When Huffman Meets Hamming: A Class of Optimal Variable-Length Error Correcting Codes

Serap A. Savari
Texas A&M University
College Station, TX
savari@ece.tamu.edu

Jörg Kliewer
New Mexico State University
Las Cruces, NM
jkliewer@nmsu.edu

Abstract

We introduce a family of binary prefix condition codes in which each codeword is required to have a Hamming weight which is a multiple of w for some integer $w \geq 2$. Such codes have intrinsic error resilience and are a special case of codes with codewords constrained to belong to a language accepted by a deterministic finite automaton. For a given source over n symbols and parameter w we offer an algorithm to construct a minimum-redundancy code among this class of prefix condition codes which has a running time of $O(n^{w+2})$.

1. Introduction

In recent years the study of joint source-channel coding with variable-length codes has gained interest because of the prospect of designing communication systems with the same performance but less complexity than systems which completely separate the source and channel coding functions. Two of the early papers on this topic are [1] and [2], and the recent paper [3] offers an overview of numerous papers in the area. The problem we address in this paper is motivated by an algorithm proposed in [3].

Assume a memoryless source X over the alphabet $\mathcal{X} = \{1,\ 2,\ \ldots,\ n\}$, and let $p_i,\ i \in \mathcal{X}$, denote the probability of symbol i. For convenience suppose that $p_1 \geq p_2 \geq\ \ldots\ \geq p_n \geq 0$. We are given a positive integer $w \geq 2$. The *Hamming weight* of a binary string is the number of ones in the string. An *optimal* or *minimum-redundancy* code over a class of codes is one which minimizes the average codeword length. We seek an optimal binary prefix condition code with the constraint that each codeword has a Hamming weight which is a multiple of w. Observe that any sequence of codewords from such a code has a Hamming weight which is a multiple of w. Therefore, the Hamming distance $d_{\min}$ or number of positions in which two distinct sequences of codewords of the same length differ must be at least two. This property introduces error resilience into the output of the source encoder, and we shall see shortly that for small values of w the optimal code does not sacrifice much in terms of compression performance. The paper [3] offers a simple heuristic for this problem when $w = 2$, i.e., when each codeword is required to have even parity.

Let $l_i,\ i \in \mathcal{X}$, be the length of the codeword assigned to source symbol i. It is possible to show that when $w = 2$, the necessary and sufficient conditions for the existence of a binary prefix condition code with even parity codewords of length $l_1,\ l_2,\ \ldots,\ l_n$ is identical to the conditions satisfied by a binary prefix condition code with codeword lengths $l_1,\ l_2,\ \ldots,\ l_n$ in which every codeword is constrained to end with the binary symbol "1." The problem of constructing minimum-redundancy "1"-ended binary prefix condition codes was first posed about 20 years in [4], and there are two polynomial-time algorithms [5], [6], [7] to solve it which are based on dynamic programming and which use a well-known connection between prefix condition codes and binary trees. In each case the solution is obtained by taking advantage of properties of certain subtrees of the binary code tree corresponding to the optimal prefix condition code. When $w = 2$, it is straightforward to modify the algorithms to find an optimal "1"-ended prefix condition code to obtain an optimal even parity prefix condition code. Moreover,

Theorem 1: The expected length $E[L]$ of an optimal even parity prefix condition code for a memoryless source X with binary entropy $H(X)$ and with minimum source symbol probability p_n satisfies

$$H(X) + p_n \leq E[L] < H(X) + 1.$$

Next assume that $w > 2$ and the maximum symbol probability of source X is p_1. The expected length $E[L]$ of an optimal prefix condition code in which each codeword has a Hamming weight which is a multiple of w satisfies

$$E[L] < \begin{cases} H(X) + (w-1)(1-p_1) + p_1 + \log_2((2\log_2 e)/e), & p_1 < 0.5 \\ H(X) + w(1-p_1) + 1 + p_1 \log_2 p_1 + (1-p_1)\log_2(1-p_1), & p_1 \geq 0.5 \end{cases}$$

Proof: (Sketch) The bounds when $w = 2$ follow from results in [4] and [8] on the performance of optimal "1"-ended codes. The bound when $w > 2$ can be obtained by determining the Huffman codeword lengths for source X, setting the shortest Huffman codeword to the all-zero word and then adding at most $w - 1$ "1"-bits to the ends of each remaining codeword as needed to ensure that each codeword has Hamming weight which is a multiple of w, and using the redundancy analysis of [9]. $\square$

We comment that the problem we consider here is a special case of the problem to construct minimum-redundancy prefix condition codes when the codewords are constrained to belong to a language accepted by a deterministic finite automaton [10, §1.1]. It is not known in general how to efficiently solve this class of problems. The papers [11] and [12] consider a larger set of languages than the ones we discuss here but restrict attention to sources for which $p_i = 1/n$ for all i.

