A RECURSIVE ALGORITHM FOR PRUNED BIT-REVERSAL PERMUTATIONS

Mohammad M. Mansour

ECE Department
American University of Beirut
Beirut, Lebanon
Email: mmansour@aub.edu.lb

ABSTRACT

A fast recursive algorithm for pruned bit-reversal permutations is proposed. The algorithm is based on a computationally efficient scheme for evaluating a novel permutation statistic called permutation inliers that counts inlier addresses under pruning. This statistic is computed by evaluating a recursion using integer shift and add operations in logarithmic time complexity. Moreover, a parallel pruned interleaving algorithm based on computing multiple inliers in parallel is proposed. The advantages of the proposed algorithm are reduced latency and reduced memory requirements, which are describe in many signal processing and communication applications.

Index Terms— Bit-reversal permutations, pruned interleavers.

1. INTRODUCTION

In certain signal processing applications, it is often required to shuffle streaming data [1] into a particular order such as in signal transform (e.g., Cooley-Tukey fast Fourier [2, 3], discrete cosine [4], Hartley transforms [5]), matrix transposition [6, 7], and matrix decomposition algorithms [8]. These shufflings are based on a permutation which is typically data-independent. Examples include bit-reversal, Gray code, linear congruential, and sorting permutations. Such permuters are implemented using double-buffers or in some cases single buffers if the permutation satisfies certain properties, in particular, low permutation order. [1].

However, in many other communications applications, it is sometimes impossible to permute the streaming data before the whole stream is received even if its permutation order is low, due to the fact that the permutation is "address-dependent". Address generation in this case is the bottleneck. This necessitates double or multiple buffers in order to maintain a desired system processing throughput, which increases the memory requirements. An example of address-dependent permuters are *pruned interleavers*. Interleavers are used as an adjunct to coding for error correction [9, 10]. They are a subclass of permuters with carefully chosen permutations to break certain patterns in the input sequence, and strategically reposition symbols according to their relevance in protecting the overall sequence against errors. Examples include interleavers in turbo codes [11], channel interleavers in bit-interleaved coded modulation schemes [12], and carrier interleaving for diversity gain in multi-carrier wireless systems with frequency-selective fading and multiple-access interference [13].

A class of computationally efficient interleavers with simple address generation are *block* interleavers [14] of power-of-2 length $k = 2^n$. They are expressed in closed-form expression by $\rho : \mathbb{Z}_k \rightarrow \mathbb{Z}_k, \rho(j) = k_1 \cdot \pi_1(j \bmod k_1) + \pi_2\left(\left\lfloor \frac{j}{k_1} \right\rfloor\right)$, where $\pi_1 : \mathbb{Z}_{k_1} \rightarrow \mathbb{Z}_{k_1}$ and $\pi_2 : \mathbb{Z}_{k_2} \rightarrow \mathbb{Z}_{k_2}$ are basic permutations of lengths $k_1 = 2^{n_1}$ and $k_2 = 2^{n_2}$, respectively, and $k = k_1 k_2$. Here the k symbols are written row-wise into a $k_2 \times k_1$ array and read column-wise after permuting the rows by π_1 and the columns by π_2. Example permutations proposed in the literature or adopted in modern communications standards [15, 16, 17] include the bit-reversal permutation (BRP) $\pi(j) = \text{BRP}(j_{(2)})$ [16] which reverses the order of bits in $j_{(2)}$, and polynomial-based permutations $\pi(j) = f_m(j) \bmod k$ where $f_m(j)$ is a degree-m permutation polynomial [18].

Block interleavers are often required to support several codes with various codeword lengths (not just powers of 2) depending on the input data rate requirements (e.g. recent communication standards [15, 16, 17]). To accommodate for flexible block lengths β, interleaving is done using a mother interleaver of length $k = 2^n$, where n is the smallest integer such that $\beta \leq k$, such that outlier interleaved addresses greater than $\beta - 1$ are pruned away. However, this pruning operation makes the permutation address-dependent, and creates a serial bottleneck since interleaved addresses are now a function of the permutation as well as the number of pruned addresses. To parallelize this operation, it is essential to characterize the permutation structure with respect to the pruning operation in order to interleave an address without interleaving all its predecessors.

