Coded Modulation for the Deep-Space Optical Channel: Serially Concatenated Pulse-Position Modulation

B. Moision[1] and J. Hamkins[1]

We present an error-control coding technique for optical communications. It consists of the serial concatenation of an outer convolutional code, an interleaver, a bit-accumulator, and pulse-position modulation (PPM). We refer to the coded modulation as serially concatenated PPM, or SCPPM. The encoding is accomplished with simple shift register operations and a table look-up to map code bits to PPM symbols. The code is decoded with an iterative demodulator–decoder, using standard turbo-decoding techniques. For system constraints typical of the Mars Laser Communications Demonstration, simulations indicate operation within 1 dB of capacity. We show that the standard decoder can be simplified by precomputing certain edge likelihoods on a reduced-edge trellis, without approximation or degradation, and that an M-input $\max^*$ function may be distributed and pipelined. A further simplification allows one to discard many of the channel observables, with negligible degradation.

I. Introduction

This article presents an efficient error-control code (ECC) and modulation for an optical communications channel. The coded modulation is designed to support NASA's Mars Laser Communications Demonstration(MLCD)—an optical communications link from Mars to Earth that will operate at data rates of up to 49 mega-bits per second (Mbps). The MLCD laser terminal will be carried aboard the Mars Telecommunications Orbiter and is scheduled to launch in 2009.

For our purposes, the optical communications channel may be reduced to the block diagram in Fig. 1. User information $\mathbf{u}$ is encoded, modulated, and transmitted over an optical channel that is modeled as a Poisson point process. At the receiver, the signal is demodulated and decoded (shown as an iterative procedure), yielding estimates of the user data. Wyner [1] showed that under peak and average power constraints, negligible capacity is lost when restricting the modulation to a binary, slotted scheme. Furthermore, for peak and average power constraints typical of a deep-space link, restricting the modulation to pulse-position modulation (PPM) [2] is near-capacity achieving as well [3]. The efficiency of PPM for an optical channel has been noted elsewhere, e.g., [4].

[1] Communications Architectures and Research Section.

The research described in this publication was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

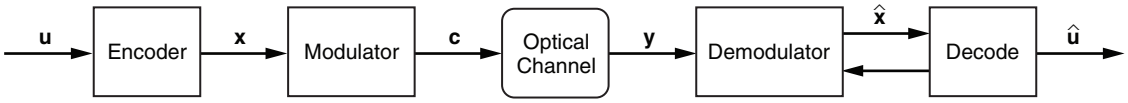

Fig. 1. A coded optical channel.

Hence, we constrain the modulation to be PPM and address the choice of a suitable ECC. Several ECCs have been considered for the Poisson PPM channel in the past. In [5], a Reed-Solomon (RS) code was proposed for error protection on the *noiseless* Poisson PPM channel. The noiseless Poisson PPM channel is a symbol erasure channel, and an (n, k) RS code may be tailored to fit an M-ary PPM channel by choosing $n = M - 1$ and taking code symbols from the Galois field with M elements. This code is optimum for the given block length n on the noiseless channel in the sense that it can correct up to $n - k + 1$ symbol erasures, which is the maximum for a linear code with block length n.

However, a block length of $n = M - 1$ is, for cases of practical interest, too small to achieve good performance. Longer block lengths may be obtained with RS codes defined in higher-order fields, in which multiple PPM symbols are associated with each code symbol, but this results in only marginal improvement [6]. RS performance on a noisy Poisson channel typically remains 3 dB or more away from capacity when conventional hard-decision decoding is used [6]. Recent improvements in RS decoding [7,8] will reduce the gap, but their use is not explored here.

A number of coding techniques involving convolutional codes have also been proposed. Massey [9] illustrated that the M-ary erasure channel is equivalent to the parallel combination of $\log_2 M$ binary erasure channels, and proposed coding each separately using a convolutional code. He demonstrated more robust performance than RS-coded PPM. Alternatively, M-ary convolutional codes can be used directly with M-ary PPM [10]. A joint decoding and demodulation of this coded modulation requires a trellis with nodes of degree M.

In this article, we propose a variant on convolutionally coded PPM, motivated by recent results in serially concatenated, iteratively decoded codes [11]. In the proposed scheme, a large block of user bits is first encoded by a short-constraint-length convolutional code. The output is then bit interleaved, passed through a rate-1 recursive accumulator, and mapped to PPM symbols. The encoding and modulation differ from prior approaches only in the introduction of a bit interleaver and accumulator. It is the decoding algorithm that is significantly different.

Conventionally, the modulation and ECC are decoded sequentially, with the demodulator sending its results to the ECC decoder. However, we may consider the combination of the modulation and the ECC as a single large code, which maps user information bits directly to the symbols transmitted on the channel and use an iterative demodulator–decoder to decode this large code. Once the underlying trellis descriptions of the outer convolutional code and inner accumulator and modulation are formulated, standard forward–backward algorithms [12,13] can be used on them. Numerical results show that this is a powerful technique to obtain near-capacity performance, sometimes several decibels better than previously proposed coding techniques.

Prior work on iterative decoders considered the application of turbo codes applied to the Poisson PPM channel. A turbo code conventionally consists of a pair of convolutional codes concatenated in parallel through a bit interleaver and decoded iteratively. In [14] a turbo code was applied to the binary PPM channel, and in [15] a turbo code was applied to M-ary PPM. In the latter article, the PPM demodulator passes soft information to the turbo code but is not involved in iterative decoding. This approach allowed performance improvements of more than 0.5 dB over RS-coded PPM. In [16], a turbo code is decoded iteratively with PPM demodulation on the discrete-time Rayleigh fading channel, illustrating performance 1–2 dB from capacity. By allowing iterative demodulation and choosing a simple short-constraint-length

outer convolutional code, our proposed approach is able to achieve both better performance and lower complexity than the turbo-coded PPM approaches.

The remainder of this article is organized as follows. In Section II, we describe the SCPPM encoder and trellis description. In Section III, we describe the decoding algorithm and some simplifications of the algorithm. In Section IV, we describe statistics generated by a Poisson channel. In Sections V through VII, we describe particular choices of interleaver, stopping rule, and bit-to-symbol mapping, illustrating performance results in Section VIII. In Section IX, we look at the performance of some lower-complexity decoding algorithms.

Lowercase u, w, y, x denote realizations of the corresponding random variables U, W, Y, X. Boldface $\mathbf{u} = (u_1, u_2, \cdots, u_N)$ and $\mathbf{U} = (U_1, \cdots, U_N)$ denote vectors. We also write $\mathbf{u} = (\mathbf{u}_1, \cdots, \mathbf{u}_N)$, where each $\mathbf{u}_k = (u_{k,1}, \cdots, u_{k,N})$ is a vector. The subscript $\mathbf{u}_{i:j} \triangleq (\mathbf{u}_i, \mathbf{u}_{i+1}, \cdots, \mathbf{u}_j)$ is used to denote a subsequence; $\mathbf{w}_{[k,i]}$ denotes the sequence $\mathbf{w}$ with element $\mathbf{w}_{k,i}$ removed. The notation $p_Y(y)$ is used to denote the probability density or mass function of random variable Y evaluated at y. When the random variable is clear from the context, we simply write $p(y)$ for $p_Y(y)$.

II. SCPPM Encoder

The serially concatenated pulse-position modulation (SCPPM) encoder is illustrated in Fig. 2. In performance results, we assume a short cyclic-redundancy-check (CRC) pattern is added to each block of user bits prior to SCPPM encoding. The CRC is used for block error detection and for terminating decoding but is not considered part of the SCPPM code. Its use is discussed in Section VI.

The SCPPM encoder consists of two constituent codes, referred to as the outer and inner codes, and a bit interleaver. The outer code is a short-constraint-length convolutional code. The inner code is composed of an accumulator (a $1/(1+D)$ filter) and a memoryless PPM modulator and referred to as an APPM code. The modulator maps each set of $\log_2(M)$ bits at the output of the accumulator to a PPM symbol. A practical and robust interleaver structure is discussed in Section V, and a bit-to-symbol mapping that provides robustness on a channel with memory is discussed in Section VII.

A block of information bits $\mathbf{u}$ is encoded by the outer code to yield a coded sequence $\mathbf{x}$. The sequence $\mathbf{x}$ is interleaved, bit-wise, to produce the sequence $\mathbf{a} = (\mathbf{a}_1, \mathbf{a}_2, \cdots, \mathbf{a}_N)$, which is encoded by the APPM code to yield the sequence $\mathbf{c} = (\mathbf{c}_1, \mathbf{c}_2, \cdots, \mathbf{c}_N)$, where each $\mathbf{a}_i$ is a $\log_2(M)$-bit pattern and each $\mathbf{c}_i$ the corresponding M-ary PPM symbol. Hence a single codeword consists of N PPM symbols. The sequence of PPM symbols is transmitted over a memoryless channel and received as $\mathbf{y}$.

The inner and outer encoding may be described by a graph or trellis consisting of a set of states $\mathcal{V}$, and a set of directed, labeled edges $\mathcal{E}$. The trellis description of the two codes will be used in the decoding algorithm. We describe the notation used for the inner code. The outer code is analogous.