2. The Algorithm

We next adapt the argument and algorithm of [5] to find a minimum-redundancy prefix condition code where all codewords have a Hamming weight which is a multiple of w for some integer $w \geq 2$. The running time of the algorithm we propose will be $O(n^{w+2})$ for $w > 2$ and $O(n^3)$ when $w = 2$. In the case $w = 2$, a modification of the procedure in [6] has running time $O(n^2)$. One of the key assumptions of [6] does not appear to extend simply when $w > 2$, but it is possible to adapt the scheme of [6] for the case where $w = 3$.

We begin by reviewing the representation of prefix condition codes by rooted binary code trees, and we use terminology from [5] and [9]. While we only discuss binary code alphabets, it is simple to extend the code construction here to D-ary code alphabets if the goal is to find minimum-redundancy prefix condition codes with codewords each having symbols which sum to a multiple of w. A binary code tree is a labeled tree consisting of nodes and edges. If two nodes are adjacent along a path, then the one closer to the root is called the *parent* of the other, and the second node is dubbed a *child* of the parent node. Each edge of a binary code tree is labeled with a binary digit, and if two edges share an endpoint then they have different labels. The binary word associated with a node in the tree is the sequence of labels on the path from the root to the node; the root node corresponds to the empty binary string. A node in a binary code tree may have no children, in which case it is called a *leaf* node, or one child or two children. Two nodes which share a common parent are called *siblings*. A node with at least one child is called an *intermediate* node. In a binary prefix condition code each letter of the source alphabet is represented by a different leaf and its corresponding binary string from the binary code tree. A binary code tree in which every intermediate node has two children is said to be *full*.

Following [5] and [12], we will be particularly interested in two functions of a node v of a binary code tree. The first is the *depth* of node v, denoted $depth(v)$, which is the number of symbols in the binary word corresponding to v. The second is the *type* of node v, denoted $type(v)$, which is the Hamming weight modulo w of the binary word associated with v; i.e., $type(v) \in \{0,\ 1,\ \ldots,\ w-1\}$ for every node v in the binary code tree. Observe that the root node satisfies $depth(root) = type(root) = 0$ and the sibling of a type-0 node other than the root would either be of type 1 or of type $w-1$. Let $C = \{\sigma_1,\ \sigma_2,\ \ldots,\ \sigma_n\}$ be a prefix condition code for which all codewords have a weight which is a multiple of w; i.e., each σ_i is associated with a distinct type-0 leaf in a binary code tree. Since we are interested in minimum-redundancy codes we assume that the codeword lengths satisfy $l_1 \leq l_2 \leq \cdots \leq l_n$. To simplify the study of optimal codes we say that a binary code tree T is *feasible* if

1) T is full.
2) Suppose T has $m < n$ type-0 leaves with depth less than d and b_j type-j nodes with depth d, $j \in \{0,\ 1,\ \ldots,\ w-1\}$. Then $b_j \leq n-m$, $j \in \{0,\ 1,\ \ldots,\ w-1\}$. Furthermore, $\sum_j b_j \leq 2(n-m)$ and is even.
3) Suppose T has $m < n$ type-0 leaves with depth less than d and b_j type-j nodes with depth d, $j \in \{0,\ w-1\}$. Then exactly $\min\{b_{w-1},\ n-m-b_0\}$ of the type-$(w-1)$ nodes of depth d are intermediate nodes.
4) If node $u \in T$ is a type-j intermediate node, then every node $v \in T$ with $depth(v) = depth(u)$ and $type(v) > j$ is also an intermediate node of T.
5) Suppose T has $m < n$ type-0 leaves with depth less than d and no type-0 nodes with depth d. (This can only apply to full trees if $w > 2$.) Let $\mathcal{T} \subseteq \{1, \ldots, w-1\}$ be the types of the nodes in T at depth d, and let t be the maximum element of $\mathcal{T}$. If $t < w-1$, then there is at least one node in T of type $t+1$ at depth $d+1$. If $t = w-1$, then there are $\min\{b_{w-1},\ n-m\}$ type-0 nodes in T at depth $d+1$.

Lemma 2: For any source there is an optimal binary code tree T which is feasible.

Proof: (Sketch) For Property 1, given an optimal binary code tree which is not full one can construct a second and larger optimal binary code tree from the first by inserting a sibling node and the corresponding edge for every node that did not have a sibling in the original tree. Since each of the n source symbols is mapped to a different leaf in the binary code tree, any binary code tree must have at least n type-0 leaves, and a smallest optimal full binary code tree must have more than n leaves. These extra leaves, which are not of type 0, do not correspond to codewords.

Next consider Property 2. Observe that if a smallest optimal full tree T has m type-0 leaves with depth less than d, then it has $n-m$ type-0 leaves with depth at least d. Each one of this latter set of leaves is on some path originating at a depth-d node of T. If a pair of depth-d sibling nodes u and v of T have the property that none of the leaves of the subtrees rooted at u and v correspond to codewords, then there is a smaller optimal full tree in which these subtrees and u and v are removed from T and the parent of u and v, which has depth $d-1$, is converted into a leaf node. The number of nodes at any depth $d \geq 1$ is even because the tree is full.

