Per-Survivor Iterative Timing Recovery for Coded Partial Response Channels

Piya Kovintavewat, John R. Barry, M. Fatih Erden,* and Erozan M. Kurtas*
Georgia Institute of Technology, Atlanta, GA 30332, USA.
*Seagate Technology, 1251 Waterfront Place, Pittsburgh, PA 15222, USA.

***Abstract*— We propose a new iterative timing recovery scheme based on per-survivor processing that jointly performs timing recovery and turbo equalization on partial response channels with error-correction codes. The scheme embeds the timing recovery process inside the Bahl, Cocke, Jelinek, and Raviv (BCJR) equalizer using per-survivor processing. This per-survivor BCJR equalizer then iteratively exchanges soft information with an error-correction decoder. Results indicate that the proposed scheme yields a better performance than a conventional receiver that performs timing recovery and turbo equalization separately, and also the iterative timing recovery scheme proposed in [1], especially when the channel encounters severe timing jitter noise. We also present evidence that suggests that the proposed scheme can correct a cycle slip much more efficiently than the others.**

***Index Terms*— Iterative timing recovery, per-survivor processing (PSP), synchronization.**

I. Introduction

THE process of synchronizing the sampler with the received analog signal is known as *timing recovery*. The quality of synchronization has a dominant impact on overall performance. The large coding gains of iterative error-correction codes (ECCs) allow reliable operation at low signal-to-noise ratio (SNR). This means that timing recovery must also function at low SNR. A conventional receiver performs timing recovery and error-correction decoding separately. Specifically, conventional timing recovery ignores the presence of ECCs. Thus, it fails to work properly at low SNR.

Theoretically, joint maximum-likelihood (ML) estimation of timing offsets and message bits, which will jointly perform timing recovery, equalization, and error-correction decoding, is a preferred method of synchronization. However, its complexity is huge. A solution based on the expectation-maximization (EM) algorithm is also complex [2]. Fortunately, a solution to this problem with complexity comparable to the conventional receiver has been proposed by Nayak, Barry, and McLaughlin [1], which will be referred to as the *NBM scheme*. It is realized by embedding the timing recovery step inside the turbo equalizer [3] so as to perform those three tasks jointly. Nonetheless, this scheme requires a large number of turbo iterations to provide a good performance even with a cycle slip [4] detection and correction algorithm as used in [1], especially when the timing jitter is large.

Per-survivor processing (PSP) [5] is a technique for jointly estimating the data sequence and unknown parameters, such as the channel coefficients, the carrier phase, and so forth. It has been employed in many applications, including channel identification, adaptive ML sequence detection, and phase/carrier recovery [5]-[7]. In [8], we applied PSP to develop the *PSP-based timing recovery* implemented based on a Viterbi algorithm [9], which performs timing recovery and ML equalization jointly.

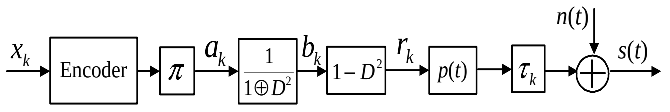

Fig. 1. Data encoding with a PR-IV channel model.

In this paper, we apply the per-survivor concept to the BCJR algorithm [10], resulting in a per-survivor BCJR equalizer denoted as “PSP-BCJR.” We also propose *per-survivor iterative timing recovery*, which iteratively exchanges soft information between PSP-BCJR and a soft-in soft-out (SISO) decoder. Although each iteration of per-survivor iterative timing recovery has high complexity, it can automatically correct a cycle slip much more efficiently than the NBM scheme. In other words, per-survivor iterative timing recovery requires fewer turbo iterations than the NBM scheme to yield good performance.

The PSP-BCJR module can be considered as a special case of the so-called *adaptive SISO* module developed by Anastasopoulos and Chugg [11]. In [11], the exact expressions for the soft metrics in the presence of parametric uncertainty modeled as a Gauss-Markov process were derived, but the complexity is huge. Suboptimal solutions have also been proposed in [11], and used in the trellis-coded modulation (TCM) in interleaved frequency-selective fading channels [11] and the turbo-coded systems with carrier phase tracking [12]. Nevertheless, the application of timing recovery for the intersymbol interference (ISI) channel with time-varying timing offsets has not been addressed and investigated.