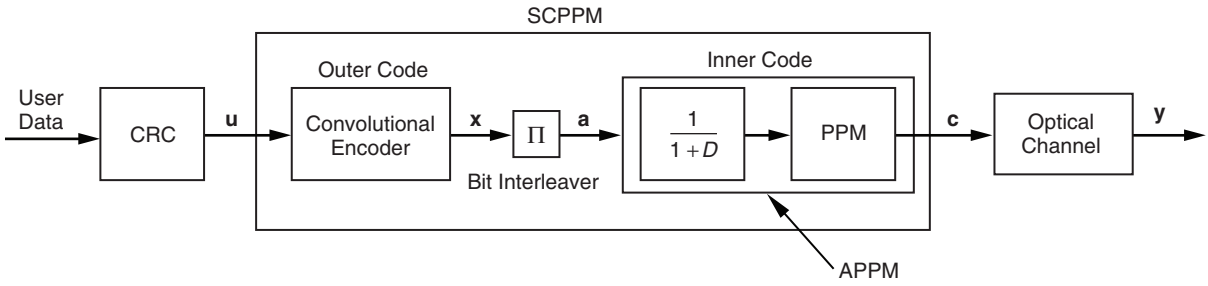

Fig. 2. The SCPPM encoder structure.

Each edge $e \in \mathcal{E}$ has an initial state $i(e)$, a terminal state $t(e)$, an input label $a(e)$, and an output label $c(e)$. Encoding proceeds by following a path through the graph and reading off the output edge labels as follows. Let s_{k-1} be the state at time $k-1$, and e the unique edge with $i(e) = s_{k-1}$ and $a(e) = \mathbf{a}_k$. Then $\mathbf{e}_k = e$, $\mathbf{c}_k = c(e)$, and $s_k = t(e)$. This notation is illustrated in Fig. 3. Let $\mathcal{E}^A_{0,i}\,(\mathcal{E}^C_{0,i})$ be the set of edges with $a(e)_i = 0\,\big(c(e)_i = 0\big)$ and $\mathcal{E}^A_{1,i}\,(\mathcal{E}^C_{1,i})$ the set of edges with $a(e)_i = 1\,\big(c(e)_i = 1\big)$.

III. Decoding Algorithm

A functional block diagram of the SCPPM decoder is illustrated in Fig. 4. The decoder consists of a soft symbol de-mapper, two soft-input soft-output (SISO) modules, which essentially invert the two constituent codes, an interleaver, and a de-interleaver. The SISO algorithms differ in their dimensions and sources of inputs and outputs, but otherwise are the same. Derivations of the SISO algorithm have appeared in various forms in the literature, e.g., [12,13]. We derive the algorithm and present it here to yield a self-contained article. In Section III.E, we note some simplifications to the implementation of the APPM SISO.

A. SISO Module—Updating Likelihoods

Each encoder maps an input sequence, either $\mathbf{u}$ or $\mathbf{a}$, into an output sequence, either $\mathbf{x}$ or $\mathbf{c}$. In the iterative decoding process, each encoder has a corresponding decoder—a SISO module. The SISOs receive, as soft inputs, noisy versions, or likelihoods, of the input and output of the encoder and produce updated likelihoods of the input, or output, or both. These likelihoods may then by transmitted to the other module, where they are treated as noisy inputs.

The inner and outer SISOs are analogous in their operation. We will examine the computation of the bit likelihoods for $\mathbf{a}$, the input to the inner code. The analysis is the same for the outer code.

As seen in Fig. 4, the inner SISO takes as input a priori likelihoods of $\mathbf{a} = (\mathbf{a}_1, \cdots, \mathbf{a}_N)$ and $\mathbf{c} = (\mathbf{c}_1, \cdots, \mathbf{c}_N)$, and produces an updated likelihood of $\mathbf{a}$. Since $\mathbf{y} = (\mathbf{y}_1, \cdots, \mathbf{y}_N)$ is a noisy version of $\mathbf{c}$, with each $\mathbf{y}_i$ corresponding to M-bit PPM symbol $\mathbf{c}_i$, the a priori likelihood of $\mathbf{c}$ given $\mathbf{y}$ can be computed from the channel transition density, $p(y|c)$. Similarly, the a priori likelihood of $\mathbf{a}$ can be thought of as being computed from a noisy version of $\mathbf{a}$, which we denote $\mathbf{w}$. The sequence $\mathbf{w}$ and channel $p(w|a)$ are artificial constructs introduced to aid in analysis of the decoder, as done in [17], and do not appear explicitly in the final algorithm. The a priori likelihoods of $\mathbf{a}$ and $\mathbf{c}$ are assumed to be independent.

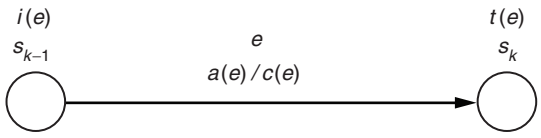

Fig. 3. Trellis edge.

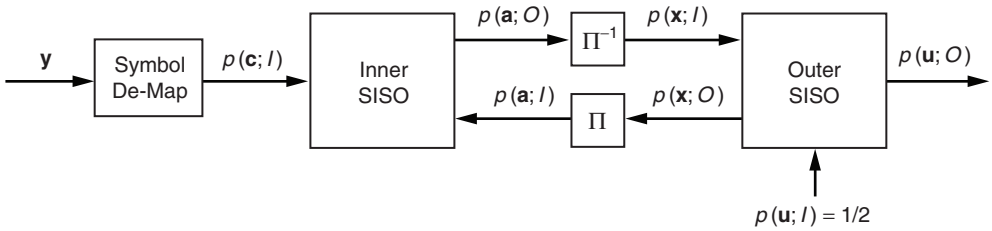

Fig. 4. SCPPM decoder.

Given the noisy observations $\mathbf{y}$ and $\mathbf{w}$, we define a priori likelihoods corresponding to the kth PPM symbol:

$$p(a_{k,i}; I) \triangleq p(w_{k,i}|a_{k,i})$$

$$p(\mathbf{a}_k; I) \triangleq \prod_{i=1}^{\log_2 M} p(a_{k,i}; I)$$

$$p(c_{k,i}; I) \triangleq p(y_{k,i}|c_{k,i})$$

$$p(\mathbf{c}_k; I) \triangleq \prod_{i=1}^{M} p(c_{k,i}; I)$$

denoted with an I to signify they are inputs to the algorithm. This is the information we have about the values of $\mathbf{a}$ and $\mathbf{c}$ from the noisy observations prior to performing decoding, that is, prior to adding knowledge of the dependence between $\mathbf{a}$ and $\mathbf{c}$ imposed by the encoder.

The output of the algorithm, given these inputs, will be the likelihoods

$$\begin{aligned} p(a_{k,i}; O) &\triangleq p(a_{k,i}|\mathbf{y}, \mathbf{w}_{[k,i]}) \\ &= \frac{p(a_{k,i}, \mathbf{y}, \mathbf{w})}{p(w_{k,i}|a_{k,i})p(\mathbf{y}, \mathbf{w}_{[k,i]})} \\ &= \frac{1}{p(a_{k,i}; I)p(\mathbf{y}, \mathbf{w}_{[k,i]})} \sum_{e \in \mathcal{E}^A_{a_{k,i},i}} p_{E_k,\mathbf{Y},\mathbf{W}}(e, \mathbf{y}, \mathbf{w}) \end{aligned}$$

The $p(a_{k,i}; O)$ are referred to in the literature as extrinsic information—the information about $a_{k,i}$ conveyed by $\mathbf{y}$ and $\mathbf{w}_{[k,i]}$. The quantity $p(a_{k,i}|\mathbf{y}, \mathbf{w})$ is referred to in the literature as the a posteriori information. We can interpret the decoder as a mapping of the a priori information into the extrinsic information by factoring out the a posteriori information as

$$p(a_{k,i}; O) = K \frac{p(a_{k,i}|\mathbf{y}, \mathbf{w})}{p(a_{k,i}; I)}$$

where $K = p(w_{k,i}|\mathbf{y}, \mathbf{w}_{[k,i]})$, a constant relative to $a_{k,i}$.

Let

$$\lambda_k(e) \triangleq p_{E_k,\mathbf{Y},\mathbf{W}}(e, \mathbf{y}, \mathbf{w}) \tag{1}$$

In order to avoid computation of $p(\mathbf{y}, \mathbf{w}_{[k,i]})$, we enforce the condition $p_{A_{k,i}}(0; O) + p_{A_{k,i}}(1; O) = 1$ so that

$$p_{A_{k,i}}(0;O) = \frac{1}{K_1 p_{A_{k,i}}(0;I)} \sum_{e \in \mathcal{E}^A_{0,i}} \lambda_k(e)$$

$$p_{A_{k,i}}(1;O) = \frac{1}{K_1 p_{A_{k,i}}(1;I)} \sum_{e \in \mathcal{E}^A_{1,i}} \lambda_k(e) \tag{2}$$

where

$$K_1 = \left(\frac{1}{p_{A_{k,i}}(0;I)} \sum_{e \in \mathcal{E}^A_{0,i}} \lambda_k(e) + \frac{1}{p_{A_{k,i}}(1;I)} \sum_{e \in \mathcal{E}^A_{1,i}} \lambda_k(e) \right)$$

Hence, the algorithm reduces to computing $\lambda_k(e)$ for each edge in the trellis. To this end, let

$$\alpha_k(s) \triangleq p(s_k, \mathbf{y}_{1:k}, \mathbf{w}_{1:k})$$

$$\beta_k(s) \triangleq p(\mathbf{y}_{k+1:N} | s_k)$$

$$\gamma_k(e) \triangleq p(e_k, y_k, w_k | s_{k-1})$$

$$= p(\mathbf{a}_k | s_{k-1}) p(\mathbf{y}_k | \mathbf{w}_k, \mathbf{a}_k, \mathbf{c}_k, s_{k-1})$$

$$p(\mathbf{w}_k | \mathbf{a}_k, s_{k-1}) p(\mathbf{c}_k | \mathbf{w}_k, \mathbf{a}_k, s_{k-1}) \tag{3}$$

$$= K_2 p(\mathbf{w}_k | \mathbf{a}_k) p(\mathbf{y}_k | \mathbf{c}_k) \tag{4}$$

$$= K_2 p_k(\mathbf{a}; I) p_k(\mathbf{c}; I)$$

We obtain Eq. (4) from Eq. (3) by dropping $p(\mathbf{c}_k | \mathbf{w}_k, \mathbf{a}_k, s_{k-1})$, which is an indicator function on allowed trellis edges, assuming that $\mathbf{Y}_k$ is (conditionally) independent of $\mathbf{W}_k$ and $\mathbf{A}_k$ given $\mathcal{C}_k$, and assuming that $K_2 = p(\mathbf{a}_k | s_{k-1})$, the a priori probability of each outgoing edge label, is a constant.

