[7] S. Verdú and T. S. Han, "A general formula for channel capacity," *IEEE Trans. Inform. Theory*, vol. 40, pp. 1147–1158, July 1994.
[8] F. M. J. Willems, "The context-tree weighting method: Extensions," in *IEEE Int. Symp. Information Theory*, Trondheim, Norway, 1994.
[9] ——, "Coding for a binary independent piecewise-identically-distributed-source," *IEEE Trans. Inform. Theory*, vol. 42, pp. 2210–2217, Nov. 1996.
[10] E. Yang and J. C. Kieffer, "Simple universal lossy data compression schemes derived from the Lempel–Ziv algorithm," *IEEE Trans. Inform. Theory*, vol. 42, pp. 239–245, Jan. 1996.
[11] J. Ziv and A. Lempel, "Compression of individual sequence via variable rate coding," *IEEE Trans. Inform. Theory*, vol. IT-24, pp. 530–536, Sept. 1978.

A Dynamic Programming Algorithm for Constructing Optimal "1"-Ended Binary Prefix-Free Codes

Sze-Lok Chan and Mordecai J. Golin, *Member, IEEE*

***Abstract*—The generic *Huffman-Encoding Problem* of finding a minimum cost prefix-free code is almost completely understood. There still exist many variants of this problem which are not as well understood, though. One such variant, requiring that each of the codewords ends with a "1," has recently been introduced in the literature with the best algorithms known for finding such codes running in exponential time. In this correspondence we develop a simple $O(n^3)$ time algorithm for solving the problem.**

***Index Terms*—Dynamic programming, one-ended codes, prefix-free codes.**

I. Introduction

In this correspondence we discuss the problem of efficiently constructing minimum-cost binary prefix-free codes having the property that each codeword ends with a "1."

We start with a quick review of basic definitions. A *code* is a set of binary words $C = \{w_1, w_2, \cdots, w_n\} \subset \{0,1\}^*$. A word $w = \sigma_{i_1}\sigma_{i_2}\cdots\sigma_{i_l}$ is a *prefix* of another word $w' = \sigma'_{i_1}\sigma'_{i_2}\cdots\sigma'_{i_{l'}}$ if w is the start of w'. Formally, this occurs if $l \leq l'$ and, for all $j \leq l$, $\sigma_{i_j} = \sigma'_{i_j}$. For example, 00 is a prefix of 00011. Finally, a code is said to be *prefix-free* if for all pairs $w, w' \in C$, w is not a prefix of w'.

Let $P = \{p_1, p_2, p_3, \cdots, p_n\}$ be a discrete probability distribution, that is, $\forall i$, $0 \leq p_i \leq 1$ and $\sum_i p_i = 1$. The cost of code C with distribution P is

$$\mathrm{Cost}\,(C,P) = \sum_i |w_i| \cdot p_i$$

where $|w|$ is the length of word w; $\mathrm{Cost}\,(C,P)$ is, therefore, the average length of a word under probability distribution P. The *prefix-coding problem* is, given P, to find a prefix-free code C that minimizes $\mathrm{Cost}\,(C,P)$. It is well known that such a code can be found in $O(n \log n)$ time using the greedy *Huffman-Encoding* algorithm, see, e.g., [5] or even $O(n)$ time if the p_i are already sorted [6].

In 1990, Berger and Yeung [1] introduced a new variant of this problem. They defined a *feasible* or 1*-ended* code to be a prefix-free code in which every word is restricted to end with a "1." Such codes are used, for example, in the design of self-synchronizing codes [3] and testing. Given P, the problem is to find the minimum-cost 1-ended code. Fig. 1 gives some examples.

$\mathbf{C_1}$:	10	11	000	001	010	011
$\mathbf{C_2}$:	01	11	001	101	0001	1001
$\mathbf{C_3}$:	1	01	001	0001	00001	000001

Fig. 1. Code C_1 is an optimal prefix-free code for the distribution $(1/6), (1/6), (1/6), (1/6), (1/6), (1/6)$. C_2 is an optimal *One-ended* prefix-free code for the same distribution. C_3 is an optimal one-ended code for the distribution $0.9,\ 0.09,\ 0.009,\ 0.0009,\ 0.00009,\ 0.00001$.

In their paper, Berger and Yeung derived properties of such codes, such as the relationship of a min-cost feasible code to the entropy of P, and then described an algorithm to construct them. Their algorithm works by examining all codes of a particular type, returning the minimum one. They noted that experimental evidence seemed to indicate that their algorithm runs in time exponential in n. A few years later, Capocelli, De Santis, and Persiano [4] noted that the min-cost code can be shown to belong to a *proper* subset of the code-set examined by Berger and Yeung. They, therefore, proposed a more efficient algorithm that examines only the codes in their subset. Unfortunately, even their restricted subset contains an exponential number of codes[1] so their algorithm also runs in exponential time.

In this correspondence we describe another approach to solving the problem. Instead of enumerating all of the codes of a particular type it uses dynamic programming to find an optimum one in $O(n^3)$ time.

II. Trees and Codes

There is a very well-known standard correspondence between prefix-free codes and binary[2] trees. In this section we quickly discuss its restriction to the 1-ended code problem. This will permit us to reformulate the min-cost feasible code problem as one that finds a min-cost tree. In this new formulation we will require that $p_1 \geq p_2 \geq \cdots \geq p_n \geq 0$ but will no longer require that $\sum_i p_i = 1$.

Definition 1: Let T be a binary tree. A leaf $u \in T$ is a *left leaf* if it is a left child of its parent; it is a *right leaf* if it is a right child of its parent.

The *depth* of a node $v \in T$, denoted by $\mathrm{depth}\,(v)$, is the number of edges on the path connecting the root to v.

We build the correspondence between trees and codes as follows. First let T be a tree. Label every left edge in T with a 0 and every right edge with a 1. Associate with a leaf v the word $w(v)$ read off by following the path from the root of T down to v. Now let $v_1, v_2, \cdots, v_n$ be the set of right leaves of T. Then $C(T) = \{w(v_1), w(v_2), \cdots, w(v_n)\}$ is the code associated with T. Note that this code is feasible since all of its words end with a 1. Note also that there can be many trees corresponding to the same feasible code. See Fig. 2 for an example.

Manuscript received March 8, 1998; revised October 6, 1999. This work was supported in part by Hong Kong RGC/CRG under Grants HKUST652/95E, 6082/97E , and 6137/98E.

The authors are with the Department of Computer Science, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong (e-mail: SZELOK@cs.ust.hk; GOLIN@cs.ust.hk).

