R \) is the distribution induced by the tree and \( S \) is a given distribution imposed on the leaves of the tree. With this perspective on parse problems in mind, we can reformulate the minimum average codeword cost problem of Karp as finding the tree to solve \( \min_Q D(P, Q) \) where \( Q \) is the distribution induced by the code tree under the mapping \( q(0) = t^{(0)}, q(1) = t^{(1)} \) and \( P \) is a given distribution imposed on the leaves of the tree, the source distribution. Specifically \( Q = \{ q_i = (t^{(0)})^{M(i)}(t^{(1)})^{N(i)} = t^{(i)}, i = 1, ..., K \} \). Note that \( \min_Q D(P, Q) \) is equivalent to \( \min_{M,N} \sum p(i)(M(i) + N(i)) \) because the entropy of \( P \) is a constant with respect to the minimization. The same algorithm does not solve both the Stubley and Blake problem and the Karp problem due to the asymmetry of \( D \) with respect to its arguments.

A similar duality can be established between Karp coding and generalized Lempel, Even, Cohn parsing [276] in which Huffman coding and the original, dyadic, parsing problem of Lempel et al. are dual to each other.

3.3. Dual Tree Coding.

In variable-to-variable-length binary coding, or dual tree coding, a parse tree with \( K \) leaf nodes is matched to a code tree with \( K \) leaf nodes to optimize some performance criterion. Ideally it would be of interest to minimize the ratio of average codeword length, or more generally average codeword cost, to average parse string length, however, that problem has not been tractable, and a number of authors have examined the alternative performance criterion of \( \min_{R, Q} D(R, Q) \) where \( R \) is the distribution induced by the parse tree and \( Q \) is the
distribution induced by the code tree [280–282]. In both of these equal costs dual tree coding problems the code tree will be the Huffman tree for the distribution induced by the parse tree so the problem reduces to finding the optimal parse tree. In general, minimum discrimination dual tree coding is not equivalent to minimizing the ratio of average codeword length to average parse string length, but for dyadic $R$ they are the same dual tree codes. Unfortunately minimum discrimination dual tree coding is also intractable for arbitrary source distributions, although good approximate solutions are known.

### 3.4. Parse Trees and Random Number Generation

Given a parse tree and an associated source distribution, a distribution is induced at the leaves of the tree. Label each leaf with one of $N \leq K$ variables. Associate the sum of the induced probabilities at the leaves that have a common label with that variable. Thus a random variable whose distribution is the source distribution can be used to generate random variables according to the labeled leaf distribution by parsing a string of source symbols according to the labeled parse tree. A problem originated by Knuth and Yao [283] and discussed in [27] is to start with the distribution of the random variable to be generated and the source distribution and construct the parse tree which minimizes the expected number of source symbols used to generate the random variable. An optimal parse tree for an equiprobable binary source can be found from the coefficients of the expansions base $1/2$ of the desired probabilities. The generalization to arbitrary discrete sources appears in [284]. Of course there is also a substantial literature on random number generation based on a variety of techniques other than parse trees.

Code trees can also be used to convert a given distribution associated with the leaves of the tree into a binary distribution induced by the codewords [84]. The induced distribution depends on the assignment of 0’s and 1’s within the codewords. Thus the concatenation of parse and code trees as in dual tree coding, or the concatenation of code and parse trees, should serve as distribution converters. Morita, Kobayashi and Hoshi [285] have begun investigating these kinds of questions, but constructive approaches for the efficient generation of a specified target distribution remain open.

### 4. Representing and Counting Code and Parse Trees

It may be desirable to represent the code or parse tree itself as a string of symbols in an unambiguously recoverable fashion. Of course the efficiency of such a representation is of interest, both in the sense of the compactness of the representation as well as in the sense of the computational effort required to recover the tree from the representation. One can always represent the tree by the probabilities from which it was generated, assuming that the algorithm from which the tree was generated, together with tie-breaking conventions, is known. Even if the tree is not a Huffman tree, it can be represented in this way by the dyadic probabilities corresponding to the tree path lengths. However, because this tends to be a computationally intensive approach, one of the other, general, tree representation techniques may be useful for code and parse tree representation. These techniques do not exploit any distributional information on the set.
of all potential code trees, nor is such distributional information typically available.

Surveys of tree representation techniques are given by Katajainen and Mäkinen [286-287]. They categorize the various techniques as permutation-based, rotation-based, and enumeration-based. Permutation-based methods involve a traversal of the tree identifying interior and leaf nodes distinctively. Rotation-based methods describe a series of transformations to take a canonical tree into the given tree. Enumeration-based methods describe a technique for generating all trees in a fixed order, and denote the given tree by its index in the list of trees so generated. Labels can be separated from tree shapes and encoded separately.

The structure function idea described by Karp [193] resembles the permutation-based approaches outlined in [286-287] and can be used as a tree representation method.

The canonical code tree of Schwartz and Kallick [288] is essentially a rotation-based method that has appeared in various forms, sometimes independently, in the code tree literature [289-293]. Canonical code trees have been central to efficient implementations of Huffman coding and decoding. See the work of Moffat and Turpin [294-295] for a discussion of implementation issues and an overview of that literature.

In an interesting paper, Parker and Ram [296] construct a lattice of binary trees based on a partial order that describes relative balance. Edges in the lattice correspond to the exchange of a pair of sibling leaves at one point in the tree for a pair elsewhere, a rotation-type operation. The Huffman algorithm can be recovered as the solution to an optimization problem in this lattice-theoretic framework. It may also be that lattice concepts will provide new insights into other optimal coding problems.

A series of papers on counting the number of distinct code trees [64, 159, 297-306] yields expressions useful in the evaluation of the efficiency of tree representation techniques relative to enumeration based techniques. This is a subset of a much larger literature on counting the number of discrete structures of various types.

Bobrow and Hakimi [307] and Fittingof and Waksman [308] have described methods for representing code and parse trees as graphs by identifying common subtrees with each other. Graph representations may not be as useful as string representations in transmitting or storing code trees. Trees with common subtree structures may lead to string representations which can themselves be further compressed.

One can also consider the possibility of representing a tree, as a special case of a graph or digraph, by means of its adjacency matrix, and then treating that matrix of 0's and 1's as a bit map for compression purposes, e.g. [21, Sect. 3.5].

5. Concluding Remarks. There are many interesting variants of coding and parsing problems for which further investigation may be worthwhile, both from an algorithmic perspective as well as from the perspective of performance evaluation and tree characterization. It is hoped that this survey is a useful guide to the existing literature. Note added in press. Additional recent references in the area of this survey are [309-312].
REFERENCES

pp. 146-150.


[94] T. Nemeth, Information type measures and the applications to finite decision problems, Carleton Mathematical Lecture Notes, No. 17, Carleton University, Ottawa, May 1977.


[227] I. Csiszar, Simple proofs of some theorems on noiseless channels, Inform. and Control,


251. S.-M. Lee, The construction of efficient variable-length codes with clear synchronizing codewords for digital video applications, Proc. Conf. on Visual Communications and Image
CODE AND PARSE TREES FOR LOSSLESS SOURCE ENCODING


[275] E. N. Gelbert and C. L. Monma, Multigram codes, IEEE Trans. on Inform. Th., IT-