Factor Eq. (1) as

$$\lambda_k(e) = \alpha_{k-1}\big(i(e)\big) \gamma_k(e) \beta_k\big(t(e)\big) \tag{5}$$

It can be shown that the the following recursive equations may be used to compute the α's and β's:

$$\alpha_k(s) = \sum_{e: t(e) = s} \alpha_{k-1}\big(i(e)\big) \gamma_k(e) \tag{6}$$

$$\beta_k(s) = \sum_{e: i(e) = s} \beta_{k+1}\big(t(e)\big) \gamma_{k+1}(e) \tag{7}$$

The algorithm is initialized by setting $\alpha_1(e)$ and $\beta_N(e)$ to reflect the initial and terminal states of the encoder.

Note that in Eq. (2) multiplying each $\lambda_k(e)$ by a constant independent of e will not affect the result. Similarly, by Eq. (5), one can multiply the α's, β's, or γ's by a constant for each k to prevent over- or under-flow in Eqs. (6) and (7).

In an analogous manner, the outer SISO receives noisy versions of the input ($\mathbf{u}$) and output ($\mathbf{x}$) of the outer code and computes bit likelihoods $p(u_{k,i}; O)$ or $p(x_{k,i}; O)$. The operations of both SISOs will be given in detail in the Appendix.

B. Algorithm Summary

In the following, we summarize the steps involved in one decoding, or iteration, of the inner SISO as it might be implemented in software. The encoder is assumed to start in state $s = 0$ and terminate in an arbitrary state.

(1) Receive symbol likelihoods $p(\mathbf{c}_k; I)$ from the channel and bit likelihoods $p(a_{k,i} = 0; I)$ from the outer code (on the first iteration, set $p(a_{k,i} = 0; I) = 1/2$). Compute

$$\begin{aligned} p(\mathbf{a}_k; I) &= \prod_{i=1}^{\log_2 M} p(a_{k,i}; I) \\ &= \prod_{i=1}^{\log_2 M} \big(a_{k,i} + (-1)^{a_{k,i}} p(a_{k,i} = 0; I)\big) \end{aligned}$$

for $k = 1, \cdots, N$ and each $\log_2(M)$-ary bit pattern $\mathbf{a}_k$.

(2) Compute

$$\gamma_k(e) = p(\mathbf{a}_k; I) p(\mathbf{c}_k; I)$$

for $e \in \mathcal{E}$ and $k = 1, \cdots, N$.

(3) Initialize

$$\alpha_1(s) = \begin{cases} 1, & s = 0 \\ 0, & \text{otherwise} \end{cases}$$

$$\beta_N(s) = \frac{1}{2}$$

Recursively compute

$$\alpha_k(s) = \sum_{e: t(e) = s} \alpha_{k-1}\big(i(e)\big) \gamma_k(e)$$

$$\beta_k(s) = \sum_{e: i(e) = s} \beta_{k+1}\big(t(e)\big) \gamma_{k+1}(e)$$

for $s \in \mathcal{V}$ and $k = 1, \cdots, N$.

(4) Compute

$$\lambda_k(e) = \alpha_{k-1}\big(i(e)\big)\gamma_k(e)\beta_k\big(t(e)\big)$$

for $e \in \mathcal{E}$ and $k = 1, \cdots, N$.

(5) Compute

$$\frac{1}{p_{A_{k,i}}(0; I)} \sum_{e \in \mathcal{E}^A_{0,i}} \lambda_k(e) \tag{8}$$

$$\frac{1}{p_{A_{k,i}}(1; I)} \sum_{e \in \mathcal{E}^A_{1,i}} \lambda_k(e) \tag{9}$$

and $p(a_{k,i}; O)$ from Eq. (2) for $k = 1, \cdots, N$, $i = 1, \cdots, \log_2(M)$.

C. Log-Domain Implementation

The SISO algorithm may be implemented in the log domain, which translates multiplications into additions, and may be less sensitive to roundoff errors in fixed-point arithmetic. We extend our example, decoding of the inner code. The algorithm may be converted in a straightforward manner by defining

$$\bar{\alpha}_k(s) \triangleq \log \alpha_k(s)$$

$$\bar{\beta}_k(s) \triangleq \log \beta_k(s)$$

$$\bar{\gamma}_k(s) \triangleq \log \gamma_k(s)$$

$$\bar{\lambda}_k(e) \triangleq \log \lambda_k(e)$$

$$\bar{p}(a_{k,i}; I) \triangleq \log \frac{p_{A_{k,i}}(0; I)}{p_{A_{k,i}}(1; I)}$$

$$\bar{p}(a_{k,i}; O) \triangleq \log \frac{p_{A_{k,i}}(0; O)}{p_{A_{k,i}}(1; O)}$$

$$\pi(\mathbf{c}_k; I) \triangleq \log p(\mathbf{c}_k; I) + \text{constant}$$

$$\pi(\mathbf{a}_k; I) \triangleq \log p(\mathbf{a}_k; I) + \text{constant}$$

As input to the SISO algorithm, we receive the bit log-likelihood ratios (LLRs) $\bar{p}(a_{k,i}; I)$ from the outer code and symbol log-likelihoods $\pi(\mathbf{c}_k; I)$ from the channel. We first compute the symbol log-likelihoods $\pi(\mathbf{a}_k; I)$. Note that for $a_{k,i} \in \{0, 1\}$

$$\log p(a_{k,i}; I) = \frac{1}{2}(-1)^{a_{k,i}}\bar{p}(a_{k,i}; I) + \frac{1}{2}\log\left(p_{A_{k,i}}(0; I)p_{A_{k,i}}(1; I)\right) \tag{10}$$

In an analogous manner to the product domain, in the log domain we may add a constant to each $\pi(\mathbf{a}_k; I)$ (or $\bar{\gamma}$, $\bar{\beta}$, $\bar{\alpha}$, $\bar{\lambda}$) in a trellis stage. The second term in Eq. (10) is a constant relative to $a_{k,i}$; hence, we may use

$$\pi(\mathbf{a}_k; I) = \sum_{i=1}^{\log_2 M} \frac{1}{2}(-1)^{a_{k,i}}\bar{p}(a_{k,i}; I)$$

as the symbol likelihoods.

Transforming the remainder of the algorithm yields

$$\bar{\gamma}_k(e) = \pi(\mathbf{a}_k; I) + \pi(\mathbf{c}_k; I)$$

$$\begin{aligned}\bar{\alpha}_k(s) &= \log \sum_{e:t(e)=s} \alpha_{k-1}\big(i(e)\big)\gamma_k(e) \\ &= \log \sum_{e:t(e)=s} \exp\left(\bar{\alpha}_{k-1}\big(i(e)\big) + \bar{\gamma}_k(e)\right)\end{aligned}$$

$$\bar{\beta}_k(s) = \log \sum_{e:i(e)=s} \exp\left(\bar{\beta}_{k+1}\big(t(e)\big) + \bar{\gamma}_{k+1}(e)\right)$$

$$\bar{\lambda}_k(e) = \bar{\alpha}_{k-1}\big(i(e)\big) + \bar{\gamma}_k(e) + \bar{\beta}_k\big(t(e)\big)$$

$$\bar{p}(a_{k,i}; O) = \overset{*}{\max}_{e\in\mathcal{E}^A_{0,i}}\left\{\bar{\lambda}_k(e)\right\} - \overset{*}{\max}_{e\in\mathcal{E}^A_{1,i}}\left\{\bar{\lambda}_k(e)\right\} - \bar{p}(a_{k,i}; I)$$

D. The $\overset{*}{\max}$ Operation

To obtain $\bar{\alpha}$ and $\bar{\beta}$, we must evaluate a function of the form $\log\sum_i \exp(a_i)$. We'll look at some of the properties of this function. Let $\overset{*}{\max}$ denote the binary function

$$a \overset{*}{\max} b \triangleq \log(e^a + e^b)$$

The $\overset{*}{\max}$ function has the following properties:

(1) Commutativity: $a \overset{*}{\max} b = b \overset{*}{\max} a$

(2) Associativity: $(a \overset{*}{\max} b) \overset{*}{\max} c = a \overset{*}{\max} (b \overset{*}{\max} c)$

(3) Identity element $(-\infty)$: $a \overset{*}{\max} (-\infty) = a$

(4) Addition distributive over $\overset{*}{\max}$: $a + (b \overset{*}{\max} c) = (a + b) \overset{*}{\max} (a + c)$

The first three properties are straightforward. The last follows as

$$\begin{aligned}(a+b) \overset{*}{\max} (a+c) &= \log(e^{a+b}+e^{a+c}) \\ &= \log\left(e^a(e^b+e^c)\right) \\ &= a+\log(e^b+e^c) \\ &= a+(b \overset{*}{\max} c)\end{aligned}$$

Since $\overset{*}{\max}$ is associative and commutative, we will write

$$a_1 \overset{*}{\max} a_2 \overset{*}{\max} \cdots \overset{*}{\max} a_p = \overset{*}{\max}_{i=1,\cdots,p} \{a_i\} \tag{11}$$

Finally, note that

$$\log(e^b+e^c) = \max(b,c) + \log(1+e^{-|b-c|})$$

By pre-computing a table of $\log(1+e^{-|b-c|})$ for a range of values of $|b-c|$, we have a low-complexity implementation of $\overset{*}{\max}$ in hardware.