In this paper, we propose a recursive algorithm to speed up pruned interleavers based on bit-reversal permutations. The algorithm is based on an efficient scheme for deriving the so-called "inliers permutation statistic" which counts pruned inliers without traversing all predecessors. This algorithm has logarithmic time-complexity compared to linear complexity for a serially pruned bit-reversal interleaver (PBRI). Moreover, we use this algorithm to parallelize a serial PBRI and reduce latency by a desired parallelism factor. Another important advantage of the proposed algorithm is that it reduces memory buffering requirements and enables on-the-fly interleaving without first storing all the data, then waiting to derive all pruned addresses before shuffling the data. The inliers statistic can be generalized to other permutations, which is the main advantage over our earlier algorithm presented in [19].

2. A BASIC PRUNED PERMUTATION ALGORITHM

Consider the sequence of integers $[k] \triangleq \{0, 1, \cdots, k-1\}$, and let π be a permutation on $[k]$. Let $(j_{n-1} \cdots j_1 j_0)_2$ denote the binary representation of an integer $j \in [k]$, where $k = 2^n$, n is the number of bits, and $j_i \in \{0, 1\}$ for $i = 0, \cdots, n-1$. The bit-reversal of j is defined as $\pi_n(j) \triangleq (j_0 j_1 \cdots j_{n-1})_2 = \sum_{i=0}^{n-1} j_i 2^{n-i}$.

A bit-reversal interleaver (BRI) maps an n-bit number α into another n-bit number y such that $y = \pi_n(\alpha)$. The values taken by α and y range from 0 to $k-1$, where $k = 2^n$ is the size of

the interleaver. A *pruned* BRI (PBRI) maps an n-bit number $\alpha < \beta \leq k$ into another n-bit number $y < \beta$ according to the bit-reversal rule. The size of the pruned interleaver is β. Pruned interleavers are used when blocks of arbitrary lengths (other than powers-of-2) are needed. To interleave a block of size β, a mother interleaver whose size is the smallest power-of-2 that is $\geq \beta$ is selected and pruned. Hence, in the following, we assume that $k/2 < \beta < k$.

There are several ways to prune addresses from the mother interleaver. One method is to ignore positions beyond $\beta - 1$ in the permuted sequence, which we consider in this work (see also [20, 21]). Other methods prune addresses beyond $\beta - 1$ in the original sequence, or prune a mixture of addresses from both the original and pruned sequences [21]. Hence any address that maps to an address $\geq \beta$ is dropped and the next consecutive address is tried instead. To determine where an arbitrary address α gets mapped, a *serial* PBRI (S-PBRI) starts from $w = 0$ and maintains the number of invalid mappings δ (called the pruning gap) that have been skipped along the way. If $w + \delta$ maps to a valid address (i.e., $\pi_n(w + \delta) < \beta$), then w is incremented by 1. If $w + \delta$ maps to a invalid address (i.e., $\pi_n(w + \delta) \geq \beta$), δ is incremented by 1. These steps are repeated until w reaches α and $\pi_n(\alpha + \delta)$ is a valid address. Therefore, in a pruned bit-reversal interleaver, $\alpha \mapsto y = \pi_n(\alpha + \delta)$. Algorithm 1 shows the pseudo-code.

Algorithm 1 Serial PBRI Algorithm: $\mathbf{y} = \text{S-PBRI}(k, w_1, w_2, \beta, \delta)$

```
w ← w1
while w ≤ w2 do
    if π_n(w + δ) < β then
        y[w] ← π_n(w + δ)
        w ← w + 1
    else
        δ ← δ + 1
    end if
end while
```