This paper is organized as follows. After explaining our channel model in Section II, we propose and describe PSP-BCJR in Section III. Section IV compares the performance of different iterative timing recovery schemes. Finally, Section V concludes the paper.

II. Channel Description

We consider the coded partial response (PR) channel model shown in Fig. 1. The message bits $\{x_k\}$ are encoded by a serial concatenation of an error-correction encoder, an s-random interleaver [13] (i.e., the π block), and a $1/(1 \oplus D^2)$

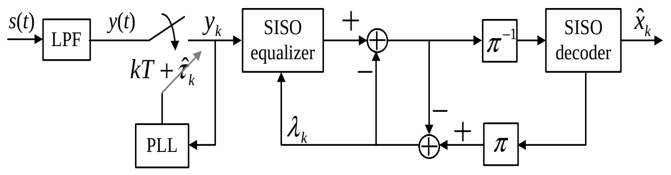

Fig. 2. Conventional receiver architecture.

precoder. The readback signal, $s(t)$, can then be written as

$$s(t)=\sum_{k=0}^{L-1} b_k h(t-kT-\tau_k)+n(t), \qquad (1)$$

where $\{b_k\}$ are the precoded bits of length L with bit period T, $h(t)=p(t)-p(t-2T)$ is a PR-IV pulse, $p(t)=\sin(\pi t/T)/(\pi t/T)$ is a 0% excess bandwidth pulse, and $n(t)$ is additive white Gaussian noise (AWGN) with two-sided power spectral density $N_0/2$. We model τ_k as a random walk [6] according to $\tau_{k+1}=\tau_k+\mathcal{N}(0,\sigma_w^2)$, where σ_w determines the severity of the timing jitter. The random walk model is chosen because of its simplicity and its ability to represent a variety of channels by changing only one parameter. We also assume perfect acquisition by setting $\tau_0=0$.

At the detector, the readback signal $s(t)$ is filtered by a low-pass filter (LPF), whose impulse response is $p(t)/T$ to eliminate out-of-band noise, and is sampled at time $kT+\hat{\tau}_k$, creating

$$y_k=y(kT+\hat{\tau}_k)=\sum_i b_i h(kT+\hat{\tau}_k-iT-\tau_i)+n_k, \qquad (2)$$

where $\hat{\tau}_k$ is the receiver's estimate of τ_k, and n_k is *i.i.d.* zero-mean Gaussian random variable with variance $\sigma_n^2=N_0/(2T)$.

Conventional timing recovery is based on a phase-locked-loop (PLL) [4]. Because perfect acquisition is assumed and our model has no frequency offset component, the sampling phase offset can then be updated by a first-order PLL according to [4]

$$\hat{\tau}_{k+1}=\hat{\tau}_k+\mu\{y_k\tilde{r}_{k-1}-y_{k-1}\tilde{r}_k\}, \qquad (3)$$

where μ is a PLL gain parameter, and $\tilde{r}_k$ is the k-th *soft estimate* of the *channel output* $r_k\in\{0,\pm 2\}$ given by [1]

$$\tilde{r}_k=E[r_k|y_k]=\frac{2\sinh(2y_k/\sigma_n^2)}{\cosh(2y_k/\sigma_n^2)+e^{2/\sigma_n^2}}. \qquad (4)$$

The soft estimate provides a better performance than the *hard estimate* [1], which is obtained by a memoryless three-level quantization of y_k.

In the conventional receiver, conventional timing recovery is followed by a turbo equalizer [3] (see Fig. 2), which iteratively exchanges soft information between the SISO equalizer for the precoded PR-IV channel and the SISO decoder.

