行政院國家科學委員會專題研究計畫　期中進度報告

先進錯誤控制技術之研究(1/3)

計畫類別：個別型計畫
計畫編號：NSC94-2213-E-009-054-
執行期間：94 年 08 月 01 日至 95 年 07 月 31 日
執行單位：國立交通大學電信工程學系(所)

計畫主持人：蘇育德

計畫參與人員：陳彥志、李昌明、龔炳全、邱泰祥、王詩堯

報告類型：精簡報告
報告附件：出席國際會議研究心得報告及發表論文
處理方式：本計畫可公開查詢

中　華　民　國 95 年 5 月 30 日

行政院國家科學委員會專題研究計畫成果報告

先進錯誤控制技術之研究 (1/3)

An Investigation on Advanced Error-Control Coding Technologies (1/3)

計畫編號：NSC 94-2213-E-009-054
執行期間：94 年 8 月 1 日至 95 年 7 月 31 日
主持人：蘇育德教授　國立交通大學電信工程學系
計畫參與人員： 陳彥志、李昌明、龔炳全、邱泰祥、王詩堯

中文摘要

在不增加編碼與平均解碼延遲下，塊間交錯渦輪碼提供了比傳統渦輪碼更佳的效能，但其需要較大的記憶體。因此我們提出一種新式的解碼程序去降低記憶體需求並進一步降低解碼延遲。我們提出了兩種圖示法來描述新發展出的解碼法之訊息轉移過程。第一種為多層次分解圖，另一為多層次分解子圖。利用這兩類圖示法可以有效的評估有限遞回次數對系統性能的影響，並且此種新圖可用來尋找、發展更佳的解碼排程。

關鍵字：塊間交錯渦輪碼、分解圖、渦輪碼。

Abstract

The class of inter-block permuted turbo codes (IBPTC) provides significant performance gain over the conventional turbo codes (TC) without increasing the average encoding and decoding latencies. The major shortcoming of IBPTC is the increased memory requirement. We present a multi-stage factor graph and sub-graph approach to describe the schedule and message-passing flow of an IBPTC decoder. Based on these new graphic representations, we propose a new decoding schedule to reduce both the memory requirement and the decoding latency. Evaluating the effect of the finite iterations on the decoder performance becomes feasible through the use of such graphs. The proposed graphic approach can be applied to find more efficient (in terms of computing complexity and storage) decoding schedules for other iterative decoders as well.

Keywords: IBPTC、Factor Graph、MSFG、Turbo Code。

1. Introduction

An inter-block permuted turbo code (IBPTC) [1] differs from a conventional turbo code (TC) in that the block-wise interleaver used in the latter is replaced by an inter-block permutation interleaver (IBPI). The structure of the IBPI enables an IBPTC decoder to extend the message-passing operation to as many blocks as possible, constrained only by the decoding iteration number and the duration of the information sequence. By contrast, the message-passing operation associated with a conventional TC decoder is confined to within a block. The extended message-passing range gives an IBPTC an (almost) unbounded equivalent interleaving depth but, with a suitable decoding schedule, does not prolong the average (per block) decoding delay.

Hardware implementation of an IBPTC is very flexible and naturally suitable for parallel processing. A multi-processor decoding architecture that exploits such flexibility was presented in [4]. The architecture shares all related decoding components, namely, a posteriori probability (APP) decoders, memory space, and interleavers/deinterleavers, while simultaneously decoding multiple

blocks. [3] considered both early-stopping and memory management concerns to minimize the storage requirement. It was shown there that a good stopping mechanism also bring about performance improvement and complexity reduction.

The pipeline decoder architecture [1] yields high throughput but does not offer much design flexibility. It is composed of several serially concatenated APP processors whose number is proportional to the maximum number of decoding iterations. Such a structure requires a large memory to store tentative extrinsic information and received samples. Moreover, most of the APP processors are very likely to become idle at high signal-to-noise ratio (SNR) due to frequent early terminations.

As was shown in [3], the decoding schedule determines the decoder configuration (i.e., the arrangement and interconnections of the decoding components mentioned above) and, to a less extent, the system performance. Therefore, it is important to have a judicious scheduling plan so as to maximize the hardware utilization efficiency and minimize the required storage space. This work exploits more dynamic decoding schedules and architectures that offer more flexibility in trading performance, latency, computing complexity and storage need. We use the pipeline-implied decoding schedule as the benchmark schedule for comparison purpose.

In this study, we propose a directional and hierarchical graph representation called multi-stage factor graph (MFG) to describe the message-passing process time evolution. We further construct a causal multi-stage sub-graph (CMSG) from an MFG to describe the operation flow associated with each input block. Based on our new graphic approach, we present new decoding schedules that require less storage space without compromising performance.

The proposed MFG and CMSG avoid ambiguous description of cyclic or loopy message-passing events. They can also be applied to (i) devise new decoding schedules for LDPC codes [6], LDPC convolutional codes [7], turbo codes, multiple turbo codes, etc., and (ii) analyze the impact of the decoding schedule on computing complexity and storage requirements.