3. PERMUTATION INLIERS STATISTIC

Permutation statistics have been used to study combinatorial properties of permutations. In [22], a new permutation statistic useful for analyzing pruned interleavers called *permutation inliers* was introduced. An integer $i \in [k]$ is called an (α, β)-*inlier* of π if $i < \alpha$ and $\pi(i) < \beta$. We denote by $\text{INL}_{\alpha,\beta}(\pi)$ the set consisting of all (α, β)-inliers of π,

$$\text{INL}_{\alpha,\beta}(\pi) \triangleq \{j \in [k] \mid j < \alpha, \pi(j) < \beta\},\ 0 < \alpha, \beta \leq k, \quad (1)$$

and by $\#\text{INL}_{\alpha,\beta}(\pi)$ the number of (α, β)-inliers of π. We call determining $\text{INL}_{\alpha,\beta}(\pi)$ for arbitrary π the *permutation inliers problem*. Similarly, an integer $i \in [k]$ is called an (α, β)-*outlier* if $i < \alpha$ and $\pi(i) \geq \beta$. $\text{OUL}_{\alpha,\beta}(\pi)$ denotes the set of all (α, β)-outliers, and $\#\text{OUL}_{\alpha,\beta}(\pi)$ their number:

$$\text{OUL}_{\alpha,\beta}(\pi) = [\alpha] - \text{INL}_{\alpha,\beta}(\pi), \quad (2)$$

where '$-$' is the set-difference operator. For example, the $(5, 7)$-inliers of the permutation $\pi = \left(\begin{smallmatrix} 0&1&2&3&4&5&6&7&8&9 \\ 3&1&7&2&5&8&6&4&0&9 \end{smallmatrix}\right)$ are $\text{INL}_{5,7}(\pi) = \{0, 1, 3, 4\}$, while the outliers are $\text{OUL}_{5,7}(\pi) = \{2\}$.

Surprisingly, determining $\text{INL}_{\alpha,\beta}(\pi)$ for arbitrary permutations has largely been unattempted before in the literature. In [23], a solution for linear permutation polynomials based on Dedekind sums [24, 25] was proposed. In [19] a combinatorial solution for bit-reversal permutations based on bit manipulations was presented. Here we present an alternative recursive solution to the problem for bit-reversal permutations based on an efficient evaluation of

$$\#\text{INL}_{\alpha,\beta}(\pi) = \sum_{j=0}^{k-1} \left\lfloor \frac{j - \alpha}{k} \right\rfloor \left\lfloor \frac{\pi(j) - \beta}{k} \right\rfloor, \quad (3)$$

which counts the desired inliers. The solution is based on evaluating sums analogous to the well-known Dedekind sums [24]. In [22], it was shown that this solution takes the form

$$\#\text{INL}_{\alpha,\beta} = \frac{\alpha\beta}{k} + \frac{1}{4k} T(k, \alpha, \beta) + K_{\text{INL}}(\alpha, \beta), \quad (4)$$

where $T(k, \alpha, \beta)$ is defined by the following recursion

$$T(k, \alpha, \beta) = \begin{cases} 2T(k/2, \alpha, (\beta - 1)/2) - 4\alpha - K_o, & \beta \text{ odd;} \\ 2T(k/2, \alpha, \beta/2) + K_e, & \beta \text{ even,} \end{cases} \quad (5)$$

and 0 if either $\alpha = 0$ or $\beta = 0$. The remaining quantities are given by: $K_o = k\Big(2\Big\lfloor \frac{\beta^* - \alpha}{k} \Big\rfloor - 2\lfloor \frac{2\alpha}{k} \rfloor - \delta\Big(\frac{\beta^* - \alpha}{k}\Big) + \delta\Big(\frac{\beta^*}{k}\Big) + \delta\big(\frac{2\alpha}{k}\big)\Big)$; $K_e = k\Big(2\Big\lfloor \frac{\beta^* - \alpha}{k} + \frac{1}{2} \Big\rfloor + 2\lfloor \frac{\alpha}{k} + \frac{1}{2} \rfloor - \delta\Big(\frac{\beta^* - \alpha}{k} + \frac{1}{2}\Big) - \delta\big(\frac{\alpha}{k} + \frac{1}{2}\big)\Big)$; $\beta^* = \pi_{n-1}((\beta - 1)/2)$ if β is odd; $\beta^* = \pi_{n-1}(\beta/2)$ if β is even; the function $\delta(x) = 1$ if x is an integer, and 0 otherwise; $K_{\text{INL}}(\alpha, \beta)$ is a constant that evaluates to either $0, \pm 1/4, 1/2, 3/4$ depending on α, β (details are omitted due to lack of space).