Communicated by D. Stinson, Associate Editor for Complexity and Cryptography.

Publisher Item Identifier S 0018-9448(00)04283-8.

[1]The proof of this fact is a straightforward argument that recursively builds an exponentially sized set of codes that belong to the restricted subset. Because of space considerations we do not include it here but the interested reader can find the details in [2].

[2]In this correspondence we use the slightly nonstandard convention that a binary tree is a tree in which every internal node has *one or two* children.

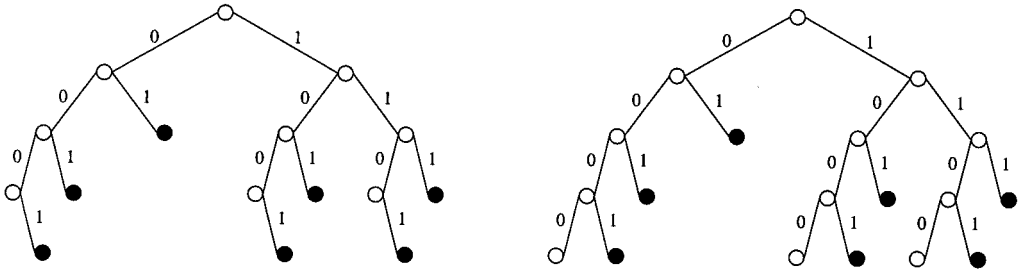

Fig. 2. Two trees with depth 4 having seven right leaves. Note that these two trees both correspond to the code $\{0001, 001, 01, 1001, 101, 1101, 111\}$. The left tree is nonfull while the right one is full.

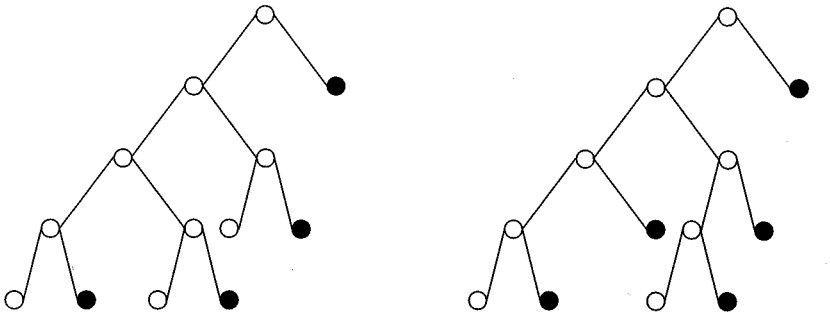

Fig. 3. The left tree is nonfeasible because, at depth 3 it contains both an internal right node *and* a left leaf. The right tree is feasible.

Now let $C = \{w_1, w_2, \cdots, w_n\}$ be any feasible code. Let $T(C)$ be the smallest tree that contains all of the paths corresponding to the w_i. Since C is prefix-free we have that the right leaves of $T(C)$ are exactly the nodes corresponding to the words of C.

Let T be a tree with n right leaves labeled $v_1, v_2, \cdots, v_n$, $P = \{p_1, p_2, p_3, \cdots, p_n\}$ and define

$$\mathrm{Cost}\,(C,T) = \sum_i \mathrm{depth}\,(v_i) \cdot p_i.$$

This is the *weighted-external path length* of T restricted to right leaves (external nodes). In all that follows $P = \{p_1, p_2, p_3, \cdots, p_n\}$ will be considered fixed and the dependence of quantities such as $\mathrm{Cost}\,(C,T)$ on P will be implicitly assumed.

Now suppose that T corresponds to some code C and $v \in T$ is a right leaf corresponding to $w \in C$; by definition $\mathrm{depth}\,(v) = |w|$. Thus

$$\begin{aligned}\mathrm{Cost}\,(C,T) &= \sum_i \mathrm{depth}\,(v_i) \cdot p_i \\ &= \sum_i |w_i| \cdot p_i = \mathrm{Cost}\,(C,P).\end{aligned}$$

Since every feasible code corresponds to some tree(s) and every tree corresponds to one feasible code this last equation tells us that we can find a min-cost code by constructing a min-code tree and returning the feasible code corresponding to it.

There is a technical problem that we need to address before proceeding. It is that our definition of cost formally requires that the right leaves of T be labeled $1, 2, \cdots, n$. Different labelings of the right leaves could lead to different costs. We note though that, for a particular tree, the minimum cost over all labelings is *always* achieved when the highest node in the tree is assigned the largest weight p_1, the second highest node the second highest weight p_2, and in general the ith highest node (with height ties broken arbitrarily) the ith weight p_i. Since we are interested in finding a minimum cost tree we will always assume that the labeling used for any particular tree is the canonical labeling with v_i being the ith highest node. For example, if the weights are $7, 6, 5, 4, 3, 2, 1$, then the trees in Fig. 2 have cost $2 \cdot 7 + 3 \cdot (6 + 5 + 4) + 4 \cdot (3 + 2 + 1) = 83$.

The Optimal Feasible Coding Problem is now seen to be equivalent to the following tree problem.

Definition 2: The *Optimal Tree Problem* Given $p_1 \geq p_2 \geq \cdots \geq p_n$ find a tree $\bar{T}$ with n right leaves with minimum cost over all trees with n right leaves, i.e.,

$$\mathrm{cost}\,(\bar{T}) = \min\{\mathrm{cost}\,(T) : T \text{ has } n \text{ right leaves}\}.$$

We end this section by pointing out that there must be an optimal tree with a very specific structure.

Definition 3: A tree T is *full* if every internal node in T has two children.

A tree T is *feasible* if T is full and it also has the additional property: if $u \in T$ is a right node and internal then *all* left nodes $v \in T$ with $\mathrm{depth}\,(v) = \mathrm{depth}\,(u)$ are also internal.

Fig. 2 illustrates a nonfull tree and a full one. Fig. 3 illustrates a nonfeasible tree and a feasible one.

Lemma 1: For every probability distribution

$$P = \{p_1, p_2, p_3, \cdots, p_n\}$$

there exists an optimal tree T that is feasible.

The proof of the lemma is straightforward but technical. To avoid breaking the flow of the correspondence it has, therefore, been relegated to the Appendix.