E. Simplified Computation with Parallel Edges

The APPM trellis has 2 states and $2M$ edges per stage. The forward and backward recursions and mapping of $\bar{\lambda}$s to bit probabilities on this trellis require $\overset{*}{\max}$ operations with M arguments. Suppose each 2-input $\overset{*}{\max}$ operation requires one clock cycle. A straightforward implementation using Eq. (11) would incur a delay of $\log_2(M)$ clock cycles. We show here how the computation may be pipelined, reducing the M-input $\overset{*}{\max}$ operation to a 2-input $\overset{*}{\max}$ operation that may be completed in one clock cycle. We consider the $\bar{\beta}$ recursion; all others follow in a similar manner.

In the product domain, it is straightforward to see an application of the distributive law (multiplication distribution over addition) saves computations on a trellis with parallel edges:

$$\begin{aligned}\beta_k(s) &= \sum_{e:i(e)=s} \beta_{k+1}\big(t(e)\big)\gamma_{k+1}(e) \\ &= \sum_{e:i(e)=s,t(e)=s} \beta_{k+1}(s)\gamma_{k+1}(e) + \sum_{e:i(e)=s,t(e)=s'} \beta_{k+1}(s')\gamma_{k+1}(e) \\ &= \left(\beta_{k+1}(s) \sum_{e:i(e)=s,t(e)=s} \gamma_{k+1}(e)\right) + \left(\beta_{k+1}(s') \sum_{e:i(e)=s,t(e)=s'} \gamma_{k+1}(e)\right) \\ &= \beta_{k+1}(s)\gamma'_{k+1}(s,s) + \beta_{k+1}(s')\gamma'_{k+1}(s,s')\end{aligned} \tag{12}$$

where

$$\gamma'_{k+1}(s,\tilde{s}) = \sum_{e:i(e)=s,t(e)=\tilde{s}} \gamma_{k+1}(e)$$

a sum over parallel edges.

We have an analogous simplification in the log domain via the distributive law (addition distributive over $\overset{*}{\max}$) which can be seen by taking logarithms of both sides of Eq. (12):

$$\begin{aligned}\bar{\beta}_k(s) &= \log\left(\exp\left(\bar{\beta}_{k+1}(s) + \bar{\gamma}'_{k+1}(s,s)\right) + \exp\left(\bar{\beta}_{k+1}(s') + \bar{\gamma}'_{k+1}(s,s')\right)\right) \\ &= \overset{*}{\max}_{\tilde{s}\in\{s,s'\}} \left\{\bar{\beta}_{k+1}(\tilde{s}) + \bar{\gamma}'_{k+1}(s,\tilde{s})\right\}\end{aligned}$$

where

$$\begin{aligned}\bar{\gamma}'_k(s,\tilde{s}) &= \log\left(\sum_{e:i(e)=s,t(e)=\tilde{s}} e^{\bar{\gamma}_{k+1}(e)}\right) \\ &= \overset{*}{\max}_{e:i(e)=s,t(e)=\tilde{s}} \left\{\bar{\gamma}_{k+1}(e)\right\}\end{aligned}$$

Since the $\bar{\gamma}'_k$s are not a function of a recursively computed quantity, they may be pre-computed via a pipeline as illustrated in Fig. 5 [18].

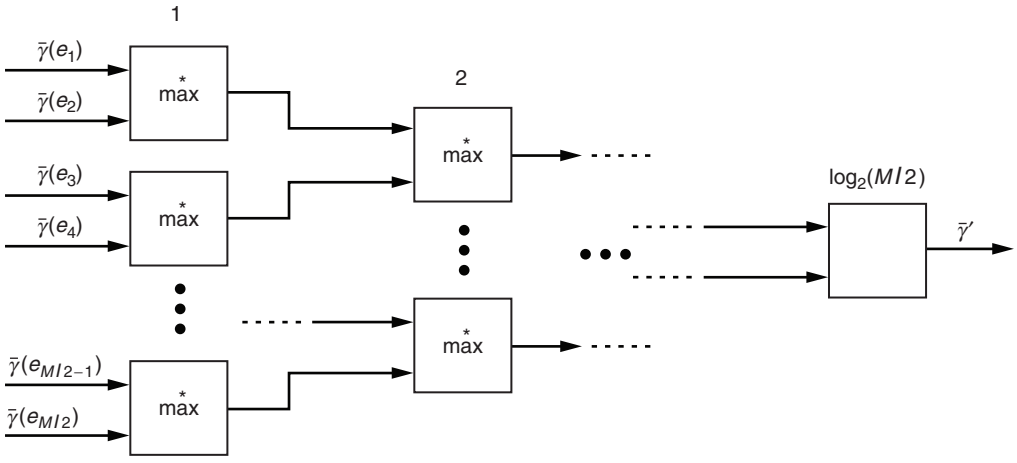

Fig. 5. Pipelined $\overset{*}{\max}$ computation.

IV. Channel Likelihoods—Partial Statistics

The output of a photon-counting detector, e.g., a photomultiplier tube, may be modeled as a Poisson process, with mean n_b in a noise slot and $n_s + n_b$ in a signal slot,

$$p_{Y|C}(k|1) = \frac{(n_s + n_b)^k e^{-(n_s+n_b)}}{k!} \tag{13}$$

$$p_{Y|C}(k|0) = \frac{n_b^k e^{-n_b}}{k!}$$

Let $\mathbf{c}^{(j)}$ denote the PPM symbol with a pulse in the jth slot.

In the special case $n_b = 0$, the PPM channel reduces to an M-ary erasure channel with

$$p_{\mathbf{C}_k|\mathbf{Y}_k}\left(\mathbf{c}^{(j)}|\mathbf{y}_k\right) = \begin{cases} 1, & y_{k,j} > 0 \\ 1/M, & \mathbf{y}_k = \mathbf{0} \\ 0, & y_{k,i} > 0, i \neq j \end{cases}$$

$$p_{\mathbf{Y}_k|\mathbf{C}_k}\left(\mathbf{y}_k|\mathbf{c}^{(j)}\right) = M p_{\mathbf{Y}_k}(\mathbf{y}_k) p_{\mathbf{C}_k|\mathbf{Y}_k}\left(\mathbf{c}^{(j)}|\mathbf{y}_k\right)$$

Factoring out $M p_{\mathbf{Y}_k}$, which is not a function of $\mathbf{c}^{(j)}$, the likelihoods depend only on whether photons are observed—not on their number. Hence, in zero background it is sufficient to transmit the location (or absence) of observed photons in each symbol.

When $n_b > 0$, the channel log-likelihoods are

$$\begin{aligned} \pi\left(\mathbf{c}^{(j)}; I\right) &= \log p(\mathbf{y}_k|\mathbf{c}^{(j)}) \\ &= \log \frac{p(y_{k,j}|c_{k,j} = 1)}{p(y_{k,j}|c_{k,j} = 0)} + \text{constant} \\ &= y_{k,j} \log\left(1 + \frac{n_s}{n_b}\right) + \text{constant} \end{aligned}$$

To realize the gains of iterative decoding algorithms would nominally require a likelihood be computed and stored for every slot of each PPM symbol in a codeword. However, high data rates, large values of M, and large interleavers can make likelihood storage and processing prohibitively expensive.

To reduce the complexity of iterative decoding, we may discard the majority of the channel likelihoods [19], operating the decoder using only the remainder. This may be accomplished by transmitting only a subset consisting of the largest likelihoods during each symbol duration—the likelihoods corresponding to the slots with the largest number of observed photons. The received values in the remaining slots are set to the mean of a noise slot. In small background, a small subset may be chosen with negligible loss.

V. Interleaver

Let f be the interleaver function, such that a bit at position i at the input to the interleaver is mapped to position $f(i)$ at the output. If f maps $S = \{0, 1, \cdots, N_I - 1\}$ one-to-one and onto S, f is a valid interleaver. Good performance has been observed with iterative coding schemes using a pseudo-random interleaving function f. However, pseudo-random functions may be complex to implement in hardware, requiring a table look-up. The SCPPM code uses a permutation polynomial (PP), which has low implementation complexity.

A second-degree polynomial $f(x) = (ax + bx^2) \bmod N_I$ with N_I divisible by 4 or not divisible by 2 is a PP (a valid interleaver) if and only if b is divisible by the prime factors of N_I and a is not [20]. Let $R_o = k_o/n_o$ be the rate of the outer code and $R_i = k_i/n_i$ the rate of the inner code. It is convenient for N_I to be divisible by n_o and k_i. For example, the value $N_I = 15120 = 2^4 \times 3^3 \times 5 \times 7$ was chosen for the MLCD SCPPM code to allow flexibility in the choice of inner and outer code rates.[2] Candidate quadratic interleavers for $N_I = 15120$ are of the form $f(x) = (ax + 210nx^2)$, where n is a positive integer and a doesn't have 2,3,5, or 7 as a factor. Among this class, we have observed good performance with the polynomial $f(x) = 11x + 210x^2$. We have observed no loss with this interleaver relative to pseudo-randomly generated interleavers chosen with a large spread.