For Property 3, we once again have that the smallest optimal full tree T has $n-m$ type-0 leaves with depth at least d. If $m + b_0 + b_{w-1} \geq n$, then T has b_0 type-0 leaves at depth d and $n-m-b_0$ type-0 leaves at depth $d+1$ having a type-$(w-1)$ parent. Otherwise, suppose $m + b_0 + b_{w-1} < n$ and there is a type-$(w-1)$ node u at depth d which is a leaf. If node u were to be converted to an intermediate node, suppose that u_0 would be its type-0 child. By our assumptions there is at least one type-0 leaf at a depth greater than or equal to $d + 2$. If the maximum-depth type-0 leaf and its sibling were removed from T and replaced by u_0 and its sibling, the resulting binary code would improve.

For Property 4, let $u \in T$ be a type-j intermediate node and let u_j and u_{j+1} respectively denote the type-j and type-$(j+1)$ children of u. Suppose that there is a type-k leaf $v \in T$ with $depth(v) = depth(u)$ and $k > j$. If none of the leaves of the subtree of T rooted at u_{j+1} correspond to codewords, then there are three possibilities: (1) if u_{j+1} is an intermediate node, then its descendants can be removed from T and u_{j+1} can be converted into a leaf without changing the average codeword length of the resulting binary code tree or (2) if u_j and u_{j+1} are both leaves, then they can be removed from T and u can be converted into a leaf node without increasing the average codeword length of the resulting binary code tree or (3) if u_j is an intermediate node and u_{j+1} is a leaf, then the existing edges between u and u_j and u_{j+1} can be removed from T and the subtree of T rooted at u_j can be moved to be rooted at u without increasing the average codeword length of the resulting binary code tree. Therefore assume that at least one codeword corresponds to a leaf from the subtree rooted at u_{j+1}. Such a leaf is on a path originating at a type-k node ν with depth at least $depth(u) + 1$. Then a better average codeword length can be obtained by moving the subtree of T rooted at ν to v.

Finally consider Property 5. First suppose $t < w - 1$ and suppose that there is no type-$(t + 1)$ node in T at depth $d + 1$. Let v be a type-t leaf at depth d. Since T contains at least n type-0 leaves there are at least $n - m$ type-0 leaves at depth greater than d. Each one of these must be on some path of T originating at a type-$(t + 1)$ node at some depth greater than or equal to $d + 2$. Let ν_{t+1} be such a type-$(t+1)$ node. If v were converted into an intermediate node, then it would have a type-$(t+1)$ child at depth $d + 1$, say v_{t+1}. If we move the subtree of T rooted at ν_{t+1} to v_{t+1}, then the resulting tree would have a smaller average codeword length than T. Next suppose $t = w - 1$. As we observed in the discussion of Property 3, if $m + b_{w-1} \leq n$ then each type-$(w - 1)$ node at depth d is an intermediate node, and hence the type-0 children of these nodes are the b_{w-1} type-0 nodes of T at depth $d + 1$. $\Box$

For a smallest optimal feasible binary code tree T, let v_j, $j \in \mathcal{X}$, be the leaf associated with the binary codeword σ_j. Then the expected length, $E[L]$, of the code can be expressed as

$$E[L] \;=\; \sum_{j=1}^{n} l_j p_j \;=\; \sum_{j=1}^{n} depth(v_j) \cdot p_j.$$

The dynamic programming technique of [5] constructs binary code trees from the root based on properties of certain subtrees of the smallest optimal feasible binary code tree. Following [5], for any binary code tree T and non-negative integer i, let $Trunc_i(T)$ represent the subtree of T consisting of the nodes with depth at most $i + 1$ and the corresponding edges. $Trunc_i(T)$ is called the *i-level truncation* of T. It is not difficult to verify that if T is feasible then so is $Trunc_i(T)$ for each i. Tree T is said to be an *i-level tree* if every intermediate node $u \in T$ satisfies $depth(u) \leq i$. We next define a "signature" and a cost function on i-level trees which are adapted from [5].

If T is a feasible i-level tree, we define its *i-level signature* as an ordered $(w + 1)$-tuple

$$\begin{aligned} sig_i(T) \;&=\; (m,\ b_0,\ b_1,\ \ldots,\ b_{w-1}) \\ \text{with } m \;&=\; |\{v \in T : v \ \text{ is a type-0 leaf}, depth(v) \leq i\}| \\ \text{and } b_t \;&=\; |\{v \in T : v \ \text{ is a type-}t\text{ leaf}, depth(v) = i + 1\}|,\ t \in \{0,\ \ldots,\ w - 1\}. \end{aligned}$$

If T is a feasible i-level tree with $sig_i(T) = (m,\ b_0,\ b_1,\ \ldots,\ b_{w-1})$ and $m \leq n$, then its *cost function* $\mathcal{L}_i(T)$ is given by

$$\mathcal{L}_i(T) \;=\; \sum_{j=1}^{m} depth(v_j) \cdot p_j + i \cdot \sum_{j=m+1}^{n} p_j,$$

where $v_1,\ v_2,\ \ldots,\ v_m$ represent the type-0 leaves of T with depth at most i, and as usual $depth(v_1) \leq \cdots \leq depth(v_m)$.