III. Truncated Trees and Signatures

Our approach will be a modification of one developed in [7]. The problem considered there was to build a min-cost *lopsided tree* (tree in which edges have different length). The solution was to build trees from the top, root node, down, accumulating the cost as levels were added. We will follow the same approach in this correspondence to

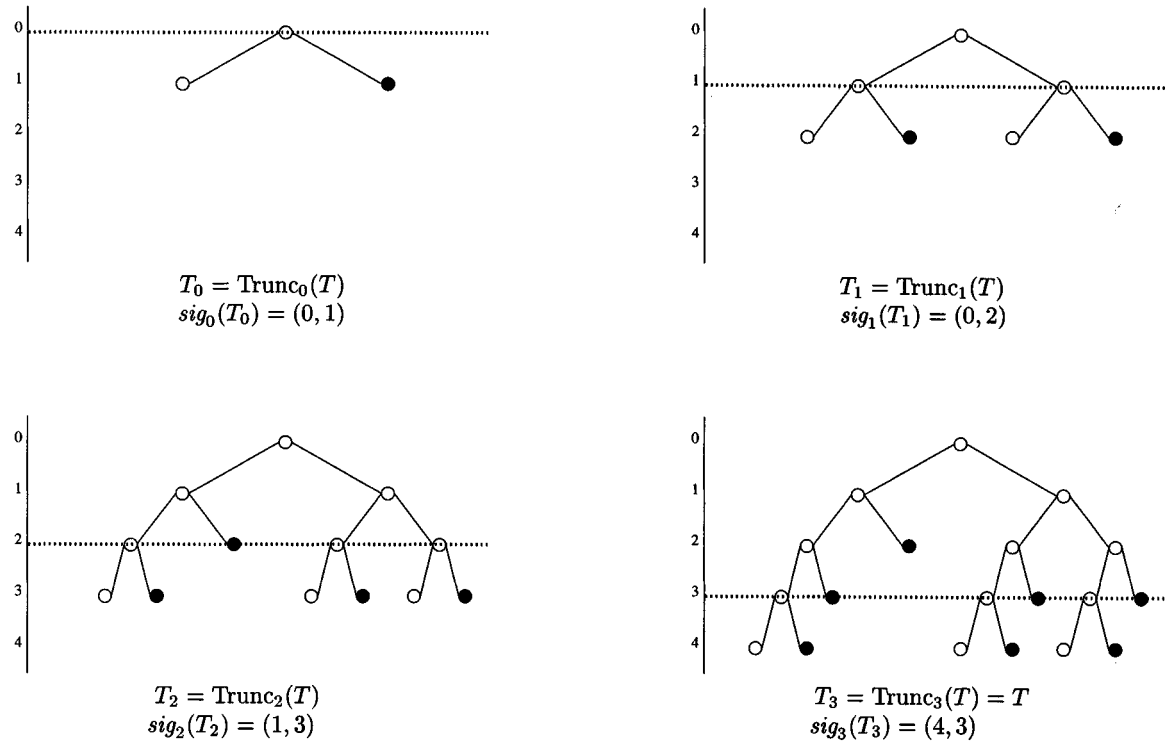

Fig. 4. The trees T_0, T_1, T_2, T_3 are the truncations of the right tree in Fig. 2 which we will call T. Each T_i is an i-level tree for that value of i and the dotted horizontal line across each tree is the truncation level. Note that $T = \text{Trunc}_4(T)$ with $\text{sig}_4(T_4) = (7, 0)$.

construct min-cost feasible trees. Since Lemma 1 guarantees that trees thus constructed will be min-cost trees among *all* trees we will have solved the problem.

To use this approach we need to define the following.

Definition 4: Let T be a tree and i a nonnegative integer. The ith-level truncation of T is the tree $\text{Trunc}_i(T)$ containing all nodes in T of depth at most $i + 1$

$$\text{Trunc}_i(T) = \{u \in T : \text{depth}(u) \leq i + 1\}.$$

A tree T is an *i-level tree* if all the internal nodes $v \in T$ satisfy $\text{depth}(v) \leq i$.

See Fig. 4 for examples. We note that $\text{Trunc}_i(T)$ is always an i-level tree and that truncation preserves feasibility, i.e., if T is a feasible tree then $\text{Trunc}_i(T)$ is also a feasible tree. We also note that if T has depth d then $\forall i \geq d - 1$, $\text{Trunc}_i(T) = T$.

The dynamic programming algorithm will strongly use the idea of subproblem optimality, i.e., if a feasible tree T is optimal then all of its i-level truncations $\text{Trunc}_i(T)$ are also optimal. In order for this observation to make sense we must define what it means for a feasible i-level tree (that might have fewer than n right leaves) to be optimal. That is, we must define a cost function on i-level trees.

Definition 5: Let T be a feasible i-level tree. The i-level signature of T is an ordered pair

$$\text{sig}_i(T) = (m, b)$$

in which

$$m = |\{v \in T : v \text{ is a right leaf}, \text{depth}(v) \leq i|$$

is the number of right leaves in T with depth at most i and

$$b = |\{v \in T : v \text{ is a right leaf}, \text{depth}(v) = i + 1\}|$$

is the number of right leaves in T at level $i + 1$ (bottom level). Note that there are $2b$ (left and right) leaves at level $i + 1$.

Now let T be an i-level tree with $\text{sig}_i(T) = (m, b)$ with $m \leq n$. The i-level *partial cost* of T is

$$\text{Cost}_i(T) = \sum_{t=1}^{m} \text{depth}(v_t) \cdot p_t + i \cdot \sum_{t=m+1}^{n} p_t \tag{1}$$

where $v_1, \cdots, v_m$ are the m highest right leaves of T ordered by depth, e.g., those with $\text{depth} \leq i$.

Note that $\text{Cost}_i(T)$ is not only dependent upon T but also upon i. For example, the right tree T in Fig. 2 is both a three-level and a four-level tree; $\text{sig}_3(T) = (4, 3)$ and $\text{sig}_4(T) = (7, 0)$. Its associated i-level costs are

$$\text{Cost}_3(T) = 2p_1 + 3(p_2 + p_3 + p_4) + 3(p_5 + p_6 + p_7)$$
$$\text{Cost}_4(T) = 2p_1 + 3(p_2 + p_3 + p_4) + 4(p_5 + p_6 + p_7)$$

which are obviously not the same.

We can now define what it means for a tree with fewer than n right leaves to be optimal.

Definition 6: Let (m, b) be a valid signature, i.e., $m, b \geq 0$. Set $\text{OPT}[m, b]$ to be the minimum cost over all i and all feasible i-level trees T with signature (m, b). More precisely

$$\text{OPT}[m, b] = \min\{\text{Cost}_i(T) : \exists i, T, \\ T \text{ is a feasible } i\text{-level tree with } \text{sig}_i(T) = (m, b)\}$$

A tree T is *min-cost* or *optimal* if, for some i, it is an i-level tree, $\text{sig}_i(T) = (m, b)$ and $\text{Cost}_i(T) = \text{OPT}[m, b]$.