III. PSP-BCJR

As shown in (3), the performance of conventional timing recovery relies on the decision $\tilde{r}_k$ provided by its own symbol detector, which might yield an unreliable decision. To overcome this drawback, a reliable decision can be extracted by utilizing the already-given information inside the trellis structure [9]. Specifically, each state transition in the trellis uniquely specifies a corresponding symbol. Thus, at least one state transition in each trellis stage will correspond to a correct decision. Using that decision for the timing update operation will improve the performance of timing recovery. The idea of using the information available in the trellis to estimate other unknown parameters is known as PSP [5].

```
*(A-1) Initialize $\hat{\tau}_0(p)=0$ for $\forall p$
(A-2)  Initialize $[\alpha_0(0)\,\ldots\,\alpha_0(Q-1)]=[1\,0\ldots 0]$
(A-3)  For $k=0,1,\ldots,L+\nu-1$ [Forward recursion]
(A-4)     For $q=0,1,\ldots,Q-1$
*(A-5)       $y_k(p)=y(kT+\hat{\tau}_k(p))$ for $\forall p$
(A-6)        $\gamma_k(p,q)=\exp\left\{-\frac{1}{2\sigma_n^2}|y_k(p)-\hat{r}(p,q)|^2+\frac{\hat{a}(p,q)\lambda_k}{2}\right\}$ for $\forall p$
(A-7)        $\alpha_{k+1}(q)=\sum_p \alpha_k(p)\gamma_k(p,q)$
*(A-8)       $\pi_{k+1}(q)=\arg\max_p\{\alpha_k(p)\gamma_k(p,q)\}$
*(A-9)       $\hat{\tau}_{k+1}(q)=\hat{\tau}_k(\pi_{k+1}(q))+\mu\{y_k(\pi_{k+1}(q))\hat{r}(\pi_k(\pi_{k+1}(q)),\pi_{k+1}(q))$
                 $-\,y_{k-1}(\pi_k(\pi_{k+1}(q)))\hat{r}(\pi_{k+1}(q),q)\}$
(A-10)    End
(A-11) End
*(A-12) Initialize $\hat{\tau}^b_{L+\nu}(p)=\hat{\tau}_{L+\nu}(p)$ for $\forall p$
(A-13) Initialize $[\beta_{L+\nu}(0)\,\ldots\,\beta_{L+\nu}(Q-1)]=[1\,0\ldots 0]$
(A-14) For $k=L+\nu-1,L+\nu-2,\ldots,0$ [Backward recursion]
(A-15)    For $p=0,1,\ldots,Q-1$
*(A-16)      $y^b_k(q)=y(kT+\hat{\tau}^b_{k+1}(q))$ for $\forall q$
(A-17)       $\gamma^b_k(p,q)=\exp\left\{-\frac{1}{2\sigma_n^2}|y^b_k(q)-\hat{r}(p,q)|^2+\frac{\hat{a}(p,q)\lambda_k}{2}\right\}$ for $\forall q$
(A-18)       $\beta_k(p)=\sum_q \gamma^b_k(p,q)\beta_{k+1}(q)$
*(A-19)      $\pi^b_k(p)=\arg\max_q\{\gamma^b_k(p,q)\beta_{k+1}(q)\}$
*(A-20)      $\hat{\tau}^b_k(p)=\hat{\tau}^b_{k+1}(\pi^b_k(p))+\mu\{y^b_k(\pi^b_k(p))\hat{r}(\pi^b_k(p),\pi^b_{k+1}(\pi^b_k(p)))$
                 $-\,y^b_{k+1}(\pi^b_{k+1}(\pi^b_k(p)))\hat{r}(p,\pi^b_k(p))\}$
*(A-21)      $\hat{\tau}^b_k(p)=\{\hat{\tau}^b_k(p)+\hat{\tau}_k(p)\}/2$  if $|\hat{\tau}^b_k(p)-\hat{\tau}_k(p)|>\Delta$
(A-22)    End
(A-23)    $\lambda^p_k=\log\left\{\frac{\sum_{\{p,q:\ \hat{a}(p,q)=+1\}}\alpha_k(p)\gamma^b_k(p,q)\beta_{k+1}(q)}{\sum_{\{p,q:\ \hat{a}(p,q)=-1\}}\alpha_k(p)\gamma^b_k(p,q)\beta_{k+1}(q)}\right\}$
(A-24) End
```

Fig. 3. PSP-BCJR algorithm, where the lines beginning with * are the additional steps beyond the conventional BCJR algorithm.

With PSP, we develop PSP-BCJR by embedding the timing recovery process inside the BCJR equalizer so as to perform timing recovery and equalization jointly. Fig. 3 shows the PSP-BCJR algorithm, where the lines beginning with * are the additional steps beyond the conventional BCJR algorithm. The key idea is that each node in the trellis has its own sampling phase offset. Thus, the branch metric is calculated based on the sampling phase offset of the starting state. Furthermore, we propose to update the timing estimate at each state based on the incoming branch that contributes the most to the state