Note that since k is a power of 2, only integer shift and add operations are needed to compute (5) and (4), assuming the product of the constants $\alpha\beta$ is computed off-line. Eq. (4) evaluates recursively in at most $\log_2 k - 1$ steps using (5), where the arguments of $T(k, \alpha, \beta)$ are reduced until T reaches 0.

Example 1 *Let* $n = 32, k = 2^n = 2^{32}, \alpha = 2^{16} - 1, \beta = 2^{16} + 1$. *Then* $\alpha\beta/k = (2^{32} - 1)/2^{32}$ *and* K_{INL} *evaluates to* $3/4$ *in this case. Using* (5)*, we have* $c^* = \pi_{31}(2^{15}) = 2^{15}$ *and* $T(2^{32}, 2^{16} - 1, 2^{16} + 1) = 2T(2^{31}, 2^{16} - 1, 2^{15}) + 2^{33} - 2^{18} - 2^2$. *Next we have* $T(2^{31}, 2^{16} - 1, 2^{15}) = 2T(2^{30}, 2^{16} - 1, 2^{14})$. *These steps are repeated using* (5)*, resulting in* $T(2^{32}, 2^{16} - 1, 2^{16} + 1) = 4294967300 = 2^{33} + 2^2$. *Therefore, using* (4) *we have* $\#INL_{2^{16}-1, 2^{16}+1} = 2$.

4. A RECURSIVE PRUNED PERMUTATION ALGORITHM

The time complexity to determine δ using the S-PBRI algorithm is $\mathcal{O}(k)$. We apply the permutation inliers problem to evaluate the pruning gap δ more efficiently in order to speed up the pruning process in bit-reversal interleavers. Using the inliers problem formulation, δ is simply the minimum non-negative integer to be added to α such that $\text{INL}_{\alpha+\delta,\beta}$ has exactly α inliers: $\min \delta \geq 0$ such that $\#\text{INL}_{\alpha+\delta,\beta} = \alpha$.

Out of the first α addresses, there are $\#\text{OUL}_{\alpha,\beta}$ outliers $\geq \beta$. Hence $\delta \geq \#\text{OUL}_{\alpha,\beta}$. Next consider the expanded interval of addresses $\alpha_1 = \alpha + \#\text{OUL}_{\alpha,\beta}$. This set contains $\#\text{OUL}_{\alpha_1,\beta}$ outliers. Hence again $\delta \geq \#\text{OUL}_{\alpha_1,\beta}$. This process is repeated by expanding the interval into $\alpha_2 = \alpha + \#\text{OUL}_{\alpha_1,\beta}$ and determining the corresponding number of outliers. The process terminates when $\#\text{OUL}_{\alpha_t,\beta} = \#\text{OUL}_{\alpha_{t-1},\beta}$ at some step t when there are no more outliers, and hence $\delta = \#\text{OUL}_{\alpha_t,\beta}$. This process is illustrated in Fig. 1). Algorithm 2 lists the pseudo-code.

Example 2 *Let* $n = 32, k = 2^n = 2^{32}, \alpha = 2^{12}, \beta = 2^{31} + 10$. *Applying the MI algorithm, we have* $\delta_1 = \#OUL_{2^{12}, 2^{31}+10} = 2047$ *using* (2)*,* (4)*. Next we expand* α *to* $\alpha + 2047$ *and recompute* $\delta_2 =$

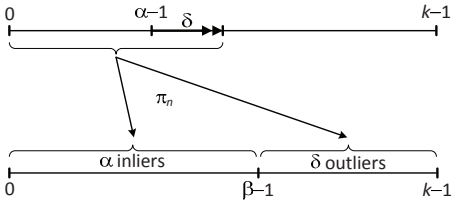

Fig. 1. The smallest interval of addresses $\alpha + \delta$ that has exactly α inliers and δ outliers with respect to β when mapped by π_n.

$\#OUL_{2^{12}+2047,2^{31}+10} = 3070$. *Similarly at step 3 we have* $\delta_3 = \#OUL_{2^{12}+3070,2^{31}+10} = 3582$. *The operations are repeated until* $t = 12$ *with* $\delta_{12} = \#OUL_{2^{12}+4093,2^{31}+10} = 4093$.