The dynamic programming algorithm of [5] is based on the premise that an optimal binary code tree T has "minimum cost" i-level truncations $Trunc_i(T)$ for every i. To modify this for our problem, for signature $(m,\ b_0,\ b_1,\ \ldots,\ b_{w-1})$ let
$OPT[m,\ b_0,\ b_1,\ \ldots,\ b_{w-1}] =$

$$\min_i\{\mathcal{L}_i(T) :\ \text{there exists a feasible } i\text{-level tree } T \text{ with signature } (m,\ b_0,\ b_1,\ \ldots,\ b_{w-1})\}.$$

Tree T is said to have *minimum cost* if T is a feasible i-level tree for some i and T satisfies $sig_i(T) = (m,\ b_0,\ b_1,\ \ldots,\ b_{w-1})$ and $\mathcal{L}_i(T) = OPT[m,\ b_0,\ b_1,\ \ldots,\ b_{w-1}]$. Observe that if T is a smallest optimal feasible binary code tree with n type-0 leaves each with depth at most i, then $sig_i(T) = (n, 0, \ldots, 0)$ and $OPT[n, 0, \ldots, 0] = \mathcal{L}_i(T) = E[L] = \sum_{j=1}^{n} depth(v_j) \cdot p_j$. Our strategy will be to use dynamic programming to obtain the table $OPT[m,\ b_0,\ b_1,\ \ldots,\ b_{w-1}]$ and to backtrack from the solution for $OPT[n, 0, \ldots, 0]$ to deduce the corresponding optimal feasible binary code tree T.

Suppose that T is a feasible i-level tree with signature $(m,\ b_0,\ b_1,\ \ldots,\ b_{w-1})$. Let us consider the possible feasible $(i+1)$-level trees with T as their i-level truncation. In each case, a (possibly empty) subset of the $b_0 + b_1 + \cdots + b_{w-1}$ nodes of T with depth $i+1$ remain as leaves in the $(i+1)$-level tree and the rest become intermediate nodes each having two children which are leaves of depth $i+2$. To be more precise, let integer vector $\mathbf{q} = (q_0,\ q_1,\ \ldots,\ q_{w-1})$ satisfy $0 \leq q_j \leq b_j$ for each j. The *expansion* $\mathbf{q}$ of T, denoted $Expand(T, \mathbf{q})$, is constructed by converting q_j of the type-j leaves at depth $i+1,\ j \in \{0,\ 1,\ \ldots,\ w-1\}$, to intermediate nodes with two children which are each leaves at depth $i+2$. To satisfy feasibility conditions we also require:

1) $q_{w-1} = \min\{b_{w-1},\ n - m - b_0\}$.
2) If $q_j > 0$ for $j \in \{0,\ 1,\ \ldots,\ w-2\}$, then $q_k = b_k$ for all $k > j$.
3) If $q_0 = b_0 = 0$, then let t be the maximum index for which $b_t > 0$. If $t < w-1$, then $q_t \geq 1$.

We have

Lemma 3: Given a feasible i-level tree T and an expansion vector $\mathbf{q}$ of T, let $\hat{T} = Expand(T, \mathbf{q})$. Then $\mathcal{L}_{i+1}(\hat{T}) = \mathcal{L}_i(T) + \sum_{j=m+1}^{n} p_j$, and

$$sig_{i+1}(\hat{T}) = (m + b_0 - q_0,\ q_0 + q_{w-1},\ q_0 + q_1,\ q_1 + q_2, \ldots,\ q_{w-2} + q_{w-1}).$$

The first part of the proof of Lemma 3 is similar to the proof of [5, Lemma 2] and we omit the details here. The second part follows from the fact that if a binary string has Hamming weight w, then its single bit extensions have Hamming weight w and $w+1$. Notice that when $w = 2$ it is sufficient to describe the i-level signature of a feasible i-level tree T by its first two entries since the number of type-0 nodes and type-1 nodes of depth $i+1$ are equal.

The significance of Lemma 3 is that $sig_i(T)$ captures all of the information about i and T that are needed to calculate the increase in cost function incurred by an expansion of the i-level tree T and the signature of this expanded tree. Therefore, the OPT table can be computed recursively. Following [5], let $\mathcal{M}(\hat{m},\ \hat{b}_0,\ \hat{b}_1,\ \ldots,\ \hat{b}_{w-1})$ denote the set of signatures that have an expansion with signature $(\hat{m},\ \hat{b}_0,\ \hat{b}_1,\ \ldots,\ \hat{b}_{w-1})$.