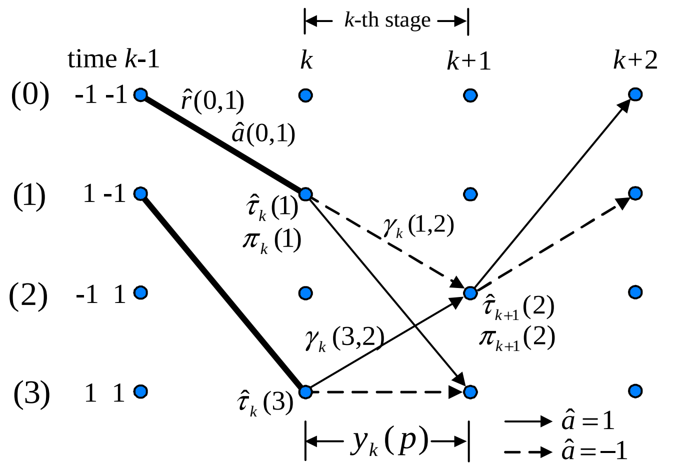

Fig. 4. The trellis structure demonstrating how PSP-BCJR performs during forward recursion.

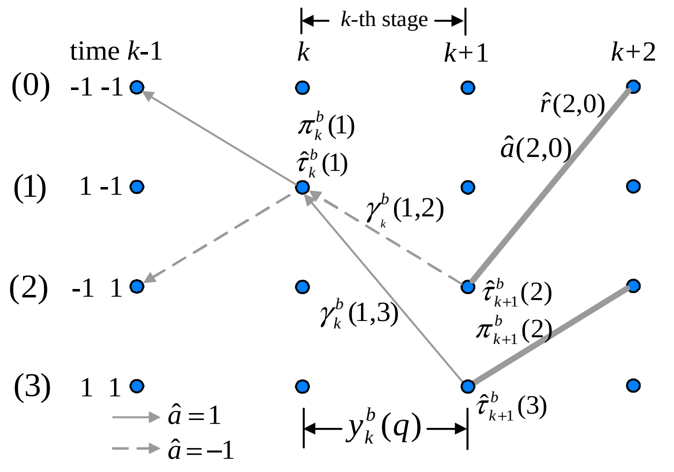

Fig. 5. The trellis structure illustrating how PSP-BCJR performs during backward recursion.

information. The detail on how PSP-BCJR performs during forward and backward recursions can be explained as follows.

A. Forward Recursion

Consider the PR-IV trellis structure shown in Fig. 4. Let $\Psi_k = \{b_{k-1}\, b_{k-2}\}$ denote the state at time k. There are $Q = 2^\nu = 4$ states in this trellis, labeled as state 0 to state 3, where $\nu = 2$ is the precoded PR-IV channel memory. Let (p, q) be the *state transition* from state p to state q, and let $\pi_k(p)$ denote a *predecessor* for state p at time k, defined as the starting state associated with the *best* state transition leading to state p at time k. We define $\hat{\tau}_k(p)$ as the k-th *forward* sampling phase offset for state p at time k, which is used to sample $y(t)$ at time k for the state transition emanating from state p at time k, e.g., $y_k(p) = y(kT + \hat{\tau}_k(p))$, where $y_k(p)$ is the k-th sampler output for state p at time k.

Consider the k-th stage of the trellis. There are two state transitions arriving at state 2 at time $k+1$, i.e., $(1, 2)$ and $(3, 2)$. First, we sample $y(t)$ using $\hat{\tau}_k(1)$ and $\hat{\tau}_k(3)$ to obtain $y_k(1)$ and $y_k(3)$, respectively. Next, we compute the metrics $\gamma_k(1, 2)$ and $\gamma_k(3, 2)$ based on (A-6), where $\hat{a}(p, q)$ is the interleaved bit (or the precoder input bit) associated with (p, q), and λ_k is the *a priori* log-likelihood ratio (LLR) of a_k. Then, the state information $\alpha_{k+1}(2)$ is updated according to (A-7).

The starting state associated with the best state transition leading to state 2 at time $k + 1$ is chosen according to (A-8). Suppose $(1, 2)$ is the best state transition leading to state 2 at time $k + 1$ so that $\pi_{k+1}(2) = 1$. We update the next forward sampling phase offset, $\hat{\tau}_{k+1}(2)$, based on (A-9). This $\hat{\tau}_{k+1}(2)$ will be used to sample $y(t)$ at time $k+1$ for the state transitions emanating from state 2 at time $k + 1$.