A. Implementation of the Mapping

Barron and Robinson[3] have demonstrated a recursive implementation of a permutation polynomial interleaver mapping that requires only additions. Let $[\cdot]$ be the mod N_I operator. Expand $f(x + 1)$:

$$\begin{aligned} f(x+1) &= \big[a(x+1) + b(x+1)^2\big] \\ &= \big[[ax + bx^2] + [a + b + 2bx]\big] \\ &= \big[f(x) + g(x)\big] \end{aligned}$$

where $g(x) = [a + b + 2bx]$. Expanding $g(x)$ similarly yields

$$g(x+1) = \big[g(x) + 2b\big]$$

Note that $\big(f(x) + g(x)\big) \leq 2N_I - 1$. Hence the modulo operations for g or f may be implemented with a comparison:

$$f(x+1) = \begin{cases} f(x) + g(x), & \text{if } f(x) + g(x) < N_I \\ f(x) + g(x) - N_I, & \text{otherwise} \end{cases}$$

B. Inverses of Permutation Polynomials

It would be convenient to be able to implement the inverse of a PP with low-degree PP. One can show the PP interleaver $f(x) = 11x + 210x^2$ has a quadratic inverse.[4] Under what conditions does this hold and how do we find the inverse?

---

[2] Personal communication, D. Boroson, Lincoln Laboratory, Massachusetts Institute of Technology, Lexington, Massachusetts, August 2004.

[3] R. Barron and B. Robinson, "Recursive Polynomial Interleaver Algorithm," Interoffice Memorandum (internal document), Lincoln Laboratory, Massachusetts Institute of Technology, Lexington, Massachusetts, September 2004.

[4] This was brought to our attention by R. Barron and B. Robinson.

The composition of two PPs is a permutation polynomial (let f, g be PPs; then $f(g(S)) = S$). Since there are a finite number of permutations of S, each PP f has a PP inverse, obtained by composing f with itself a sufficient number of times. Hence, the collection of PPs forms a group with binary operation composition. This yields a straightforward algorithm to find the inverse of a PP f by self-composition:

```
h(x) = f(x)
while f(h(x)) ≠ x
   h(x) = f(h(x))
end
```

This algorithm allows one to show that the degree of the inverse of a quadratic PP is no larger than the largest power of a prime divisor of N_I:

Lemma 1. *If f is a second degree PP, then the n-fold composition of f with itself, reduced modulo N_I, has degree no larger than the largest power of a prime divisor of N_I.*

Proof. Let $f(x) = ax + bx^2$ be a PP over N_I and m the largest power of a prime divisor of N_I. The coefficient of the degree k terms in f are divisible by b^{k-1}. If the coefficient of the degree k terms in $f^{(n)}$ are divisible by b^{k-1}, then the coefficients of the degree k terms in

$$f^{(n+1)} = af^{(n)} + b\left(f^{(n)}\right)^2$$

are clearly divisible by b^{k-1}; b is divisible by the factors of N_I, hence $b^m = 0 \bmod N_I$ and $f^{(n)}$ contains no terms of degree larger than m. ❒

Applying the composition algorithm to $f(x) = 11x + 210x^2$ yields the inverse

$$f^{-1}(x) = 12371x + 7770x^2 + 2520x^3 + 7560x^4$$

which may be reduced to the equivalent quadratic polynomials

$$f^{-1}(x) \equiv 7331x + 7770x^2$$

$$\equiv 14891x + 210x^2$$

VI. Stopping Rule

Iterations between the inner and outer code are terminated based on a stopping rule. Let $\mathcal{C}_{CRC}$ denote the collection of CRC codewords and $\mathcal{C}_o$ the collection of outer (convolutional) codewords. Let $\hat{\mathbf{u}}$ and $\hat{\mathbf{x}}$ denote the binary vectors of bit estimates made on the corresponding a posteriori bit likelihoods. The SCPPM decoder terminates iterations if $\hat{\mathbf{u}} \in \mathcal{C}_{CRC}$ and $\hat{\mathbf{x}} \in \mathcal{C}_o$. If this does not occur, iterations are ended after a maximum is reached.

Checking whether $\hat{\mathbf{u}} \in \mathcal{C}_{CRC}$ and $\hat{\mathbf{x}} \in \mathcal{C}_o$ are the well-known problems of computing the syndrome of a cyclic and convolutional code, respectively. Both checks may be implemented with simple, well-known circuits in hardware, e.g., [21,22].

We have observed no degradation in performance with this joint rule relative to a genie-aided stopping rule which terminates iterations when decoding converges to the correct codeword. The genie-aided rule terminates at the minimum number of iterations required for correct decoding. The joint rule may stop sooner (yielding an undetected error) but no later, since it stops once a valid codeword is found. Hence, the average number of iterations is no greater than that with the genie-aided rule.

A. Probability of Undetected Codeword Error

The SCPPM decoder *terminates* iterations if $\hat{\mathbf{u}} \in \mathcal{C}_{CRC}$ and $\hat{\mathbf{x}} \in \mathcal{C}_o$ and *ends* iterations after a maximum is reached. A codeword error is declared if the maximum is reached without termination. An error is undetected if the decoder terminates iterations and $\hat{\mathbf{u}} \neq \mathbf{u}$. What is the probability of undetected error for this decoder?

To simplify analysis, assume the decoder runs a fixed number of iterations after which it decides whether a codeword error has occurred based on the described criteria. Then

$$\begin{aligned} P_{ud} =& P(\hat{\mathbf{u}} \neq \mathbf{u}, \hat{\mathbf{x}} \in \mathcal{C}_o, \hat{\mathbf{u}} \in \mathcal{C}_{CRC}) \\ =& P(\hat{\mathbf{u}} \neq \mathbf{u}, \hat{\mathbf{u}} \in \mathcal{C}_{CRC}) P(\hat{\mathbf{x}} \in \mathcal{C}_o | \hat{\mathbf{u}} \neq \mathbf{u}, \hat{\mathbf{u}} \in \mathcal{C}_{CRC}) \end{aligned}$$

In order to evaluate $P(\hat{\mathbf{u}} \neq \mathbf{u}, \hat{\mathbf{u}} \in \mathcal{C}_{CRC})$, we require a distribution on error patterns that result from estimates on $p(\mathbf{u}; O)$ and a description of the CRC. Assume the error patterns may be modeled as the output of a binary symmetric channel with appropriately chosen crossover probability and an l-bit CRC. Under this assumption, $P(\hat{\mathbf{u}} \neq \mathbf{u}, \hat{\mathbf{u}} \in \mathcal{C}_{CRC})$ approaches 2^{-l} in the limit of large block lengths [23] (a good approximation for block lengths larger than 500). We'll use this approximation to obtain the (approximate) bound

$$P_{ud} \lessapprox 2^{-l} P(\hat{\mathbf{x}} \in \mathcal{C}_o | \hat{\mathbf{u}} \neq \mathbf{u}, \hat{\mathbf{u}} \in \mathcal{C}_{CRC})$$

The second term corresponds to the event that the decoder chooses a valid codeword given it has chosen a valid, but incorrect, input pattern. We conjecture the probability of this event for a large codeword is very small at all operating points. We have yet to observe an undetected error at any operating point in extensive simulations ($>10^6$ codeword errors) with an $l = 16$-bit CRC.

VII. Bit-to-Symbol Mapping

On a memoryless channel, the mapping of user bits to PPM symbols has no effect on performance (one could convert one bit-to-symbol mapping to another by re-ordering the sequence in which the slots are transmitted). However, real-world effects such as timing errors and finite bandwidth responses to photon arrivals introduce inter-slot interference (ISI) and memory to the channel. In the presence of ISI, a fraction of each pulse energy may appear in an adjacent slot. Noise due to ISI alters the symmetry of the signal set. For example, PPM symbols with pulses in adjacent slots may be more likely to be confused with one another in ISI. In this case, the performance will depend on the bit-to-symbol mapping.

Figure 6 illustrates the inner APPM code. The sequence $\mathbf{a}$ is passed through a $1/(1+D)$ filter to produce the sequence $\mathbf{b}$. Each consecutive $\log_2 M$ bits of $\mathbf{a}$ map to a corresponding $\log_2 M$-bit pattern of $\mathbf{b}$, which is mapped to an M-ary PPM symbol $\mathbf{c}$.

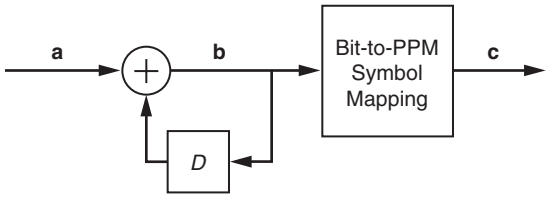

Fig. 6. The APPM inner code.

Consider three candidate bit-to-symbol mappings for the APPM code: natural, Gray, and anti-Gray, illustrated in Table 1 for $M = 8$. In the table, $\mathbf{a}$ is the 3-bit input to an accumulator in the given state, and $\mathbf{b}$ the corresponding output. Patterns are inserted to the accumulator from right least significant bit (lsb) to left most significant bit (msb). In a natural mapping, $\mathbf{b}$ is mapped to a pulse in position $\mathbf{b}$ (the decimal representation of $\mathbf{b}$). In a Gray-mapping, pairs of inputs $\mathbf{a}_1, \mathbf{a}_2$ with minimal Hamming distance (one) are mapped to pairs of PPM symbols $\mathbf{c}_1, \mathbf{c}_2$ with pulses in adjacent slots. In an anti-Gray mapping, input pairs $\mathbf{a}_1, \mathbf{a}_2$ with maximal Hamming distance ($\log_2(M)$ or $\log_2(M) - 1$) are mapped to pairs of PPM symbols with pulses in adjacent slots.