Lemma 4: The table of OPT values can be calculated as follows: $OPT[0, 1, 1, 0, 0, \ldots, 0] = 0$. For $(\hat{m},\ \hat{b}_0,\ \hat{b}_1,\ \ldots,\ \hat{b}_{w-1}) \neq (0, 1, 1, 0, 0, \ldots, 0)$,

$$OPT[\hat{m},\ \hat{b}_0,\ \hat{b}_1,\ \dots,\ \hat{b}_{w-1}] =$$

$$\min_{(m,\ b_0,\ b_1,\ \dots,\ b_{w-1}) \in \mathcal{M}(\hat{m},\ \hat{b}_0,\ \hat{b}_1,\ \dots,\ \hat{b}_{w-1})} \left\{ OPT[m,\ b_0,\ b_1,\ \dots,\ b_{w-1}] + \sum_{j=m+1}^{n} p_j \right\}.$$

The proof of Lemma 4 makes use of the existence of an optimal feasible binary code tree, and it is nearly identical to the proof of [5, Lemma 3]. We omit most of the details here. We mention, however, that the unique tree with signature $(0, 1, 1, 0, 0, \dots, 0)$ is the 0-level tree consisting of the root and its two children, which is easily seen to have zero cost.

We next define an ordering on signatures. Let $(m,\ b_0,\ b_1,\ \dots,\ b_{w-1})$ and $(\hat{m},\ \hat{b}_0,\ \hat{b}_1,\ \dots,\ \hat{b}_{w-1})$ be given. We say that $(m,\ b_0,\ b_1,\ \dots,\ b_{w-1})$ *precedes* $(\hat{m},\ \hat{b}_0,\ \hat{b}_1,\ \dots,\ \hat{b}_{w-1})$, written

$$(m,\ b_0,\ b_1,\ \dots,\ b_{w-1}) \prec (\hat{m},\ \hat{b}_0,\ \hat{b}_1,\ \dots,\ \hat{b}_{w-1}),$$

if and only if

$$\begin{aligned} \{m < \hat{m}\} \quad \text{or} \quad & \{m = \hat{m},\ b_0 < \hat{b}_0\} \text{ or } \{m = \hat{m},\ b_0 = \hat{b}_0 > 0,\ b_1 < \hat{b}_1\} \\ \text{or} \quad & \{m = \hat{m},\ b_0 = \hat{b}_0 = 0,\ \max_{t: b_t > 0} t < \max_{\hat{t}: \hat{b}_{\hat{t}} > 0} \hat{t}\} \end{aligned}$$

We have

Lemma 5: If $(m,\ b_0,\ b_1,\ \dots,\ b_{w-1})$ and $(\hat{m},\ \hat{b}_0,\ \hat{b}_1,\ \dots,\ \hat{b}_{w-1})$ are signatures for which $(m, b_0, b_1, \dots, b_{w-1}) \in \mathcal{M}(\hat{m}, \hat{b}_0, \hat{b}_1, \dots, \hat{b}_{w-1})$, then $(m, b_0, b_1, \dots, b_{w-1}) \prec (\hat{m}, \hat{b}_0, \hat{b}_1, \dots, \hat{b}_{w-1})$.

The preceding results imply that the algorithm below correctly tabulates the OPT values and leads to the construction of an optimal binary code tree. The algorithm proceeds so that all signatures in $\mathcal{M}(\hat{m},\ \hat{b}_0,\ \hat{b}_1,\ \dots,\ \hat{b}_{w-1})$ are processed before $(\hat{m},\ \hat{b}_0,\ \hat{b}_1,\ \dots,\ \hat{b}_{w-1})$. By an inductive argument similar to the one in [5], the OPT value of any signature is correctly set before it is used to help determine the OPT value of one of its expansions.