B. Backward Recursion

The backward timing update operation serves as refining the samples $\{y_k\}$ so as to improve the quality of the branch metrics. To explain how it works, we introduce the *backward transition* represented by the gray arrows as shown in Fig. 5. We define $\hat{\tau}^b_{k+1}(q)$ as the k-th *backward* sampling phase offset for state q at time $k + 1$, which is used to sample $y(t)$ at time k during backward recursion, e.g., $y^b_k(q) = y(kT + \hat{\tau}^b_{k+1}(q))$. Let $\pi^b_{k+1}(q)$ be a *successor* for state q at time $k+1$, defined as the starting state associated with the *best* backward transition leading to state q at time $k + 1$.

Consider the backward transition at the k-th stage. There are two backward transitions arriving at state 1 at time k, which correspond to $(1, 2)$ and $(1, 3)$. We first sample $y(t)$ using $\hat{\tau}^b_{k+1}(2)$ and $\hat{\tau}^b_{k+1}(3)$ to obtain $y^b_k(2)$ and $y^b_k(3)$, respectively. Then, we compute the metrics $\gamma^b_k(1, 2)$ and $\gamma^b_k(1, 3)$ based on (A-17), and update the state information $\beta_k(1)$ using (A-18). Similarly, the starting state associated with the best backward transition leading to state 1 at time k is selected according to (A-19). Suppose $(1, 2)$ corresponds to the best backward transition leading to state 1 at time k so that $\pi^b_k(1) = 2$. The next backward sampling phase offset, $\hat{\tau}^b_k(1)$, is updated by (A-20).

To avoid a cycle slip [4] when $\hat{\tau}^b_k(1)$ starts deviating from $\hat{\tau}_k(1)$, we propose a simple remedy by averaging the backward sampling phase offset according to (A-21), where Δ is the threshold that allows the backward sampling phase offset to deviate from the forward one. In this paper, we set $\Delta = 0.1T$ because we want to keep $\{\hat{\tau}^b_k\}$ close to $\{\hat{\tau}_k\}$. This $\hat{\tau}^b_k(1)$ will be used to sample $y(t)$ at time $k - 1$ for the backward transitions emanating from state 1 at time k. Finally, the *a posteriori* LLR of a_k, λ^p_k, is computed based on (A-23).

C. Complexity of PSP-BCJR

Apparently, for the precoded PR-IV channel, PSP-BCJR requires eight PLLs, i.e., one PLL for each state in one stage of the trellis during both forward and backward recursions. Furthermore, instead of storing the received analog signal $y(t)$, we could uniformly sample $y(t)$ at symbol rate to obtain a set of samples $\{y_k\}$. Then, we can only store this set of samples because the bandlimited nature of $y(t)$ makes them sufficient statistics. Consequently, PSP-BCJR can perform the timing update operation using $\{y_k\}$ and a digital interpolation filter, thus decreasing its complexity. In this paper, a 21-tap sinc interpolation filter is employed.

Beyond the conventional BCJR, PSP-BCJR also needs new storage requirements for (i) the forward/backward sampling phase offsets, (ii) the starting states, and (iii) the sampler outputs. However, only backward sampling phase offsets, starting states, and sampler outputs of the *current* and *previous* stages need to be stored, thus minimizing extra memory.

IV. Numerical Results

The per-survivor iterative timing recovery scheme is easily obtained by discarding the front-end PLL in Fig. 2 and replacing the BCJR equalizer with PSP-BCJR.

Consider a rate-8/9 system in which a block of 3636 message bits is encoded by a rate-1/2 recursive systematic convolutional encoder with a generator polynomial $[1, \frac{1\oplus D\oplus D^3\oplus D^4}{1\oplus D\oplus D^4}]$, and then punctured to a block length of 4095 bits by retaining only every the eighth parity bit. The punctured sequence passes through an s-random interleaver with $s = 16$ to obtain an interleaved sequence of a_k. Both the SISO equalizer and the SISO decoder are implemented based on BCJR. Note that the PLL gain parameters for different iterative timing recovery schemes were optimized based on minimizing the RMS timing error, $\sigma_\epsilon = \sqrt{E[(\tau_k - \hat{\tau}_k)^2]}$, at a per-bit SNR, E_b/N_0, of 5 dB. Each bit-error rate (BER) point was computed using as many data sectors as possible until at least 100 sectors in error were collected at the 100-th iteration.