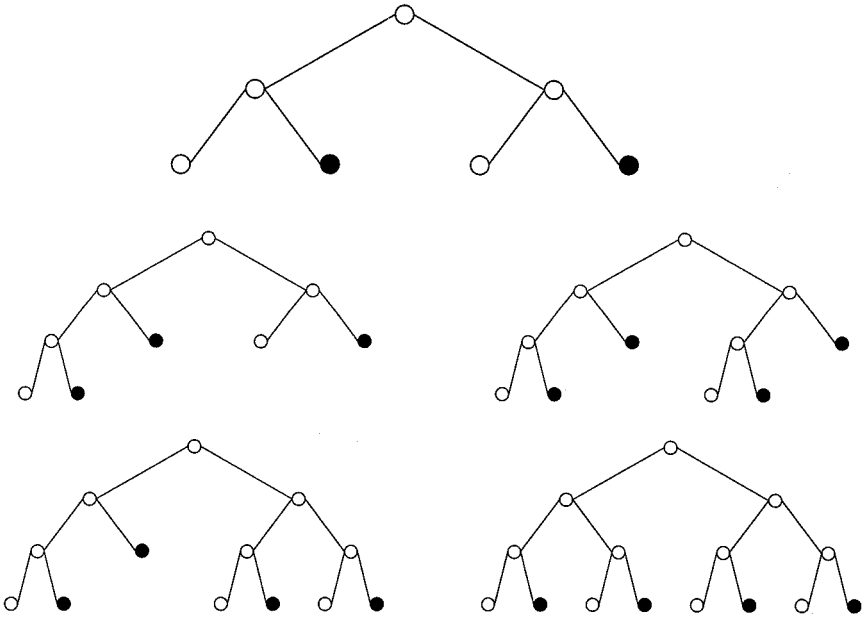

Fig. 5. The tree in the top row is our original tree T which is a one-level tree with $\mathrm{sig}_1(T) = (0, 2)$. The next two rows are the four possible expansions $\mathrm{Expand}\,(T, 1)$, $\mathrm{Expand}\,(T, 2)$, $\mathrm{Expand}\,(T, 3)$, and $\mathrm{Expand}\,(T, 4)$. We do not draw $\mathrm{Expand}\,(T, 0)$ which is simply T.

Note that if T is a feasible tree with n right leaves and depth $d \leq i$ then $\mathrm{sig}_i(T) = (n, 0)$ so, if T' is an optimal feasible tree (with n right leaves), then

$$\mathrm{OPT}\,[n, 0] = \mathrm{Cost}_i(T') = \sum_{t=1}^{n} \mathrm{depth}\,(v_t) \cdot p_t.$$

Thus $\mathrm{OPT}\,[n, 0]$ is exactly the cost of the optimal tree that we are trying to calculate. We will calculate its value by using a dynamic programming approach to fill in the OPT table. Backtracking the dynamic programming will permit us to construct T'.

Before continuing we briefly digress to explain why we defined $\mathrm{OPT}\,[m, b]$ to be the minimum cost only among *feasible* trees and not among all trees.[3] The reason is that we will be building optimal trees level-by-level. Since Lemma 1 tells us that our final result is a feasible tree and we know that all truncations of feasible trees are feasible trees our construction will work by building feasible trees level by level, always storing the min-cost ones.

Now suppose that T is an i-level tree with $\mathrm{sig}_i(T) = (m, b)$. What feasible $(i+1)$-level trees can T be grown into? The only way to grow a feasible tree is by making some of the $2b$ nodes on level $i+1$ internal and making the remainder of the nodes leaves. From Lemma 1 we know that all of the left nodes must be made internal before any of the right ones are. We therefore define an *Expansion* operator as follows.

Definition 7: Let T be an i-level tree with $\mathrm{sig}_i(T) = (m, b)$. Let $0 \leq q \leq 2b$. The qth expansion of T is the tree

$$T' = \mathrm{Expand}\,(T, q)$$

constructed by making q of the leaves at level $i+1$ (bottom level) of T internal nodes as follows:

- if $q \leq b$, make q left nodes at level $i+1$ internal.
- if $q > b$, make all b left nodes and $q - b$ right nodes at level $i+1$ internal.

[3]The algorithm to be presented actually remains correct even if we optimized over *all* trees and not just all feasible ones. The reason for the restriction to feasible trees is that it makes the result both easier to understand and prove.

In Fig. 5 we see a tree and all of its expansions.

Once q is fixed both $\mathrm{Cost}_{i+1}(T')$, the number of nodes at level $i+2$, and the signature $\mathrm{sig}_{i+1}(T')$ of T' can be found.

Lemma 2: Suppose T is an i-level tree with $\mathrm{sig}_i(T) = (m, b)$. Let $T' = \mathrm{Expand}\,(T, q)$ be its qth expansion. Then T' is an $i+1$-level tree with

$$\mathrm{Cost}_{i+1}(T') = \mathrm{Cost}_i(T) + \sum_{m < t \leq n} p_t$$

and

- if $0 \leq q \leq b$ then $\mathrm{sig}_{i+1}(T') = (m + b, q)$,
- if $b + 1 \leq q \leq 2b$, then $\mathrm{sig}_{i+1}(T') = (m + 2b - q, q)$.

Proof: Let $(m', b') = \mathrm{sig}_{i+1}(T')$. Since T' has exactly $m' - m$ right leaves on level $i+1$ we find

$$\begin{aligned}
\mathrm{Cost}_{i+1}(T') &= \sum_{t=1}^{m'} \mathrm{depth}(v_t) \cdot p_t + (i+1) \cdot \sum_{t=m'+1}^{n} p_t \\
&= \sum_{t=1}^{m} \mathrm{depth}(v_t) \cdot p_t + (i+1) \cdot \sum_{t=m+1}^{m'} p_t \\
&\quad + (i+1) \cdot \sum_{t=m'+1}^{n} p_t \\
&= \mathrm{Cost}_i(T) + \sum_{m < t \leq n} p_t.
\end{aligned}$$

The proof of the second part of the lemma follows directly from the definition of the *Expand* operator. □

This lemma tells us that to calculate the extra cost added by a level-i expansion of T and the signature of the new expanded tree it is not necessary to know T or i but only $\mathrm{sig}_i(T)$. We can, therefore, define a recurrence relationship for calculating $\mathrm{OPT}\,[.,.]$. In what follows $\mathcal{M}(m', b')$ is exactly the set of signatures (m, b) that have some expansion with signature (m', b').