The Algorithm

```
**Initialize the $OPT$ table**
For all $m, b_0, b_1, \dots, b_{w-1}$ with $0 \le m \le n,\ 0 \le b_i \le n-m,\ \sum_j b_j \le 2(n-m)$ and even
  Set $OPT[m,\ b_0,\ b_1,\ \dots,\ b_{w-1}] = \infty$;
For all $m$ such that $0 \le m \le n$, Set $S_{m+1} = \sum_{j=m+1}^{n} p_j$;
$OPT[0, 1, 1, 0, 0, \dots, 0] := 0$;
**Calculate the $OPT$ values**
for $m := 0$ to $n$
  $b_0 := 0$;
    for $t := 2$ to $w-1$
      for $b_1 := 0$ to $n-m$
        for $b_2 := 0$ to $\min\{n-m,\ 2(n-m) - 2 - b_1\}$
          ⋮
          for $b_{t-2} := 0$ to $\min\{n-m,\ 2(n-m) - 2 - b_1 - b_2 - \cdots - b_{t-3}\}$
            for $b_{t-1} := 1$ to $\min\{n-m,\ 2(n-m) - 1 - b_1 - b_2 - \cdots - b_{t-2}\}$
              for $b_t := 1$ to $\min\{n-m,\ 2(n-m) - b_1 - b_2 - \cdots - b_{t-1}\}$
                with $\sum_j b_j$ even
                **Process the signature** $(m,\ b_0,\ \dots,\ b_{w-1})$
              if $t < w-1$
                {for $j := 1$ to $b_t - 1$
                  {$q_t := j$;
                  for $k := 0$ to $t-1$
                    $q_k := 0$;
                  for $k := t+1$ to $w-1$
                    {$b_k := 0$; $q_k := 0$;}
                  $X := \min(OPT[m,\ b_0,\ b_1,\ \ldots,\ b_{w-1}] + S_{m+1},$
                    $OPT[m,\ q_0 + q_{w-1},\ q_0 + q_1,\ \ldots,\ q_{w-2} + q_{w-1}])$
                  if $X < OPT[m,\ q_0 + q_{w-1},\ q_0 + q_1,\ \ldots,\ q_{w-2} + q_{w-1}]$
                    {$OPT[m,\ q_0 + q_{w-1},\ q_0 + q_1,\ \ldots,\ q_{w-2} + q_{w-1}] := X$;
                    $Q[m,\ q_0 + q_{w-1},\ q_0 + q_1,\ \ldots,\ q_{w-2} + q_{w-1}] := (m,\ b_0,\ \ldots,\ b_{w-1})$;}}
              for $i := t-1$ down to 1
                for $j := 0$ to $b_i$
                  {$q_i := j$;
                  for $k := 0$ to $i-1$
                    $q_k := 0$;
                  for $k := i+1$ to $t$
                    $q_k := b_k$;
                  $X := \min(OPT[m,\ b_0,\ b_1,\ \ldots,\ b_{w-1}] + S_{m+1},$
                    $OPT[m,\ q_0 + q_{w-1},\ q_0 + q_1,\ \ldots,\ q_{w-2} + q_{w-1}])$
                  if $X < OPT[m,\ q_0 + q_{w-1},\ q_0 + q_1,\ \ldots,\ q_{w-2} + q_{w-1}]$
                    {$OPT[m,\ q_0 + q_{w-1},\ q_0 + q_1,\ \ldots,\ q_{w-2} + q_{w-1}] := X$;
                    $Q[m,\ q_0 + q_{w-1},\ q_0 + q_1,\ \ldots,\ q_{w-2} + q_{w-1}] := (m,\ b_0,\ \ldots,\ b_{w-1})$;}}
for $b_0 := 1$ to $n-m$
  for $b_1 := 0$ to $n-m$
    for $b_2 := 0$ to $\min\{n-m,\ 2(n-m) - b_0 - b_1\}$
      $\vdots$
      for $b_{w-1} := 0$ to $\min\{n-m,\ 2(n-m) - b_0 - b_1 - b_2 - \cdots - b_{w-2}\}$
        with $\sum_j b_j$ even
        **Process the signature** $(m,\ b_0,\ \ldots,\ b_{w-1})$
        $q_{w-1} := \min\{b_{w-1},\ n-m-b_0\}$;
        if $q_{w-1} < b_{w-1}$
          {for $k := 0$ to $w-2$
            $q_k := 0$;
          $X := \min(OPT[m,\ b_0,\ b_1,\ \ldots,\ b_{w-1}] + S_{m+1},$
            $OPT[m + b_0 - q_0,\ q_0 + q_{w-1},\ q_0 + q_1,\ \ldots,\ q_{w-2} + q_{w-1}])$
          if $X < OPT[m + b_0 - q_0,\ q_0 + q_{w-1},\ q_0 + q_1,\ \ldots,\ q_{w-2} + q_{w-1}]$
            {$OPT[m + b_0 - q_0,\ q_0 + q_{w-1},\ q_0 + q_1,\ \ldots,\ q_{w-2} + q_{w-1}] := X$;
            $Q[m + b_0 - q_0,\ q_0 + q_{w-1},\ q_0 + q_1,\ \ldots,\ q_{w-2} + q_{w-1}] := (m,\ b_0,\ \ldots,\ b_{w-1})$;}
          }
        else
          {for $i := w-2$ down to 0
            for $j := 0$ to $b_i$
              $q_i := j$;
            if $i > 0$
               {for $k := 0$ to $i - 1$
                  $q_k := 0;$}
            for $k := i + 1$ to $w - 2$
               $q_k := b_k;$
         $X := \min(OPT[m,\ b_0,\ b_1,\ \ldots,\ b_{w-1}] + S_{m+1},$
            $OPT[m + b_0 - q_0,\ q_0 + q_{w-1},\ q_0 + q_1,\ \ldots,\ q_{w-2} + q_{w-1}])$
         if $X < OPT[m + b_0 - q_0,\ q_0 + q_{w-1},\ q_0 + q_1,\ \ldots,\ q_{w-2} + q_{w-1}]$
            $\{OPT[m + b_0 - q_0,\ q_0 + q_{w-1},\ q_0 + q_1,\ \ldots,\ q_{w-2} + q_{w-1}] := X;$
            $Q[m + b_0 - q_0,\ q_0 + q_{w-1},\ q_0 + q_1,\ \ldots,\ q_{w-2} + q_{w-1}] := (m,\ b_0,\ \ldots,\ b_{w-1});\}$
      }
```

Backtracking

```
m:=n;
for $j := 0$ to $w - 1$
   $b_j := 0$
repeat $\{(m,\ b_0,\ b_1,\ \ldots,\ b_{w-1}) = Q[m,\ b_0,\ b_1,\ \ldots,\ b_{w-1}];$ print $Q;\}$
   until $(m,\ b_0,\ b_1,\ \ldots,\ b_{w-1}) = (0, 1, 1, 0, 0, \ldots, 0)$
```