The gap δ in Algorithm 2 is initialized to zero for simplicity. There are ways to initialize δ. For example, it can be shown [22] that δ satisfies the following bound:

$$\alpha(k/\beta - 1) + W_l k/\beta \leq \delta \leq \alpha(k/\beta - 1) + W_u k/\beta \qquad (6)$$

where W_l and W_u are constants. δ can be initialized with the lower bound, or with the upper bound but the gap in Algorithm 2 must be reduced instead of expanded every iteration.

Algorithm 2 Minimal Inliers (MI) Algorithm: $\delta = \text{MI}(k, \alpha, \beta)$

$t \leftarrow 0$
$\delta_0 \leftarrow 0$
repeat
  $\delta_{t+1} \leftarrow \#\text{OUL}_{\alpha+\delta_t,\beta}$
  $t \leftarrow t + 1$
until $\delta_t = \delta_{t-1}$
$\delta \leftarrow \delta_t$

The convergence rate of the MI algorithm is $1 - \beta/k$ as Theorem 1 states. The proof is based on the bounds given in (6). The details are omitted due to lack of space.

Theorem 1 (Rate of Convergence) *The minimal inliers algorithm converges at a rate* $\mu = 1 - \beta/k$.

Using the MI algorithm a *parallel* PBRI of length β with a parallelism factor of p over the S-PBRI can be designed using p (or $p + 1$ if $\beta \bmod p \neq 0$) S-PBRI's as shown in Algorithm 3. The S-PBRI's are initialized with their respective δ's using the MI algorithm.

Algorithm 3 Parallel PBRI Algorithm: $\mathbf{y} = \text{P-PBRI}(k, p, \beta)$

for all $i = 0 \rightarrow p - 1$ **do**
  $\delta_i \leftarrow \text{MI}(k, i\lfloor\beta/p\rfloor, \beta)$
  $\mathbf{y}[i\lfloor\beta/p\rfloor : (i+1)\lfloor\beta/p\rfloor - 1] \leftarrow \text{S-PBRI}(k, i\lfloor\beta/p\rfloor, (i+1)\lfloor\beta/p\rfloor - 1, \beta, \delta_i)$
end for
if $\beta \bmod p > 0$ **then**
  $\delta_p \leftarrow \text{MI}(k, p\lfloor\beta/p\rfloor, \beta)$
  $\mathbf{y}[p\lfloor\beta/p\rfloor : \beta - 1] \leftarrow \text{S-PBRI}(k, p\lfloor\beta/p\rfloor, \beta - 1, \beta, \delta_{p-1})$
end if

5. SIMULATION RESULTS AND CONCLUSIONS

Figure 2 compares the interleaving time of the S-PBRI algorithm and the proposed P-PBRI algorithm as a function of interleaver size, for two values of p. As shown from the plots, the time to interleave degrades significantly for the S-PBRI algorithm as k increases. The P-PBRI algorithm attains a speed up improvement of slightly more than p over the S-PBRI algorithm.

Figure 3 illustrates the lower and upper bounds of the pruning gap δ. The plot demonstrates the tightness of the bounds given in (6).

Figure 4 shows the convergence rate of the MI algorithm. The asymptotic bounds designate the lower and upper bounds of δ, while the solid line shows the actual values of δ as the algorithm converges. The rate of convergence depends on the ratio of β/k, which is the ratio of the pruned interleaver size to the mother interleaver size.

The plots demonstrate the advantages of the proposed MI and P-PBRI algorithms in speeding up the interleaving process when pruning is employed. They eliminate the serial bottleneck of the S-PBRI algorithm. The P-PBRI algorithm can be efficiently utilized to design parallel pruned channel and turbo interleavers employed in modern communications standards such as [15, 16, 17], and hence reduce interleaving latency on the transmitter side and deinterleaving latency on the receiver side.

The proposed scheme to accelerate the interleaving speed of pruned interleavers based on the minimum inliers statistic can be generalized to other permutations structures beyond bit-reversal permutations. An interesting permutation is the one currently employed in LTE [15] which is based on quadratic permutation polynomials (QPP) [18]. The recursive solution in (4) needs to be adapted to QPP, which is currently work in progress.