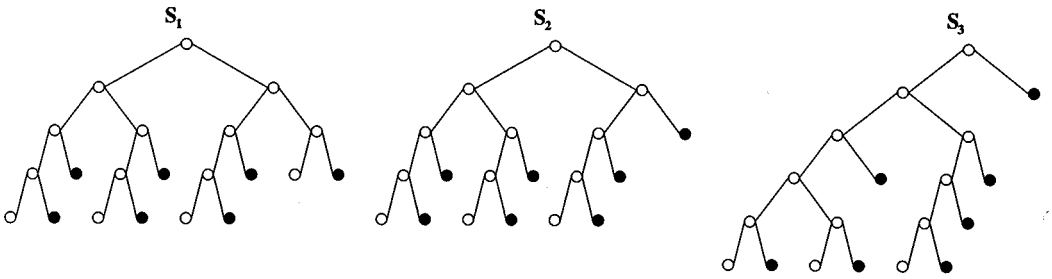

Fig. 6. These three trees have the property that $\mathrm{sig}_3(S_1) = \mathrm{sig}_3(S_2) = \mathrm{sig}_4(S_3) = (4,3)$. In fact, these three trees are the only way to realize the signature $(4,3)$ and thus $\mathcal{M}(4,3) = \{(0,4),(1,3),(3,2)\}$.

Lemma 3: Set

$$\mathcal{M}(m',b') = \Big\{(m,b) : (m,b) \neq (m',b') \text{ and } \exists q \text{ s.t. } 0 \le q \le b \text{ and } (m',b') = (m+b,q) \text{ or } \exists q \text{ s.t. } b+1 \le q \le 2b \text{ and } (m',b') = (m+2b-q,q)\Big\}.$$

Then

$$\mathrm{OPT}\,[0,1] = 0 \tag{2}$$

and, for $(m',b') \neq (0,1)$

$$\mathrm{OPT}\,[m',b'] = \min_{(m,b)\in\mathcal{M}(m',b')} \left\{ \mathrm{OPT}\,[m,b] + \sum_{m<t\le n} p_t \right\}. \tag{3}$$

Proof: Fig. 6 illustrates an example of $\mathcal{M}(m',b')$.

To prove (2) we note that the *only* feasible tree with signature $(0,1)$ is the 0-level tree consisting of the root and its two children and this tree has 0-level cost 0.

To prove (3) first suppose that $\mathrm{OPT}\,[m',b']$ is realized by an $i+1$-level tree T' with $\mathrm{sig}_{i+1}(T') = (m',b')$ and

$$\mathrm{Cost}_{i+1}(T') = \mathrm{OPT}\,[m',b'].$$

Now set $T = \mathrm{Trunc}_i(T')$ and let q be the number of internal nodes on level $i+1$ of T. Also set $(m,b) = \mathrm{sig}_i(T)$. Then, by the definition of the *Trunc* and *Expand* operators we have that $T' = \mathrm{Expand}\,(T,q)$. Thus from Lemma 2 and the definition of $\mathrm{OPT}\,[,]$

$$\begin{aligned}\mathrm{OPT}\,[m',b'] &= \mathrm{Cost}_{i+1}(T') \\ &= \mathrm{Cost}_i(T) + \sum_{m<t\le n} p_t \\ &\ge \mathrm{OPT}\,[m,b] + \sum_{m<t\le n} p_t \\ &\ge \min_{(m,b)\in\mathcal{M}(m',b')} \left\{ \mathrm{OPT}\,[m,b] + \sum_{m<t\le n} p_t \right\}.\end{aligned}$$

To see the other direction let $(m,b) \in \mathcal{M}(m',b')$ and set T to be such that

$$\mathrm{sig}_i(T) = (m,b) \quad \text{and} \quad \mathrm{OPT}\,[m,b] = \mathrm{Cost}_i(T).$$

Let q be such that

$$\mathrm{sig}_{i+1}\mathrm{Expand}(T,q) = (m',b').$$

Such a q must exist by the definition of $\mathcal{M}(m',b')$. Let $T' = \mathrm{Expand}\,(T,q)$. Then from Lemma 2 $\mathrm{sig}_{i+1}(T') = (m',b')$ and

$$\begin{aligned}\mathrm{Cost}_{i+1}(T') &= \mathrm{Cost}_i(T) + \sum_{m<t\le n} p_t \\ &= \mathrm{OPT}\,[m,b] + \sum_{m<t\le n} p_t.\end{aligned}$$

Since this is true for every $(m,b) \in \mathcal{M}(m',b')$ we thus find that

$$\mathrm{OPT}\,[m',b'] \le \min_{(m,b)\in\mathcal{M}(m',b')} \left\{ \mathrm{OPT}\,[m,b] + \sum_{m<t\le n} p_t \right\}$$

completing the proof. □

IV. The Algorithm

Using Lemma 3 we can directly design an algorithm for calculating the $\mathrm{OPT}\,[,]$ values and constructing an optimal tree. Code for the algorithm is given in Fig. 7 and a worked example is shown in Table I and Fig. 8. In the algorithm, the entry $Q[m',b']$ stores the pair $(m,b) \in \mathcal{M}(m',b')$ such that

$$\mathrm{OPT}\,[m',b'] = \mathrm{OPT}\,[m,b] + \sum_{m<t\le n} p_t.$$

We will now prove the correctness of the algorithm and then show that it runs in $O(n^3)$ time. We start by recalling the definition of a *lexicographical ordering* on pairs.

Definition 8: Let $(m,b),(m',b')$ be given. Then (m,b) is *lexicographically smaller* than (m',b')

$$(m,b) \prec (m',b')$$

if and only if

$$m < m' \quad \text{or} \quad m = m' \quad \text{and} \quad b < b'.$$

It is now easy to see that

Lemma 4: Let $(m,b),(m',b')$ be signatures such that $(m,b) \in \mathcal{M}(m',b')$. Then $(m,b) \prec (m',b')$.

Proof: This follows from the definition of $\mathcal{M}(m',b')$. We first point out that $b \neq 0$ because it is impossible for $(m,b) \in \mathcal{M}(m',b')$ if $b = 0$. Thus $b \ge 1$.

There are two cases. In the first case $\exists q \le b$ such that $(m',b') = (m+b,q)$. In this case, since $b \ge 1$ we have that $m < m'$ so $(m,b) \prec (m',b')$.