Observe that there are $O(n^{w+1})$ signatures processed by the algorithm. Recall that for any expansion $\mathbf{q}$ of a signature with the property $q_j > 0$ for some $j \in \{0,\ 1,\ 2,\ \ldots,\ w-1\}$, one of the feasibility conditions implies that $q_k = b_k$ for all $k > j$. Therefore, for signature $(m,\ b_0,\ \ldots,\ b_{w-1})$ there are at most $\sum_i (b_i + 1) = O(n)$ expansion vectors to handle, and this leads to an $O(n^{w+2})$ running time for the algorithm. Although the algorithm applies for any $w \geq 2$, for $w = 2$ it is preferable to use the slightly simpler algorithm on [5, p. 1642], which takes advantage of the constraint $b_0 = b_1$ to reduce the complexity to $O(n^3)$.

To illustrate backtracking, consider an example with $n = 11$, $p_1 = p_2 = p_3 = 0.25$, $p_4 = p_5 = \cdots = p_{11} = 0.03125$ and $w = 3$. Here $H(X) = 2.75$ bits per symbol, and the code obtained by this procedure uses 3.34375 bits per symbol. The sequence of signatures output by the backtracking step are $[11, 0, 0, 0]$, $[7, 4, 0, 4]$, $[4, 3, 1, 4]$, $[2, 2, 1, 3]$, $[1, 1, 1, 2]$, $[1, 0, 1, 1]$, $[0, 1, 1, 0]$. $[0, 1, 1, 0]$ corresponds to the initial set of words 0 and 1. 0 is selected as a codeword and 1 is replaced by its single letter extensions 10 and 11. Next these two-bit words are each replaced with their single letter extensions 100, 101, 110, 111. 111 is then chosen as the second codeword, and the remaining three three-bit words are replaced by 1000, 1001, 1010, 1011, 1100, 1101. 1011 and 1101 are the next two codewords, and the other four are replaced by 10000, 10001, 10010, 10011, 10100, 10101, 11000, 11001. The next three codewords are 10011, 10101, 11001, and the final four are 100011, 100101, 101001, 110001.

3. Numerical results

In the following we present some numerical results and comparisons. The performance for both the English alphabet (see, e.g., [3]) and typical discrete source distributions is given in Table 1, where Laplace, Gaussian, and uniform distributions are obtained by a 6-bit uniform scalar quantization of the corresponding continuous random variables. We can see that the optimal construction with $w = 2$ leads to the shortest expected word-length among all codes with a minimum Hamming distance of $d_{\text{min}} = 2$, in particular the algorithm EWVLC proposed in [3] and the variable-length error correction (VLEC) construction from [13].

For the application of joint source-channel coding the prefix code is used as an outer source code which is serially concatenated with an inner channel code, where decoding is carried out iteratively

Table 1. Expected word-lengths $E[L]$ of different prefix code for both the English alphabet and discrete source distributions derived from typical continuous random variables by 6-bit uniform quantization. All constructions except the Huffman code have $d_{\text{min}} = 2$.

	$E[L]$ (bits)			
	Engl. alphabet	Laplace	Gaussian	Uniform
Huffman	4.1557	3.3328	4.492	6
Proposed opt., $w = 2$	**4.2263**	**3.3473**	**4.5846**	**6.5**
EWVLC [3]	4.2366	3.3867	4.6227	6.5
VLEC $d_{\text{min}} = 2$ [13]	4.2366	3.4474	4.6227	7.875
$H(X)$	4.1209	3.2963	4.47	6

analogous to the decoding of serially concatenated codes (see, e.g., [3], [14]). It can be shown that in order to ensure convergence for the iterative decoder, $d_{\text{min}} \geq 2$ must hold in general for the outer code [14], and therefore must be imposed on the prefix code in this setup. Further, it has been shown in [3] that thresholds close to capacity can be obtained for such a concatenated source-channel code if, in addition, residual source redundancy is exploited at the decoder. If a first-order Markov process I_k is assumed for the source, the overall code rate for long source blocks is given as

$$R = \frac{H(I_k|I_{k-1})}{H(I_k)} \cdot \frac{H(I_k)}{E[L]} \cdot R_{\text{Inner}},$$

where R_{Inner} is the inner code rate, $\frac{H(I_k|I_{k-1})}{H(I_k)}$ the rate contribution due to the Markov property of the source process, and $\frac{H(I_k)}{E[L]}$ the rate associated with the prefix code.