Note that for Gray and anti-Gray mappings the properties are defined relative to $\mathbf{a}$, the *input* to the accumulator for each accumulator state. Following convention, the natural mapping is relative to $\mathbf{b}$, the *output* of the accumulator. For all mappings, the mapping of $\mathbf{b}$, the *output* of the accumulator, to PPM symbols is consistent for both accumulator states. Changing the accumulator state inverts the first output bit of the accumulator for the same input. Hence, by defining mappings relative to inputs $\mathbf{a}$ for state 0 and preserving the corresponding mapping of outputs $\mathbf{b}$ for state 1, we obtain a consistent mapping property for both states. This allows one to construct a fixed mapping of the outputs of the accumulator to PPM symbols.

Table 1. Bit-to-symbol APPM mappings, $M = 8$.

State	**a**	**b**	PPM pulse position		
			Natural	Gray	Anti-Gray
0	000	000	0	0	0
0	001	111	7	7	5
0	010	110	6	3	6
0	011	001	1	4	3
0	100	100	4	1	2
0	101	011	3	6	7
0	110	010	2	2	4
0	111	101	5	5	1
1	000	111	7	7	5
1	001	000	0	0	0
1	010	001	1	4	3
1	011	110	6	3	6
1	100	011	3	6	7
1	101	100	4	1	2
1	110	101	5	5	1
1	111	010	2	2	4

There exist several standard constructions of Gray mappings. We construct a Gray code by label reflection: (1) take a Gray code of length n and write it as a matrix with each codeword a row of the matrix, (2) double the number of rows by adding the reflection of the code, and (3) append the sequence $0^n 1^n$ to the last column. In the resulting table, each row is distinct and is Hamming distance one from its neighbor.

We construct an anti-Gray code as follows. Take the first half of the Gray code constructed by label reflection. After each codeword, insert its inverse, i.e., bit-wise exclusive OR with the all-ones pattern. Since the Gray codewords were distinct and all agreed in the last position, the anti-Gray codewords are distinct. Each codeword has neighbors at distances n and $n-1$, which yield the maximum achievable average Hamming distance between adjacent codewords.

We have observed gains in using the anti-Gray mapping relative to the natural or Gray mappings on channels with memory due to ISI. The relative gain depends on the magnitude and nature of the ISI. For example, modeling the channel pulse shape as a truncated sinc pulse with a main lobe that is half the slot width and assuming uniform pulse arrivals in a slot produces ISI. For this ISI model, we observe gains of 0.3 dB of the anti-Gray mapping relative to the Gray or natural mapping.[5]

VIII. Performance

Due to changes in atmospheric conditions and the Sun–Earth–spacecraft geometry, the deep-space optical channel sees wide variations in the signal and noise power. Figures 7 through 9 illustrate performance of an SCPPM code relative to $(n, k) = (4085, 2047)$ RS-coded PPM (RSPPM) and capacity at three different representative operating points for the MLCD mission. The SCPPM code uses PPM order $M = 64$, the constraint length 3, rate 1/2, distance 5, (5,7) outer convolutional code, the permutation polynomial interleaver of length 15120 described in Section V, and the joint stopping rule described in

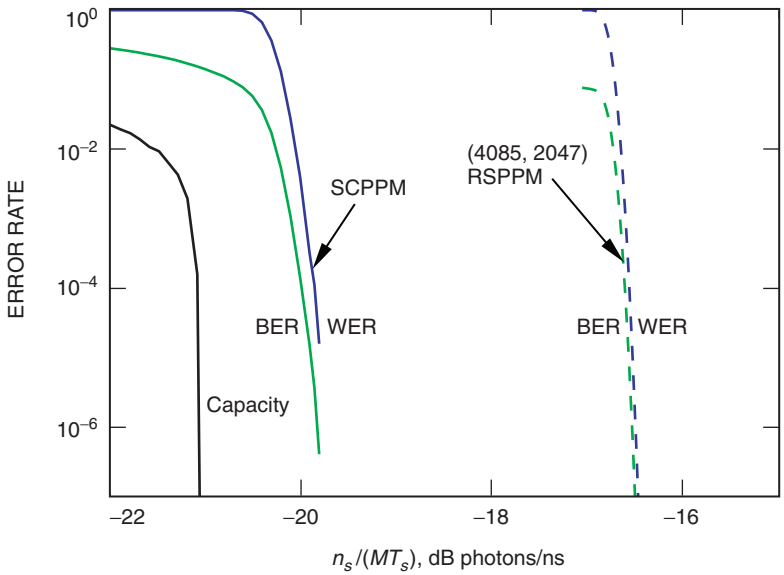

Fig. 7. SCPPM and RSPPM performance, Poisson channel, $M = 64$, $n_b = 0.0025$, $T_s = 1.6$ ns.

[5] B. Moision, M. Srinivasan, and C. Lee, "Sequence Detection for the Optical Channel in the Presence of ISI," JPL Interoffice Memorandum (internal document), Jet Propulsion Laboratory, Pasadena, California, February 2004.

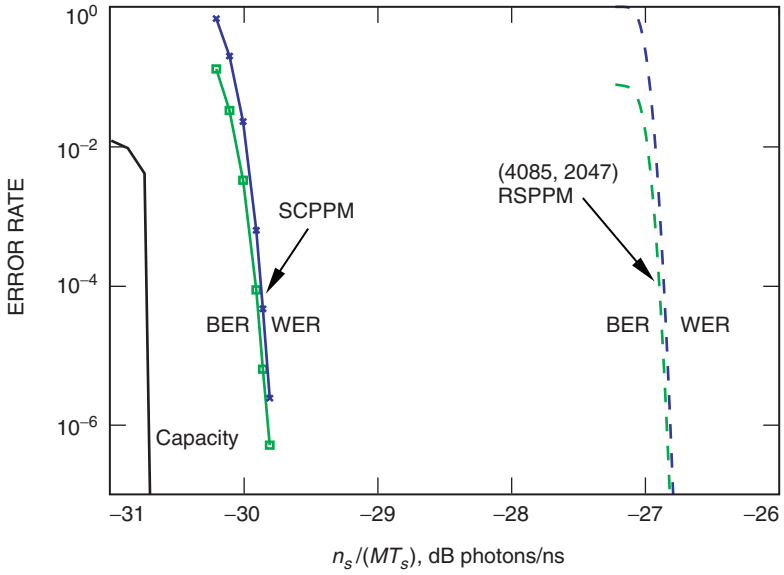

Fig. 8. SCPPM and RSPPM performance, Poisson channel, $M = 64$, $n_b = 0.2$, $T_s = 32$ ns.

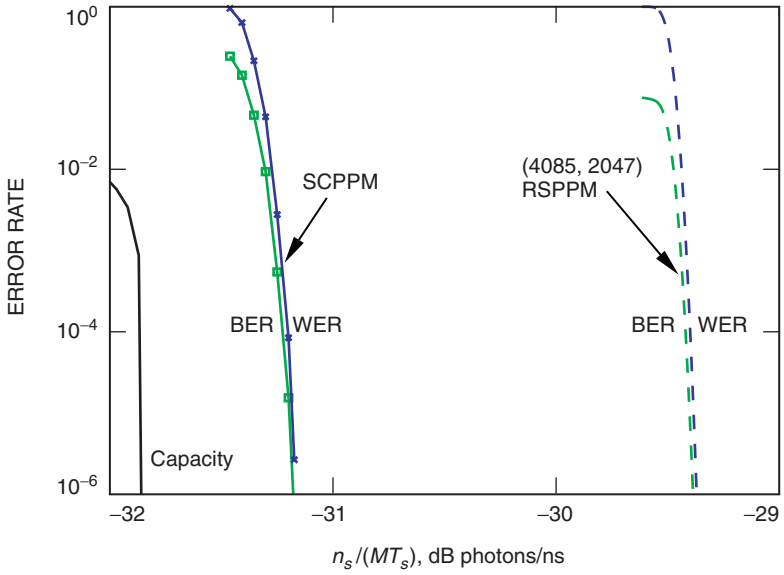

Fig. 9. SCPPM and RSPPM performance, Poisson channel, $M = 64$, $n_b = 40.0$, $T_s = 350$ ns.

Section VI. An explicit summary of the decoding algorithm for this code is presented in the Appendix. The RS block length was chosen to yield the best performance over all RS codes with the same rate associating an integer number of PPM symbols with each code symbol [6]. Signal power is scaled by the slot width, which is varied to support different throughputs. We see SCPPM performance 0.7, 0.9, and 1.2 dB from capacity for background noise $n_b = 0.0025, 0.2$, and 40.0, respectively. This represents gains of 3.3, 3.0, and 1.8 dB over RSPPM.

IX. Alternate Decoding Algorithms

The SCPPM code is composed of three constituent codes: a (5,7) convolutional code (CC), a $1/(1+D)$ accumulator (A), and a PPM mapping. In this section, we look at the trade-off of performance and complexity by changing the way we decode these three building blocks of the SCPPM code.

In the implementation proposed in this article, the accumulator and PPM mapping are treated as a single code, referred to as the APPM code. The APPM code may be decoded using the SISO algorithm described earlier on a trellis with 2 states and $2M$ edges. This may be prohibitively complex for large M. We may reduce the complexity by separately decoding the accumulate and PPM codes or by not including certain modules in iterations. Each of the simplifications comes at a loss in performance.

Figures 10 and 11 illustrate performance of a number of ways of utilizing the same three building blocks. The channel is the Poisson channel given by Eq. (13) with $M = 64$ and $n_b = 0.2$. PPM, APPM, A, and CC denote SISO modules for the associated code. An arrow denotes the direction that information is passed. A bidirectional arrow denotes iteration between the SISO modules. When information is passed iteratively, the bit likelihoods are interleaved (otherwise we see a sharp performance degradation). For example, $CC \leftrightarrow A \leftarrow PPM$ denotes soft decoding of PPM followed by iterations between the accumulate and convolutional decoders.

