TURBO LIKE DECODING OF LDPC CODES

Jin Lu and José M. F. Moura
Department of Electrical and Computer Engineering, Carnegie Mellon Univerity,
5000 Forbes Ave., Pittsburgh, PA, U.S.A.

Introduction

Low-density parity check (LDPC) codes exhibit performance close to the Shannon limit. High-rate LDPC codes are actively being considered in magnetic recording because, when decoded by the iterative message-passing algorithm, they show better decoding performance than turbo codes as the block length increases. However, their decoding complexity is a challenge.

In this paper, we develop a new decoding scheme for LDPC codes that effectively reduces the decoding complexity. We refer to it as *turbo like decoding*. By graph transformations, we divide systematically an arbitrary factor graph with cycles into several cycle-free parts that are connected to each other by interleavers, forming an equivalent turbo-like structure. Turbo-like decoding then decodes in each cycle-free component and shares probabilistic information among them. We show that turbo-like decoding converges to decoding success faster than the sum-product decoding algorithm at comparable BER performance.

Graph Partition Algorithm

We present now an algorithm that partitions a factor graph with cycles into several cycle-free components (trees or forests) connected by interleaver-like structures. The algorithm is as follows:

```
1. Generate a tree $T_G$ from the original factor graph $G$. Let $G_1 = G - T_G$, i.e., $G_1$ is the
   remaining graph when all edges in $T_G$ are deleted from $G$. Let $C_1 = T_G$ and m = 1.
2. If the remaining graph $G_1$ is cycle-free go to step 4.
   If NOT, m = m+1, go to step 3.
3. Assume $G_1$ contains $n$ disconnected parts $P_1, P_2, ..., P_n$.
   For i = 1 : n
       Generate a tree $T_i$ for the $i$th disjoint part $P_i$.
   End
   Let $C_m = T_1 \cup T_2 \cup \cdots \cup T_n$ and $G_1 = G_1 - C_m$, go to step 2.
4. $C_{m+1} = G_1$, End.
```

From this algorithm, $G = C_1 \cup C_2 \cup \cdots \cup C_{m+1}$ and C_i's are cycle-free. Fig. 1 shows that this algorithm splits the graph in fig.1.(a) into the two trees shown in fig 1.(b). During the partition process, some variable nodes and those edges incident on them may split into different cycle-free components, e.g., see the variable node x_1 in figure 1(a) that is split into two, see fig. 1.(b). When interchanging information between different components, we need to remerge those edges together, as shown in fig. 1.(c). We refer to these merged parts as an interleaver.

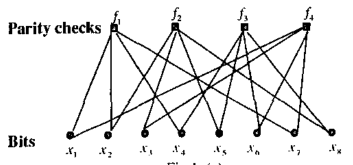

Fig.1. (a)

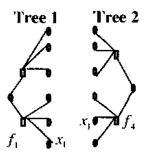

Fig.1. (b)

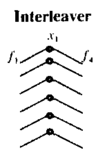

Fig.1. (c)

Turbo-like decoding algorithm (TLDA)

Assume a factor graph G contains K cycle-free components $C_1, C_2, ..., C_K$.

```
Step 1: Initialization
Step 2: For i = 1 : K
            Decode the $i$th cycle-free component $C_i$, using the updated information from its interleaver
            and then transmit the new probabilistic information to its interleaver.
        End
Step 3: Compute the decoding output. If success is achieved, go to step 4. Otherwise, go back to step 2.
Step 4: End.
```

The message updating equations are the same as those for the sum-product decoding algorithm.

Advantages of turbo-like decoding algorithm

It is well known [F. R. Kschischang and B. J. Frey, Feb. 1998] that with a cycle-free factor graph, the sum-product algorithm terminates in a finite number of steps and yields minimum symbol error probability. Therefore, in isolation, the local decoding for each cycle-free component is optimal. TLDA is still iterative: each component transmits its *a posteriori probability* (APP) information to the others through interleavers and, in turn, these components use these APPs as a priori information to start their own decoding process.

BER performance and computational cost

We consider a randomly generated regular LDPC code with uniform column weight 2. Its block length is 5850 bits and its code rate is 8/9. We compare by 10^4 Monte Carlo simulations in the plots below the TLDA and the standard sum-product algorithm in an AWGN channel. The plot on the left shows that the two algorithms have similar BER performance, while the plot on the right shows that TLDA is 50 % faster than the sum product algorithm at low SNR (We have proved that the computational cost per iteration of TLDA is exactly the same as the computational cost per iteration of sum-product algorithm).

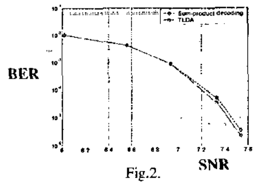

Fig.2.

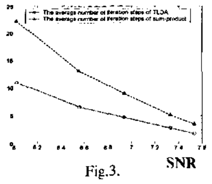

Fig.3.

Acknowledgment: This work is supported by the DSSC at Carnegie Mellon University.