Fig. 1 shows simulation results for different prefix codes, where the transmission is carried out over an AWGN channel with a block length of 1000 source symbols. The source process I_k is obtained by filtering white Gaussian noise with a first-order recursive filter having a correlation coefficient of $a = 0.9$ and subsequent quantization with a 4-bit uniform quantizer. The channel code is a $(15, 14)_8$ recursive systematic convolutional code which is obtained by an EXIT chart convergence analysis to approximately match both the (very similar) EXIT functions for the optimal prefix code for $w = 2$ and the EWVLC from [3]. The channel code is punctured to $R_{\text{Inner}} = 1$ by keeping only the parity sequence, where every tenth parity bit is replaced by a systematic bit. The symbol error rate (SER) is measured with the Levenshtein distance [15], which is defined as the minimal number of insertions, deletions or substitutions that transform one sequence into the other one and thus accounts for the self-synchronizing property of prefix codes. We can see from Fig. 1 that all schemes suffer from an error floor, but the one which employs the optimal prefix code has a lower error floor than those obtained by using the EWVLC [3] or the VLEC [13]. The code rate for the latter two schemes is $R \approx 0.6$, whereas the scheme based on the optimal prefix construction has a slightly higher rate of $R \approx 0.63$ due to the better source compression capabilities.

Acknowledgment. The authors thank R. Thobaben for providing them with simulation programs he wrote in the past while collaborating with the second author.

References

[1] V. Buttigieg and P. G. Farrell, "Constructions for variable-length error-correcting codes," in C. Boyd, Ed., *Cryptography and Coding*, pp. 282-291, Berlin: Springer, 1995.

[2] V. B. Balakirsky, "Joint source-channel coding with variable length codes," *Proc. IEEE Int. Symp. Inform. Thy.*, p. 419, Ulm, Germany, June 1997.

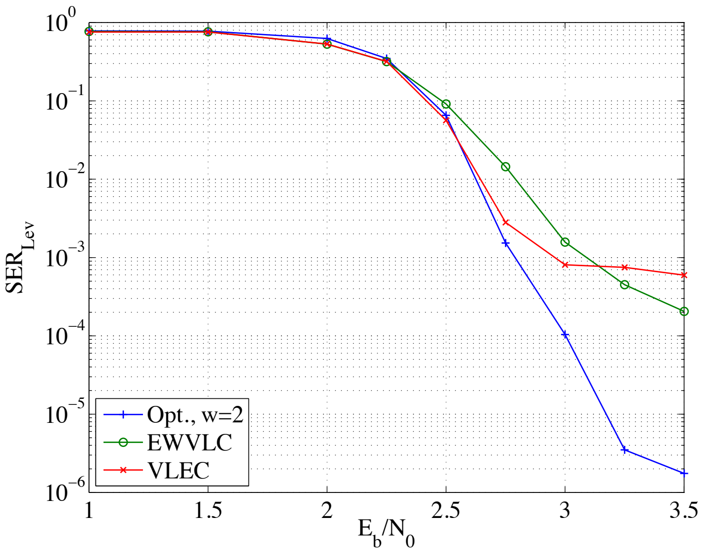

Figure 1. SER based on the Levenshtein distance versus E_b/N_0 for different prefix codes as source codes in a joint source-channel coding setup. The input process is a Gauss-Markov process with correlation coefficient of $a = 0.9$, and the block length is chosen as 1000 4-bit symbols.

[3] R. Thobaben and J. Kliewer, "An efficient variable-length code construction for iterative source-channel decoding," *IEEE Trans. Comm.*, vol. 57, pp. 2005-2013, July 2009.

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

[5] S.-L. Chan and M. J. Golin, "A dynamic programming approach for constructing optimal "1"-ended binary prefix-free codes," *IEEE Trans. Inform. Theory*, vol. 46, pp. 1637–1644, July 2000.

[6] M. J. Golin, X. Xu, and J. Yu, "A generic top-down dynamic-programming approach to prefix-free coding," arXiv:0809.4577 [cs.DS], Sept. 26, 2008.

[7] M. J. Golin, X. Xu, and J. Yu, "A generic top-down dynamic-programming approach to prefix-free coding," in *Proc. ACM-SIAM Symp. Disc. Alg. (SODA'09)*, Jan. 2009.

[8] 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.

[9] R. G. Gallager, "Variations on a theme by Huffman," *IEEE Trans. Inform. Theory*, vol. IT-24, pp. 668-674, Nov. 1978.

[10] M. Sipser, *Introduction to the Theory of Computing*, Second Ed., Kentucky: Course Technology, 2005.

[11] M. J. Golin and H. Na, "Optimal prefix-free codes that end in a specified pattern and similar problems: the uniform probability case (Extended Abstract)," in *Proc. 2001 IEEE Data Compression Conference (DCC 2001)*, pp. 143–152, March 2001.

[12] M. J. Golin and Z. Liu, "The structure of optimal prefix-free codes in restricted languages: the uniform probability case (Extended Abstract)," in *Proc. 2005 Workshop on Algorithms and Data Structures (WADS '05)*, pp. 372–384, LNCS 3608, Aug. 2005.

[13] J. Wang, L.-L. Yang, and L. Hanzo, "Iterative construction of reversible variable-length codes and variable-length error-correcting codes," *IEEE Communications Letters*, vol. 50, pp. 671-673, Nov. 2004.

[14] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "Serial concatenation of interleaved codes: Performance analysis, design, and iterative decoding," *IEEE Trans. Inform. Theory*, vol. 44, pp. 909-926, May 1998.

[15] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," *Sov. Phys.-Dokl.*, vol. 10, pp. 707-710, Feb. 1966.