In the second case, $\exists q$ with $b+1 \le q \le 2b$ such that $(m',b') = (m+2b-q,q)$. In this case, if $q < 2b$ then again $m < m'$ so $m < m'$

The Algorithm

Initialize the $OPT[,]$ table
$\forall m, n,\ 0 \le m \le n,\ 1 \le b \le n-m$,
  Set $OPT[m,b] := \infty$;
$\forall m, 0 \le m \le n$ Set $P_m := \sum_{m<t\le n} p_t$;
$OPT[0,1] := 0$; $OPT[n,0] := \infty$;

Calculate $OPT[,]$ values
for $m := 0$ *to* n
  for $b := 1$ *to* $n-m$
    Process the pair (m,b)
    for $q := 0$ *to* b
      $X := \min(OPT[m,b] + P_m,\ OPT[m+b,q])$
      if $X < OPT[m+b,q]$
        $\{OPT[m+b,q] := X;\quad Q[m+b,q] := (m,b);\}$
    for $q := b+1$ *to* $2b$
      $X := \min(OPT[m,b] + P_m,\ OPT[m+2b-q,q])$
      if $X < OPT[m+2b-q,q]$
        $\{OPT[m+2b-q,q] := X;$
          $Q[m+2b-q,q] := (m,b);\}$

Backtracking and outputting tree
$m := n$; $b := 0$;
repeat $\{(m,b) = Q[m,b]$; print $q;\}$
  until $(m,b) = (1,0)$

Fig. 7. The dynamic programming algorithm plus backtracking. The algorithm will output the number of right leaves on every level of some optimal tree.

TABLE I
VALUES FOR $n = 7$ WITH WEIGHTS $7, 6, 5, 4, 3, 2, 1$

	$b=0$	1	2	3	4	5	6	7
$m=0$	∞	**0**	**28** (**0,1**)	∞	56 (0,2)	∞	∞	∞
1	28 (0,1)	28 (0,1)	49 (1,1)	56 (0,2)	70 (1,2)	∞	77 (1,3)	
2	49 (1,1)	49 (1,1)	**56** (**0,2**)	70 (1,2)	71 (2,2)	77 (1,3)		
3	64 (2,1)	64 (2,1)	70 (1,2)	71 (2,2)	77 (1,3)			
4	71 (2,2)	71 (2,2)	**71** (**2,2**)	77 (1,3)				
5	77 (4,1)	77 (4,1)	80 (3,2)					
6	77 (4,2)	**77** (**4,2**)						
7	**78** (**6,1**)							

so $(m,b) \prec (m',b')$. If $q = 2b$ then $m' = m + 2b - q = m$ but then $b < 2b = b'$ so we still have $(m,b) \prec (m',b')$. □

We now show that the algorithm correctly fills in the $\mathrm{OPT}\,(m,b)$ values (using a standard dynamic programming correctness proof).[4]

[4]We quickly sketch a second way of proving this, one similar to that developed in [7]. Essentially, one can create a weighted graph $G = (V,E)$ in which V in the set of all signatures and $((m,b),(m',b')) \in E$ if $(m,b) \in \mathcal{M}(m',b')$. The weight of this edge is defined to be $\sum_{m<t\le n} p_t$. Then $\mathrm{OPT}\,[m,b]$ can be shown to be equal to the cost of the *minimum-cost path* connecting $(0,1)$ to (m,b) in G. The graph G can be shown to be acyclic and the algorithm presented in Fig. 8 is exactly the code for finding shortest paths in a directed acyclic graph, see, e.g., [8].

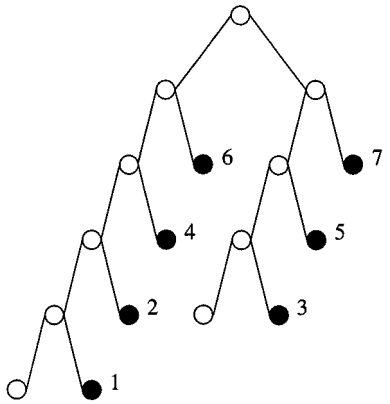

Fig. 8. An optimal tree for $n = 7$ with weights $7, 6, 5, 4, 3, 2, 1$. This tree is derived from Table I.

Note that if $(m,b) \in \mathcal{M}(m',b')$ then either $(m',b') = (m+b,q)$ or $(m',b') = (m+2b-q,q)$. In both of these cases we find that $m+b \le m'+b'$. Now note that the set of signatures processed by the algorithm is exactly

$$\begin{aligned}\mathcal{M} &= \{(m,b) : 0 \le m \le n,\ 0 \le b \le n-m\}\\ &= \{(m,b) : 0 \le m, b,\ m+b \le n\}.\end{aligned}$$

Thus if $(m',b') \in \mathcal{M}$ and $(m,b) \in \mathcal{M}(m',b')$ then $m+b \le m'+b' \le n$ so $(m,b) \in \mathcal{M}$. Therefore, $\mathcal{M}(m',b') \subseteq \mathcal{M}$.

Next note that the algorithm actually processes the signatures in $\mathcal{M}$ in *lexicographical order* so, from Lemma 4, *all* signatures in $\mathcal{M}(m',b')$ are processed before (m',b') and no such signatures are processed after it.

Correctness now follows by induction on (m',b'), the induction order being the lexicographic order. The induction hypothesis is that, *at the time immediately preceding the processing of* (m',b') *the value of* $\mathrm{OPT}\,[m,b]$ *will already have been correctly set.*

The first signature processed is $(0,1)$ and since $\mathrm{OPT}\,[0,1]$ is originally set to 0 and never changed afterwards, the statement is correct for $(0,1)$. Now suppose all signatures preceding (m',b') in the lexicographic order have already been processed and it is now the turn of (m',b'). In particular, this implies that all (m,b) with $(m,b) \in \mathcal{M}(m',b')$ have already been processed. By the induction hypothesis, at the time such an (m,b) was processed $\mathrm{OPT}\,[m,b]$ was already correctly set. Thus the statement executed at the time of processing (m,b) was equivalent to

$$\mathrm{OPT}\,[m',b'] = \min\{\mathrm{OPT}\,[m',b'], \mathrm{OPT}\,[m,b] + P_m\}.$$

Since this is done for all $(m,b) \in \mathcal{M}(m',b')$ but no other (m,b)'s, the value stored in $\mathrm{OPT}\,[m',b']$ is exactly

$$\min_{(m,b)\in\mathcal{M}(m',b')} \left\{ \mathrm{OPT}[m,b] + \sum_{m<t\le n} p_t \right\},$$

completing the proof that the $\mathrm{OPT}\,[,]$ table is filled in correctly.