6. REFERENCES

[1] A. Parsons, "The symmetric group in data permutation, with applications to high-bandwidth pipelined fft architectures," *IEEE Sig. Proc. Let.*, vol. 16, no. 6, pp. 477–480, Jun. 2009.

[2] J. Cooley and J. Tukey, "An algorithm for the machine calculation of complex fourier series," *Mathematics of Computation*, vol. 19, no. 90, pp. 297–301, 1965.

[3] C. S. Burrus, "Unscrambling for fast DFT algorithms," *IEEE Trans. on Acoustics, Speech and Signal Proc.*, vol. 36, no. 7, pp. 1086–1087, Jul. 1988.

[4] A.N. Skodras and A.G. Constantinides, "Efficient input-reordering algorithms for fast DCT," *Elec. Let.*, vol. 27, no. 21, pp. 1973–1975, Oct. 1991.

[5] D. Evans, "An improved digit-reversal permutation algorithm for the fast Fourier and Hartley transforms," *IEEE Trans. on Acoustics, Speech and Signal Proc.*, vol. 35, no. 8, pp. 1120–1125, Aug. 1987.

[6] K. Kim, "Shuffle memory system," in *Int. Symp. on Parallel and Distributed Proc.*, Apr. 1999, pp. 268–272.

[7] M. R. Portnoff, "An efficient parallel-processing method for transposing large matrices in place," *IEEE Trans. on Image Proc.*, vol. 8, no. 9, pp. 1265–1275, Sep. 1999.

[8] I. Verbauwhede, F. Catthoor, J. Vandewalle, and H. Man, "In-place memory management of algebraic algorithms on application specific ICs," *The Journal of VLSI Signal Proc.*, vol. 3, pp. 193–200, 1991.

[9] J. Ramsey, "Realization of optimum interleavers," *IEEE Trans. on Info. Theory*, vol. 16, no. 3, pp. 338–345, May 1970.

[10] Jr. Forney, G., "Burst-correcting codes for the classic bursty channel," *IEEE Trans. on Comm. Tech.*, vol. 19, no. 5, pp. 772–781, Oct. 1971.

[11] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo codes," in *Proc. IEEE Conf. on Comm.*, 1993, pp. 1064–1070.

[12] E. Zehavi, "8-PSK trellis codes for a rayleigh channel," *IEEE Trans. on Comm.*, vol. 40, no. 5, pp. 873–884, May 1992.

[13] J. Bingham, "Multicarrier modulation for data transmission: an idea whose time has come," *IEEE Comm. Mag.*, vol. 28, no. 5, pp. 5–14, May 1990.

[14] R. Garello, G. Montorsi, S. Benedetto, and G. Cancellieri, "Interleaver properties and their applications to the trellis complexity analysis of turbo codes," *IEEE Trans. on Comm.*, vol. 49, no. 5, pp. 793 –807, May 2001.

[15] 3GPP TS 36.212, "Modulation and channel coding," Tech. Rep., Evolved Universal Terrestrial Radio Access (EUTRA), (Release 8).

[16] 3rd Generation Partnership Project 2 (3GPP2), "IEEE standard for local and metropolitan area networks part 20: Air interface for mobile broadband wireless access systems supporting vehicular mobility - physical and media access control layer specification," *IEEE Std 802.20-2008*, 2008.

[17] "IEEE standard for local and metropolitan area networks part 16: Air interface for broadband wireless access systems," *IEEE Std 802.16-2009 (Revision of IEEE Std 802.16-2004)*, 2009.

[18] J. Sun and O.Y. Takeshita, "Interleavers for turbo codes using permutation polynomials over integer rings," *IEEE Trans. on Info. Theory*, vol. 51, no. 1, pp. 101–119, Jan 2005.

[19] M. M. Mansour, "A parallel pruned bit-reversal interleaver," *IEEE Trans. on VLSI Systems*, vol. 17, no. 8, pp. 1147–1151, Aug. 2009.

[20] M. Ferrari, F. Scalise, and S. Bellini, "Prunable S-random interleavers," in *Proc. IEEE Conf. on Comm., New York, USA*, Oct 2002, vol. 3, pp. 1711–1715.