With more than two SISO modules in iteration, the order of decoding becomes an issue. However, the order of iterations affects only the decoding complexity and not the performance. With three modules, we use the order $PPM \rightarrow A \rightarrow CC \rightarrow A \rightarrow PPM$.

Table 2 lists the relative performance and complexity of the various decoding algorithms. Each is measured relative to (non-iterative) decoding of $CC \leftarrow A \leftarrow PPM$. Complexity is measured as the average number of trellis edges traversed by the decoding algorithm in decoding one codeword. For iterative decoding, the average iterations required to achieve a bit-error rate of 10^{-5} were used. This is a first-order estimate of relative complexity, ignoring the cost of additional interleavers or, for example, the ability to parallelize the PPM SISO in the absence of memory.

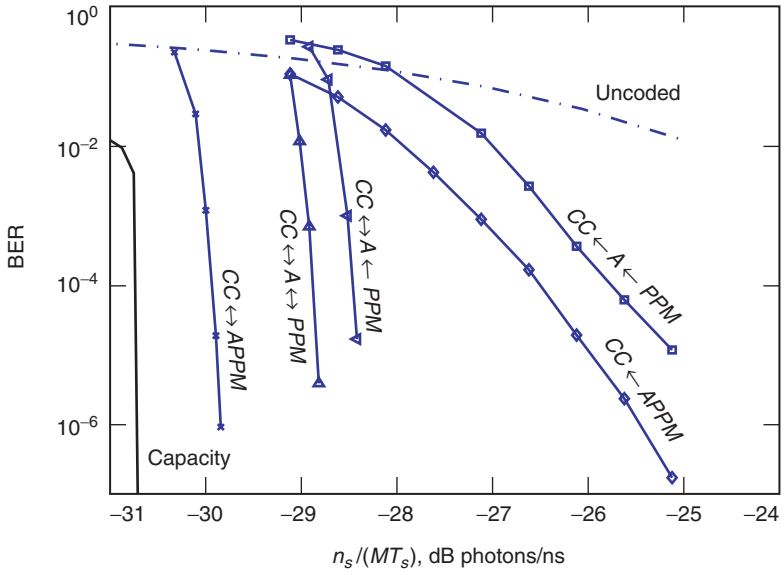

Fig. 10. Bit-error rates, various ways of decoding SCPPM, Poisson channel, $M = 64$, $n_b = 0.2$, $T_s = 32$ ns.

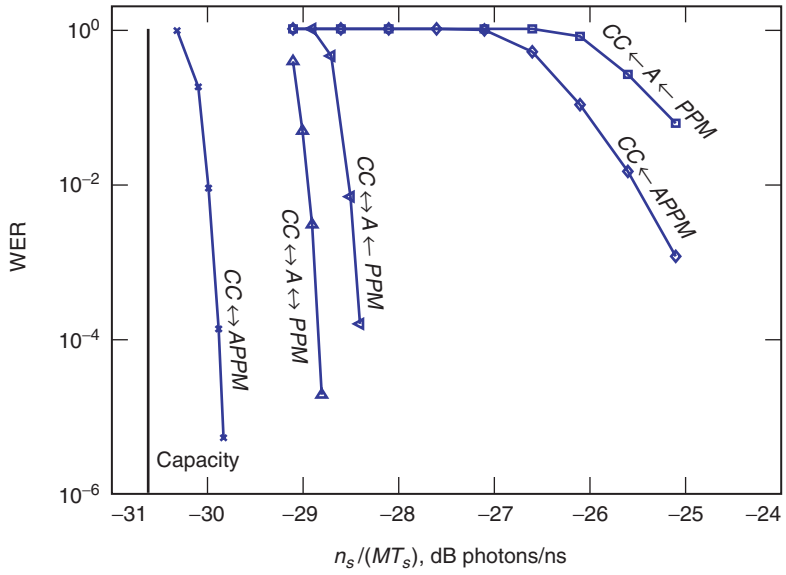

Fig. 11. Word-error rates, various ways of decoding SCPPM, Poisson channel, $M = 64$, $n_b = 0.2$, $T_s = 32$ ns.

Table 2. Decoding algorithm performance and complexity trade-off, $M = 64, n_b = 0.2$.

Decoding algorithm	Normalized average edge operations/codeword	$n_s/(MT_s)$ gain at BER $= 10^{-5}$, dB
$CC \leftarrow A \leftarrow PPM$	1	0
$CC \leftarrow APPM$	1.3	1
$CC \leftrightarrow A \leftarrow PPM$	3.3	3.45
$CC \leftrightarrow A \leftrightarrow PPM$	5.7	3.88
$CC \leftrightarrow APPM$	11.6	4.88

We see the best performance with $CC \leftrightarrow APPM$. We see approximately 1-dB loss when the accumulate and PPM modules are not coupled, and an additional 0.4-dB loss when the PPM SISO is not included in iterations.

X. Conclusions

Consider the design of coded modulation for an average power constrained optical channel. The channel operates efficiently with a large peak-to-average power ratio, which we choose to implement with PPM. The SCPPM code concatenates PPM with a recursive accumulator, a simple short constraint length convolutional code and bit interleaver. This is essentially convolutionally coded PPM, which, in itself, is not a new idea. The gains of SCPPM lie in incorporating the demodulation with iterative decoding of the convolutional code. With this twist, convolutionally coded PPM outperforms all currently known coded PPM architectures (for operating points considered).

Furthermore, the SCPPM encoder has a low-complexity implementation, and the decoder a moderate-complexity implementation. The bottle neck in decoding with large PPM orders lies in the iterative

demodulation of PPM. We have shown several methods to alleviate the hardware complexity of this step. We have also laid out explicit low-complexity, robust choices for an interleaver and stopping rule, other important components of the SCPPM code. This allows a high-speed implementation of the decoder in hardware. A detailed description of a hardware implementation appears in a companion article [24].

Acknowledgments

We benefited from discussions on hardware implementation with Michael Cheng and Michael Nakashima, on implementation of the stopping rule with Maged Barsoum, on permutation polynomial inverses with Matthew Klimesh, on anti-Gray mappings with Kenneth Andrews, on interleaver implementations and undetected error rates with Richard Barron, and on parameter selection with Don Boroson.

References

[1] A. D. Wyner, "Capacity and Error Exponent for the Direct Detection Photon Channel—Part I," *IEEE Transactions on Information Theory*, vol. 34, pp. 1449–1461, November 1988.

[2] R. M. Gagliardi, *Introduction to Communications Engineering*, New York: John Wiley & Sons, 1995.

[3] J. Hamkins and B. Moision, "Multipulse Pulse-Position Modulation on Discrete Memoryless Channels," *The Interplanetary Network Progress Report*, vol. 42-161, Jet Propulsion Laboratory, Pasadena, California, pp. 1–13, May 15, 2005. http://ipnpr/progress_report/42-161/161I.pdf

[4] R. G. Lipes, "Pulse-Position-Modulation Coding as Near-Optimum Utilization of Photon Counting Channel with Bandwidth and Power Constraints," *The Deep Space Network Progress Report 42-56, January and February 1980*, Jet Propulsion Laboratory, Pasadena, California, pp. 108–113, April 15, 1980. http://tmo.jpl.nasa.gov/tmo/progress_report2/42-56/56N.PDF

[5] R. J. McEliece, "Practical Codes for Photon Communication," *IEEE Transactions on Information Theory*, vol. IT-27, pp. 393–398, July 1981.

[6] B. Moision and J. Hamkins, "Deep-Space Optical Communications Downlink Budget: Modulation and Coding," *The Interplanetary Network Progress Report 42-154, April–June 2003*, Jet Propulsion Laboratory, Pasadena, California, pp. 1–28, August 15, 2003. http://ipnpr.jpl.nasa.gov/tmo/progress_report/42-154/154K.pdf

[7] V. Guruswami and M. Sudan, "Improved Decoding of Reed-Solomon and Algebraic-Geometry Codes," *IEEE Transactions on Information Theory*, vol. 45, pp. 1757–1767, September 1999.

[8] R. Koetter and A. Vardy, "Algebraic Soft-Decision Decoding of Reed-Solomon Codes," *IEEE Transactions on Information Theory*, vol. 49, pp. 2809–2825, November 2003.

[9] J. L. Massey, "Capacity, Cutoff Rate, and Coding for a Direct-Detection Optical Channel," *IEEE Transactions on Communications*, vol. COM-29, pp. 1615–1621, November 1981.

[10] G. S. Mecherle, *Maximized Data Rate Capability for Optical Communication Using Semiconductor Devices with Pulse Position Modulation*, Ph.D. thesis, University of Southern California, Los Angeles, May 1986.

[11] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "Serial Concatenation of Interleaved Codes: Performance Analysis, Design, and Iterative Decoding," *IEEE Transactions on Information Theory*, vol. 44, pp. 909–926, May 1998.

[12] L. Bahl, J. Cocke, F.Jelinek, and J.Raviv, "Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate," *IEEE Transactions on Information Theory*, vol. 20, pp. 284–287, March 1974.

[13] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "A Soft-Input Soft-Output Maximum A Posteriori (MAP) Module to Decode Parallel and Serial Concatenated Codes," *The Telecommunications and Data Acquisition Progress Report 42-127, July–September 1996*, Jet Propulsion Laboratory, Pasadena, California, pp. 1–20, November 15, 1996.
http://tmo.jpl.nasa.gov/tmo/progress_report/42-127/127H.pdf

[14] K. Kiasaleh, "Turbo-Coded Optical PPM Communication Systems," *Journal of Lightwave Technology*, vol. 16, pp. 18–26, January 1998.