While filling in the table, the algorithm also keeps track of where the optima came from by storing the appropriate $Q[m',b'] = (m,b)$ such that

$$\mathrm{OPT}[m',b'] = \mathrm{OPT}[m,b] + \sum_{m<t\le n} p_t.$$

After filling in the table, the algorithm then uses the $Q[,]$ table to backtrack and print out the number of right nodes on every level of the

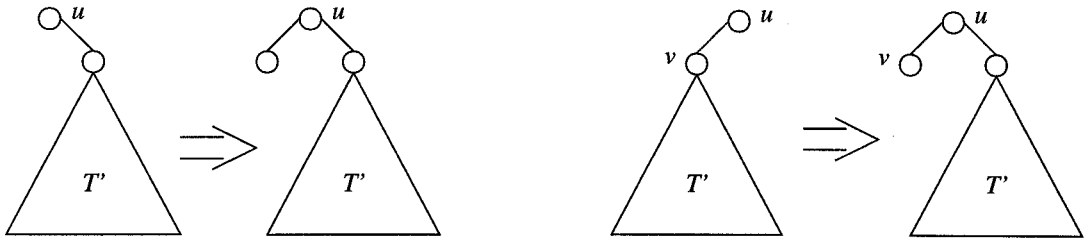

Fig. 9. Cases I and II in the proof.

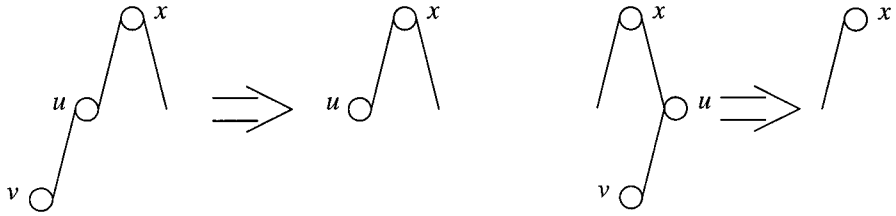

Fig. 10. Cases III and IV in the proof.

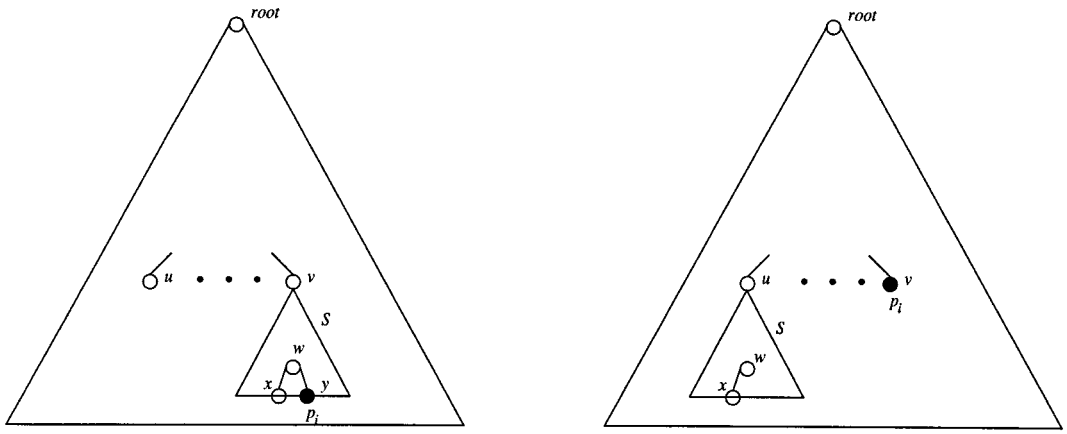

Fig. 11. Illustration of why some optimal tree must be feasible.

optimal tree. Since the tree is full with every right node being matched by some left node this gives the full tree and we are done.

Finally, we note that for each of the $O(n^2)$ signatures (m, b) generated, the algorithm does $O(n)$ work (the two **for** loops over q). Thus the algorithm runs in $O(n^3)$ total time.

Table I contains a worked example for $n = 7$ with weights $7, 6, 5, 4, 3, 2, 1$. The top element in each entry is $\mathrm{OPT}[m, b]$ and the bottom one is $Q[m, b]$. The ∞ entries are signatures that are unrealizable by any feasible tree. The boldface entries are the ones that correspond to the optimal tree found by the backtracking section of Fig. 7. Reading them off we find that the number of right nodes on the levels of the optimal tree are, starting from the top level and working down, $1, 2, 2, 2, 1$. The tree itself can be seen in Fig. 8. Note that this tree has (optimal) cost 78 as compared to the trees in Fig. 2. These have cost 83 for the same weights

V. Conclusion

In this correspondence we have shown that it is possible to calculate optimal one-ended binary prefix-free codes in $O(n^3)$ time improving upon the old exponential time algorithms. Our approach used dynamic programming on an appropriate subproblem space. The main open question is whether it is possible to improve the algorithm, perhaps even to $O(n)$ time, matching the linear time used by the standard Huffman-encoding algorithm.

Appendix

In this section we prove Lemma 1, that there is always a feasible optimal tree.

Proof (of Lemma 1): We first show that there exists an optimal full tree. If T is a tree and $u \in T$ an internal node we will call u *bad* if it has only one child. If a tree has no bad nodes it is a full tree.

Let B be the minimal number of bad nodes an optimal tree can have. If $B = 0$ there is a full optimal tree and we are done. Otherwise, let T be an optimal tree with B bad nodes and the fewest total number of nodes among all optimal trees with B bad nodes. We will show a contradiction by building a new optimal tree with fewer bad nodes or the same number of bad nodes and fewer total nodes.

Let u be a highest bad node in T. Note that u cannot be the root because if the root were bad we could simply erase it; its (only) child then becomes the root of the new tree and, since the depth of every

leaf has been decreased by 1, this new tree is cheaper than the old one, contradicting optimality.

In what follows refer to Figs. 9 and 10 for illustration as we do a case-by-case analysis.

(Case I) If u had a right child but no left one we could simply add its left child to get a new tree with the same cost but fewer bad nodes, contradicting the definition of T. Thus u must have a left child v but no right child. There are two cases.

(Case II) If v is the root of some tree T' then we could move T' to be rooted at the right child of u and leave v a leaf. The new resulting tree has the same cost but fewer bad nodes, again leading to a contradiction.

Otherwise, v is itself a leaf. Let x be the parent of u.