Fig. 6 compares the BER performance of different schemes for the system with a moderate random walk parameter $\sigma_w/T = 0.5\%$, which implies a low probability of occurrence of a cycle slip. Note that the number inside the parenthesis in Fig. 6 indicates the total number of iterations used to generate each curve. The curve labeled "Perfect timing" represents the conventional receiver that uses $\hat{\tau}_k = \tau_k$ to sample $y(t)$. Also, the curve labeled "Genie-aided receiver" represents the conventional receiver whose PLL has access to all correct decisions, thus serving as a lower bound for a timing recovery scheme that is based on a PLL. The μ's for the conventional receiver, the NBM scheme, the per-survivor iterative timing recovery scheme, and the genie-aided receiver are 0.0053, 0.0053, 0.0028, and 0.0036, respectively. Apparently, per-survivor iterative timing recovery performs slightly better than the NBM scheme at the 50-th iteration, and both yield about a 0.45 dB gain at BER = 10^{-5} over the conventional receiver. In addition, per-survivor iterative timing recovery performs close to the genie-aided receiver and is only a 0.35 dB away from the system with perfect timing at BER = 10^{-5}.

Next, let us consider the system with a severe random walk parameter $\sigma_w/T = 1\%$, which implies a high probability of occurrence of a cycle slip. The μ's for the conventional receiver, the NBM scheme, the per-survivor iterative timing recovery scheme, and the genie-aided receiver are 0.0103, 0.0103, 0.006, and 0.007, respectively. Fig. 7 compares the BER performance of different schemes for the system with $\sigma_w/T = 1\%$. The NBM scheme still outperforms the conventional receiver; however, it seems to have an error floor at high BER. On the other hand, per-survivor iterative timing recovery provides a large performance gain over the NBM scheme and starts to have an error floor at low BER. Again, per-survivor iterative timing recovery still performs similar to the genie-aided receiver and loses approximately a 0.35 dB relative to the system with perfect timing at BER = 10^{-5}.

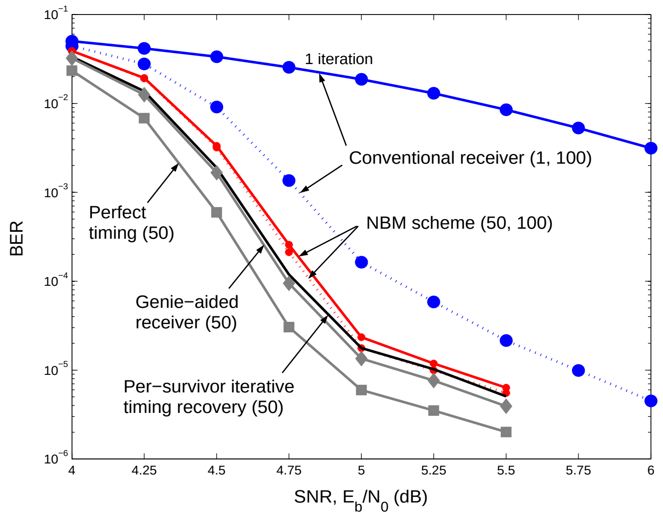

Fig. 6. Performance comparison with $\sigma_w/T = 0.5\%$.

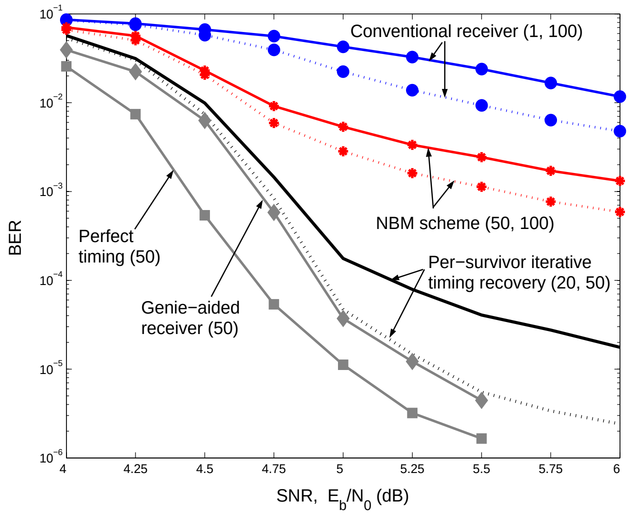

Fig. 7. Performance comparison with $\sigma_w/T = 1\%$.