[15] J. Hamkins, "Performance of Binary Turbo-Coded 256-ary Pulse-Position Modulation," *The Telecommunications and Mission Operations Progress Report 42-138, April–June 1999*, Jet Propulsion Laboratory, Pasadena, California, pp. 1–15, August 15, 1999.
http://tmo.jpl.nasa.gov/tmo/progress_report/42-138/138B.pdf

[16] M. Peleg and S. Shamai, "Efficient Communication over the Discrete-Time Memoryless Rayleigh Fading Channel with Turbo Coding/Decoding," *European Transactions on Telecommunications*, vol. 11, pp. 475–485, September–October 2000.

[17] A. Ashikhmin, G. Kramer, and S. ten Brink, "Extrinsic Information Transfer Functions: Model and Erasure Channel Properties," *IEEE Transactions on Information Theory*, vol. 50, pp. 2657–2673, November 2004.

[18] M. Barsoum and B. Moision, *Method and Apparatus for Fast Digital Turbo Decoding for Trellises with Parallel Edges*, JPL Technical Report 41234, Jet Propulsion Laboratory, Pasadena, California, August 2004.

[19] B. Moision and J. Hamkins, "Reduced Complexity Decoding of Coded Pulse-Position Modulation Using Partial Statistics," *The Interplanetary Network Progress Report*, vol. 42-161, Jet Propulsion Laboratory, Pasadena, California, pp. 1–20, May 15, 2005. http://ipnpr/progress_report/42-161/161O.pdf

[20] J. Sun and O. Y. Takeshita, "Interleavers for Turbo Codes Using Permutation Polynomials over Integer Rings," *IEEE Transactions on Information Theory*, vol. 51, pp. 101–119, January 2005.

[21] S. Lin and J. Daniel J. Costello, *Error Control Coding: Fundamentals and Applications*, New Jersey: Prentice Hall, 1983.

[22] R. Johannesson and K. S. Zigangirov, *Fundamentals of Convolutional Coding*, IEEE Series on Digital and Mobile Communication, Piscataway, New Jersey: IEEE Press, 1999.

[23] K. A. Witzke and C. Leung, "A Comparison of Some Error Detecting CRC Code Standards," *IEEE Transactions on Communications*, vol. 33, pp. 996–998, September 1985.

[24] M. K. Cheng, M. A. Nakashima, J. Hamkins, B. E. Moision, and M. Barsoum, "A Field-Programmable Gate Array Implementation of the Serially Concatenated Pulse-Position Modulation Decoder," *The Interplanetary Network Progress Report*, vol. 42, Jet Propulsion Laboratory, Pasadena, California, pp. 1–14, May 15, 2005. http://ipnpr/progress_report/42-161/161S.pdf

Appendix

Full Summary of SCPPM Decoding

Here we enumerate all the steps of the algorithm in the log domain for an example where the inner code is accumulate-PPM with $M = 64$, the outer code is the rate-1/2, 4-state (5,7) convolutional code, and the interleaver has length 15120 bits (these are the MLCD SCPPM code parameters). The outer code is terminated in the all-zeros state. The inner code is not terminated.

(1) (Inner SISO) Receive symbol log-likelihoods $\pi(\mathbf{c}_k; I)$ from the channel. On initialization, set the bit LLRs $\bar{p}(a_{k,i}; I) = 0$. On successive iterations, receive the bit LLRs $\bar{p}(a_{k,i}; I)$ from the outer code. Compute edge input symbol log-likelihoods (LRs)

$$\pi(\mathbf{a}_k; I) = \sum_{i=1}^{p} \frac{1}{2}(-1)^{\mathbf{a}_i} \bar{p}(a_{k,i}; I)$$

for $\mathbf{a} \in \{0, 1, \cdots, 63\}$ (meaning the corresponding 6-bit vectors $\mathbf{a}$) and $k = 1, \cdots, 2520$.

(2) Compute

$$\bar{\gamma}_k(e) = \pi_k\big(a(e); I\big) + \pi_k\big(c(e); I\big)$$

for each of the 128 edges in an inner code trellis stage and each stage $k = 1, \cdots, 2520$.

(3) Compute

$$\bar{\gamma}'_k(s, s') = \overset{*}{\max}_{e:i(e)=s,t(e)=s'} \left\{\bar{\gamma}_k(e)\right\}$$

for each pair of initial and terminal states $(s, s') \in \{(0,0), (0,1), (1,0), (1,1)\}$ and each $k = 1, \cdots, 2520$. These are 32-input $\overset{*}{\max}$ operations, and may be computed using the pipelined algorithm.

(4) Initialize

$$\bar{\alpha}_1(s) = \begin{cases} 0, & s = 0 \\ -\infty, & \text{otherwise} \end{cases}$$

$$\bar{\beta}_{2521}(s) = 0$$

Recursively compute

$$\bar{\alpha}_k(s) = \overset{*}{\max}\left(\bar{\alpha}_{k-1}(0) + \bar{\gamma}'_{k+1}(0,s), \bar{\alpha}_{k-1}(1) + \bar{\gamma}'_{k+1}(1,s)\right)$$

$$\bar{\beta}_k(s) = \overset{*}{\max}\left(\bar{\beta}_{k+1}(0) + \bar{\gamma}'_{k+1}(s,0), \bar{\beta}_{k+1}(1) + \bar{\gamma}'_{k+1}(s,1)\right)$$

for $s \in \{0,1\}$ and $k = 1, \cdots, 2520$. These are 2-input $\overset{*}{\max}$ operations.

(5) Compute

$$\bar{\lambda}_k(e) = \bar{\alpha}_{k-1}\big(i(e)\big) + \bar{\gamma}_k(e) + \bar{\beta}_k\big(t(e)\big)$$

for $e \in \mathcal{E}$ and $k = 1, \cdots, 2520$.

(6) Compute

$$\bar{p}(a_{k,i}; O) = \overset{*}{\max}_{e \in \mathcal{E}^A_{0,i}} \left\{\bar{\lambda}_k(e)\right\} - \overset{*}{\max}_{e \in \mathcal{E}^A_{1,i}} \left\{\bar{\lambda}_k(e)\right\} - \bar{p}(a_{k,i}; I)$$

for $k = 1, \cdots, 2520$, $i = 0, \cdots, 63$. These are 64-input $\overset{*}{\max}$ operations and may be computed using the pipelined algorithm.

(7) (De)-interleave the $\bar{p}(a_{k,i}; O)$. The output of the interleaver is the $\bar{p}(x_{k,i}; I)$.

(8) (Outer SISO) Receive bit LLRs $\bar{p}(x_{k,i}; I)$ from the inner code. Compute edge output symbol log-likelihoods (LRs)

$$\pi(\mathbf{x}_k; I) = \sum_{i=1}^{2} \frac{1}{2}(-1)^{\mathbf{x}_{k,i}} \bar{p}(x_{k,i}; I)$$

for each $\mathbf{x} \in \big\{(00), (01), (10), (11)\big\}$ and $k = 1, \cdots, 7560$.

(9) Set

$$\bar{\gamma}_k(e) = \pi\big(x(e); I\big)$$

for each of the 8 edges e in the outer code trellis and $k = 1, \cdots, 7560$. Note that $\pi\big(u(e); I\big) = 0$ for all iterations; hence it doesn't appear in the addition.

(10) Initialize

$$\bar{\alpha}_1(s) = \begin{cases} 0, & s = 0 \\ -\infty, & \text{otherwise} \end{cases}$$

$$\bar{\beta}_{7561}(s) = \begin{cases} 0, & s = 0 \\ -\infty, & \text{otherwise} \end{cases}$$

Recursively compute

$$\bar{\alpha}_k(s) = \overset{*}{\max}_{e:t(e)=s} \left\{ \bar{\alpha}_{k-1}\big(i(e)\big) + \bar{\gamma}_k(e) \right\}$$

$$\bar{\beta}_k(s) = \overset{*}{\max}_{e:i(e)=s} \left\{ \bar{\beta}_{k+1}\big(t(e)\big) + \bar{\gamma}_{k+1}(e) \right\}$$

for $s \in \{0, 1, 2, 3\}$ and $k = 1, \cdots, 7560$. These are 2-input $\overset{*}{\max}$ operations.

(11) Compute

$$\bar{\lambda}_k(e) = \bar{\alpha}_{k-1}\big(i(e)\big) + \bar{\gamma}_k(e) + \bar{\beta}_k\big(t(e)\big)$$

for each of the eight edges and $k = 1, \cdots, 7560$.

(12) Compute

$$\bar{p}(u_{k,i}; O) = \overset{*}{\max}_{e \in \mathcal{E}^U_{0,i}} \left\{ \bar{\lambda}_k(e) \right\} - \overset{*}{\max}_{e \in \mathcal{E}^U_{1,i}} \left\{ \bar{\lambda}_k(e) \right\}$$

$$\bar{p}(x_{k,j}; O) = \overset{*}{\max}_{e \in \mathcal{E}^X_{0,j}} \left\{ \bar{\lambda}_k(e) \right\} - \overset{*}{\max}_{e \in \mathcal{E}^X_{1,j}} \left\{ \bar{\lambda}_k(e) \right\} - \bar{p}_{k,j}(X; I)$$

for $k = 1, \cdots, 7560$, $i = 0$, and $j = 0, 1$.

(13) Check the stopping rule. If satisfied, terminate; otherwise continue.

(14) Interleave the $\bar{p}(x_{k,j}; O)$ and go to step (1). The output of the interleaver is the $\bar{p}(a_{k,j}; I)$.