(Case III) If u is a left child of x then we simply remove v, leaving u as a left leaf. The cost of the resulting tree is the same as before but it has one fewer bad node. Again a contradiction.

(Case IV) Otherwise, u is the right child of x and removing u could add a new right child to the tree, possibly even raising its cost. Therefore, in this case we remove *both* u and v. Since x was not bad before (because it is higher than u) removing u does not add a new right leaf to the tree so the cost of the resulting tree remains the same. Since x has now become bad the new tree still has B bad nodes but it has fewer total nodes than T, again causing a contradiction.

We have just seen that there exists some optimal full tree T. We now prove that T is feasible. See Fig. 11 for illustration.

Suppose T is not feasible. Then there exists some right internal node $v \in T$ and left leaf $u \in T$ such that $\mathrm{depth}\,(v) = \mathrm{depth}\,(u)$. Let S be the subtree rooted at v, y the deepest right node $y \in S$, and x the left sibling of y (x and y must exist because T is full). Also suppose that probability p_i is assigned to y. Now detach S from v and attach it to u, erase y and assign p_i to node v. Denote the new tree thus created by T'. Since the only probability whose assigned right leaf has changed is p_i we find that

$$\mathrm{Cost}\,(T') = \mathrm{Cost}\,(T) + (\mathrm{depth}\,(v) - \mathrm{depth}\,(y))p_i.$$

But $\mathrm{depth}\,(v) < \mathrm{depth}\,(y)$ so $\mathrm{Cost}\,(T') < \mathrm{Cost}\,(T)$ contradicting optimality of T. Thus T must be feasible. □

REFERENCES

[1] T. Berger and R. W. Yeung, "Optimum "1"-ended binary prefix codes," *IEEE Trans. Inform. Theory*, vol. 36, pp. 1435–1441, Nov. 1990.

[2] C. Szelok, "Variations of prefix free codes," M.Phil. thesis, Dept. Comput. Sci., Hong Kong Univ. Sci. Technol., Dec. 1997.

[3] R. M. Capocelli, A. D. Santis, L. Gargano, and U. Vaccaro, "On the construction of statistically synchronizable codes," *IEEE Trans. Inform. Theory*, vol. 38, pp. 407–414, Mar. 1992.

[4] R. M. Capocelli, A. D. Santis, and G. Persiano, "Binary prefix codes ending in a "1"," *IEEE Trans. Inform. Theory*, vol. 40, pp. 1296–1302, July 1994.

[5] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. Boston, MA: MIT Press, 1993.

[6] S. Even, *Graph Algorithms*. Rockville, MD: Comput. Sci. Press, 1979.

[7] M. Golin and G. Rote, "A dynamic programming algorithm for constructing optimal prefix-free codes for unequal letter costs," *IEEE Trans. Inform. Theory*, vol. 44, pp. 1770–1781, Sept. 1998.

[8] K. Mehlhorn, *Data Structures and Algorithms 2: Graph Algorithms and NP-Completeness*. Berlin, Germany: Springer-Verlag, 1984.

A Quantum Analog of Huffman Coding

Samuel L. Braunstein, Christopher A. Fuchs, Daniel Gottesman, and Hoi-Kwong Lo

***Abstract*—We analyze a generalization of Huffman coding to the quantum case. In particular, we notice various difficulties in using instantaneous codes for quantum communication. Nevertheless, for the *storage* of quantum information, we have succeeded in constructing a Huffman-coding-inspired quantum scheme. The number of computational steps in the encoding and decoding processes of N quantum signals can be made to be of polylogarithmic depth by a massively parallel implementation of a quantum gate array. This is to be compared with the $O(N^3)$ computational steps required in the sequential implementation by Cleve and DiVincenzo of the well-known quantum noiseless block-coding scheme of Schumacher. We also show that $O(N^2(\log N)^\alpha)$ sequential computational steps are needed for the *communication* of quantum information using another Huffman-coding-inspired scheme where the sender must disentangle her encoding device before the receiver can perform any measurements on his signals.**

***Index Terms*—Data compression, Huffman coding, instantaneous codes, quantum coding, quantum information, variable-length codes.**

I. INTRODUCTION

There has been much recent interest in the subject of quantum information processing. Quantum information is a natural generalization of classical information. It is based on quantum mechanics, a well-tested scientific theory in real experiments. This correspondence concerns quantum information.

The goal of this correspondence is to find a quantum source coding scheme analogous to Huffman coding in the classical source coding theory [3]. Let us recapitulate the result of classical theory. Consider the simple example of a memoryless source that emits a sequence of independent and identically distributed signals each of which is chosen from a list $w_1, w_2, \cdots, w_n$ with probabilities $p_1, p_2, \cdots, p_n$. The task of source coding is to store such signals with a minimal amount of resources. In classical information theory, resources are measured in bits. A standard coding scheme to use is the optimally efficient Huffman coding algorithm, which is a well-known lossless coding scheme for data compression.

Apart from being highly efficient, it has the advantage of being instantaneous, i.e., unlike block coding schemes, the encoding and decoding of each signal can be done immediately. Note also that codewords of variable lengths are used to achieve efficiency. As we will see

Manuscript received May 7, 1999; revised January 11, 2000. This work was supported in part by EPSRC under Grants GR/L91344 and GR/L80676, by the Lee A. DuBridge Fellowship, by DARPA under Grant DAAH04-96-1-0386 through the Quantum Information and Computing (QUIC) Institute administered by ARO, and by the U.S. Department of Energy under Grant DE-FG03-92-ER40701. The work of H.-K. Lo was performed while the author was with Hewlett-Packard Laboratories, Bristol, U.K.

S. L. Braunstein is at SEECS, University of Wales, Bangor LL57 1UT, U.K., and at Hewlett-Packard Labs, Filton Road, Stoke Gifford, Bristol BS34 8QZ, U.K. (e-mail: schmuel@sees.bangor.ac.uk).

C. A. Fuchs is at the Norman Bridge Laboratory of Physics 12-33, California Institute of Technology, Pasadena, CA 91125 USA.

D. Gottesman was with the California Institute of Technology, Pasadena and with the Los Alamos National Laboratory. He is now with Microsoft Research, Microsoft Corporation, Redmond, WA 98052 USA.

H.-K. Lo is with MagiQ Technologies, Inc., New York, NY 10001 USA (e-mail: hkl@magiqtech.com).

Communicated by A. M. Barg, Associate Editor for Coding Theory.

Publisher Item Identifier S 0018-9448(00)04288-7.