One reason that per-survivor iterative timing recovery outperforms the NBM scheme when σ_w/T is large might be because the front-end PLL used in the NBM scheme does not work as well as PSP-based timing recovery [8]. Additionally, per-survivor iterative timing recovery can automatically correct a cycle slip (without a cycle slip detection and correction technique as employed in the NBM scheme [1]) much more efficiently than the NBM scheme. This property is illustrated in Fig. 8 by plotting the probability of an uncorrected cycle slip, where we declare a cycle slip when the actual timing offset and the estimated one are $0.75T$ apart from each other for more than 100 consecutive bit periods. Clearly, the NBM scheme requires a large number of iterations to correct a cycle slip as opposed to per-survivor iterative timing recovery. Note that the reason that the NBM scheme increases the probability of an uncorrected cycle slip at the first 10 iterations (see Fig. 8)

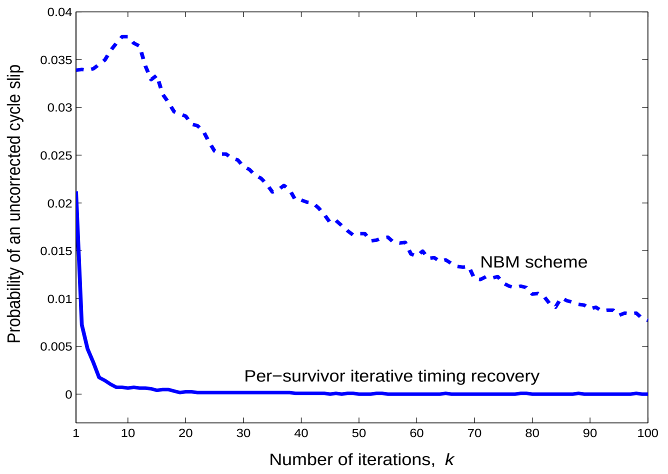

Fig. 8. Probability of an uncorrected cycle slip at SNR = 5 dB and $\sigma_w/T = 1\%$.

might be because it takes a few iterations for the cycle slip detection algorithm to recognize a cycle slip according to our cycle slip criterion.

Finally, Fig. 9 shows the timing waveform for two different sample packets, where the gray curve represents the actual τ sequence, and the number labeled on each curve denotes the number of iterations. As shown in Fig. 9(a), the NBM scheme takes more than 150 iterations to correct a cycle slip, whereas per-survivor iterative timing recovery takes only one iteration. This implies that the NBM scheme corrects a cycle slip slowly. Similarly, Fig. 9(b) shows that per-survivor iterative timing recovery takes about 5 iterations to correct a cycle slip, whereas the NBM scheme cannot correct it even with 50 iterations. This implies that per-survivor iterative timing recovery corrects a cycle slip quickly.

V. Conclusion

In this paper, we proposed a per-survivor version of the BCJR algorithm that performs timing recovery and equalization jointly. With a per-survivor BCJR equalizer, we proposed a per-survivor iterative timing recovery scheme to jointly perform timing recovery, equalization, and error-correction decoding, for coded partial response channels.

Simulation results have shown that per-survivor iterative timing recovery performs close to the genie-aided receiver, provided that the number of turbo iterations is large enough. Furthermore, it also performs better than the NBM scheme, especially when the timing error is large. This is because it can automatically correct a cycle slip much more efficiently than the NBM scheme. In other words, per-survivor iterative timing recovery requires fewer turbo iterations than the NBM scheme to yield good performance.