[21] L. Dinoi and S. Benedetto, "Design of fast-prunable s-random interleavers," *IEEE Trans. on Wireless Comm.*, vol. 4, no. 5, pp. 2540–2548, Sep 2005.

[22] M. M. Mansour, "A mathematical characterization of bit-reversal permutations," 2011, In preparation to be submitted to IEEE Trans. on Comm.

[23] M. M. Mansour, "Parallel lookahead algorithms for pruned interleavers," *IEEE Trans. on Comm.*, vol. 57, no. 11, pp. 3188–3194, Nov. 2009.

[24] D. Knuth, *The Art of Computer Programming - Seminumerical Algorithms*, Addison-Wesley, Reading, MA, 3rd edition, 1998.

[25] U. Dieter and J. Ahrens, "An exact determination of serial correlations of pseudo-random numbers," *Numerische Mathematik*, vol. 17, pp. 101–123, 1971.

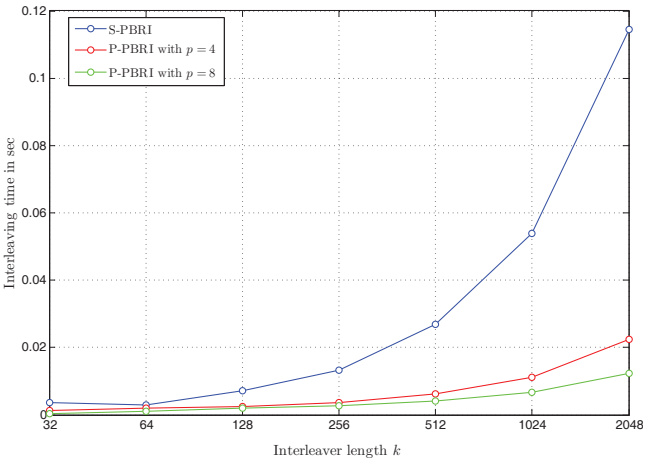

Fig. 2. Comparison of convergence time of S-PBRI and P-PBRI algorithms versus interleaver length k for $p = 4$ and $p = 8$.

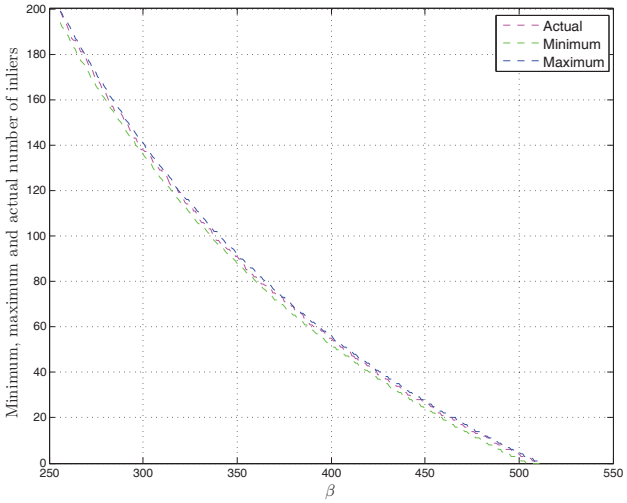

Fig. 3. Lower and upper bounds of the pruning gap δ

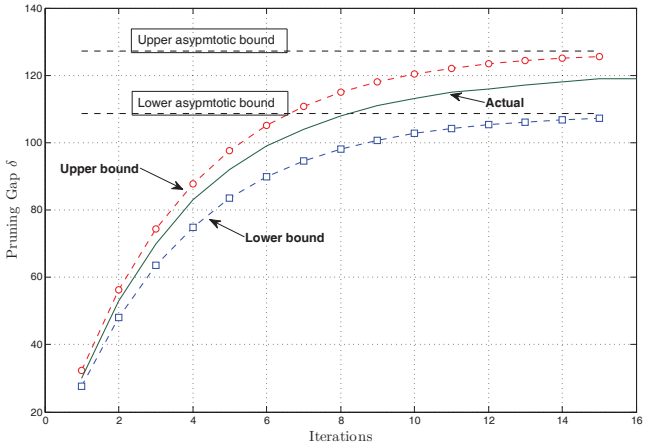

Fig. 4. Convergence rate of the MI algorithm.