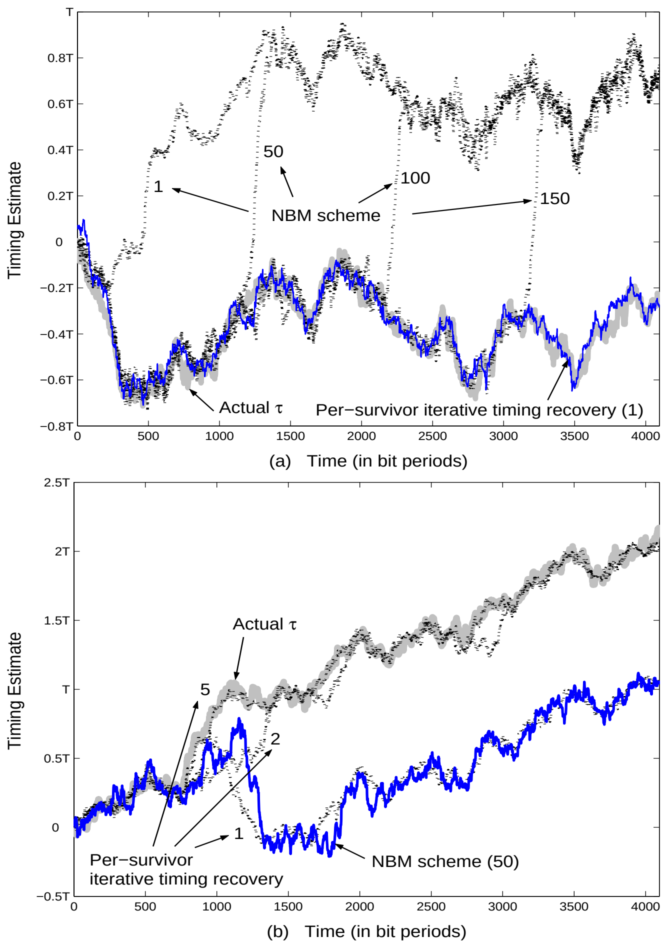

Fig. 9. Cycle slip correction from two different sample packets (SNR = 5 dB and $\sigma_w/T = 1\%$).

References

[1] A. R. Nayak, J. R. Barry, and S. W. McLaughlin, "Joint timing recovery and turbo equalization for coded partial response channels," *IEEE Trans. Magnetics*, vol. 38, no. 5, pp. 2295-2297, September 2003.

[2] C. N. Georghiades and D. L. Snyder, "The expectation-maximization algorithm for symbol unsynchronized sequence detection,"*IEEE Trans. Commun.*, vol. 39, pp. 54-61, January 1991.

[3] T. Souvignier, A. Friedmann, M. Oberg, P. Siegel, R. Swanson, and J. Wolf, "Turbo decoding for PR4: parallel vs. serial concatenation," in *Proc. of ICC'99*, vol. 3, pp. 1638-1642, 1999.

[4] J. W. M. Bergmans, *Digital baseband transmission and recording.* Boston, Massachusetts: Kluwer Academic Publishers, 1996.

[5] R. Raheli, A. Polydoros, and C. K. Tzou, "Per-survivor processing: a general approach to MLSE in uncertain environments," *IEEE Trans. Commun.*, vol. 43, no. 2, pp. 354-364, Feb/Mar/Apr 1995.

[6] A. N. D' Andrea, U. Mengali, and G. M. Vitteta, "Approximate ML decoding of coded PSK with no explicit carrier phase reference," *IEEE Trans. Commun.*, vol. 42, pp. 1033-1039, Feb/Mar/Apr 1994.

[7] A. V. Coralli, P. Salmi, S. Cioni, G. E. Corazza, and A. Polydoros, "A performance review of PSP for joint phase/frequency and data estimation in future broadband satellite networks," *IEEE J. Selected Areas Commun.*, vol. 19, no. 12, pp. 2298-2309, December 2001.

[8] P. Kovintavewat, J. R. Barry, M. F. Erden, and E. Kurtas, "Per-survivor timing recovery for uncoded partial response channels," to appear in *Proc. of ICC'04*, Paris, June 20-24, 2004.

[9] G. D. Forney, "Maximum-likelihood sequence estimation of digital sequences in the presence of intersymbol interference," *IEEE Trans. Inform. Theory*, vol. IT-18, no. 3, pp. 363-378, May 1972.

[10] L. R. Bahl, J. Cocke, F. Jelinek and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. Inform Theory*, vol. IT-20, pp. 248-287, March 1974.

[11] A. Anastasopoulos and K. M. Chugg, "Adaptive soft-in soft-output algorithms for iterative detection with parametric uncertainty," *IEEE Trans. Commun.*, vol. 48, pp. 1638-1649, October 2000.

[12] A. Anastasopoulos and K. M. Chugg, "Adaptive iterative detection for phase tracking in turbo-coded systems," *IEEE Trans. Commun.*, vol. 49, pp. 2135-2144, December 2001.

[13] C. Heegard and S. B. Wicker, *Turbo Coding.* Boston, Massachusetts: Kluwer Academic Publishers, 1999.