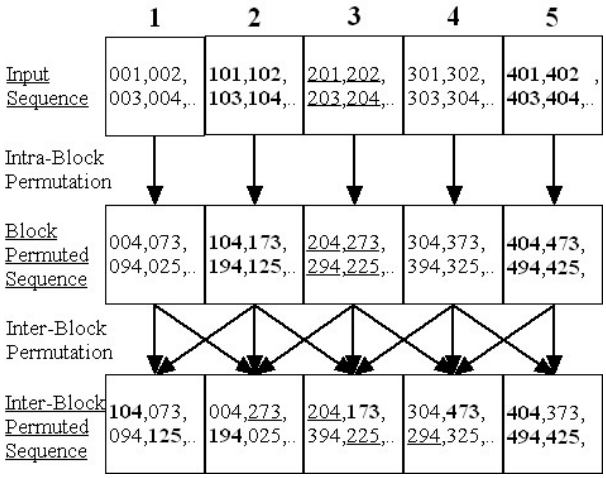

Fig. 1: An example of inter-block permutation; a conventional TC needs only to perform the (intra-)block permutation.

The rest of this report is organized as follows. The following section provides an overview of some IBPTC design issues including the interleaver structure, pipeline and dynamic architectures, and stopping criteria. Section 3 presents a hierarchical graphic representation for IBPTC and its generalization. We show how a CMSG, as a subset of a MFG, can be constructed and present new IBPTC decoding schedules via the CMSG representation.

Simulated performance of the proposed schedule and that of the standard schedule is given in Section 4. The last section provides some concluding remarks.

2. Introduction to IBPTCs

2.1 IBP and encoding procedure

As shown in Fig. 1, IBP interleaving is a two-stage process. The input data stream is segmented into blocks and bits within each block are interleaved through the intra-block permutation. The interleaved data bits in each block are further permuted to locations in the two neighboring blocks and the original block through the inter-block permutation. We call such an interleaver as a symmetric IBP interleaver with interleaving span $S = 1$.

For an IBPTC to encode a given, say the ith, interleaved block, the encoder has to wait until the $(i+S)$th block is completely received. Hence there is $(1+S)L$-bit encoding delay, where L is the block size, and the encoding complexity is the same as that of a conventional TC with block size $(1+S)L$.

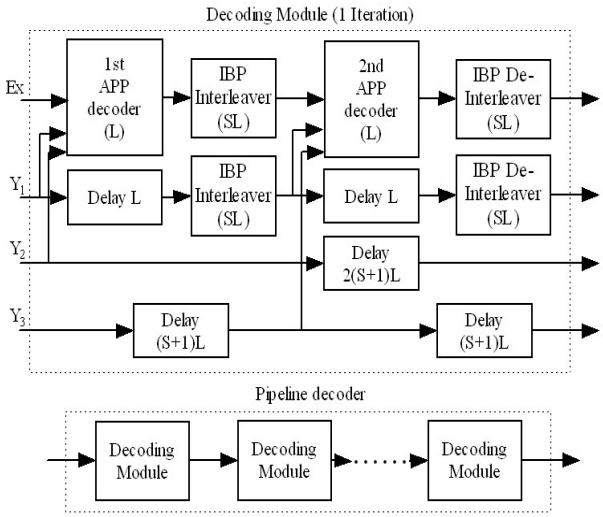

Fig. 2: A pipeline decoder for general IBPTCs.

2.2 Pipeline decoding architecture

Fig. 2 shows a pipeline decoder for IBPTCs. The APP decoder requires L decoding time units to output the extrinsic information and the IBP interleaver requires S more extrinsic information blocks to generate permuted version. Therefore there is an $(S+1)L$ delay between two consecutive APP decoders and the decoder outputs a block after $2N(1+S)L$ unit times, where N is the iteration number.

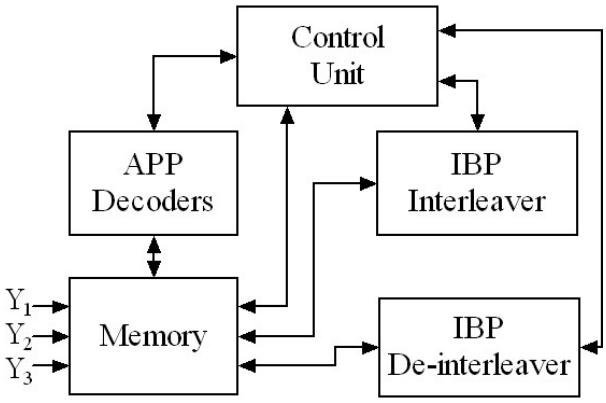

Fig. 3: A dynamic IBPTC decoder structure.

2.3 A dynamic decoding architecture

Dynamic decoder [4] refers to a class of generic decoder structures as that shown in Fig. 3. The control unit determines the operation schedule of all available decoding resources--the APP decoders, the interleavers and memory storage. The structure provides trade-off between the complexity and decoding latency. Furthermore stopping mechanism and memory control can be joint design to economize memory usage.

The pipeline decoder can be regarded as a dynamic decoder with a special decoding schedule which we refer to as the standard schedule. To present the proposed schedule, we need a new graph representation to describe the corresponding time sequence and message-passing procedure. The new representation can be regarded as a

time-evolution version of the conventional factor graph [5] and will be detailed in the next section.

2.4 Stopping mechanism and its applications

[3] presents a joint IBPTC decoder and stopping mechanism design to improve the system performance. The multi-order hybrid CRC and sign check stopping mechanism in [3] provides a very reliable stopping indication, which also offers extra information to help decoding unterminated blocks and enhance performance. Taking the finite storage into design consideration, the stopping mechanism decisions are also used for releasing memory space. Early termination thus has much more impact in a dynamic decoder than in a conventional decoder.

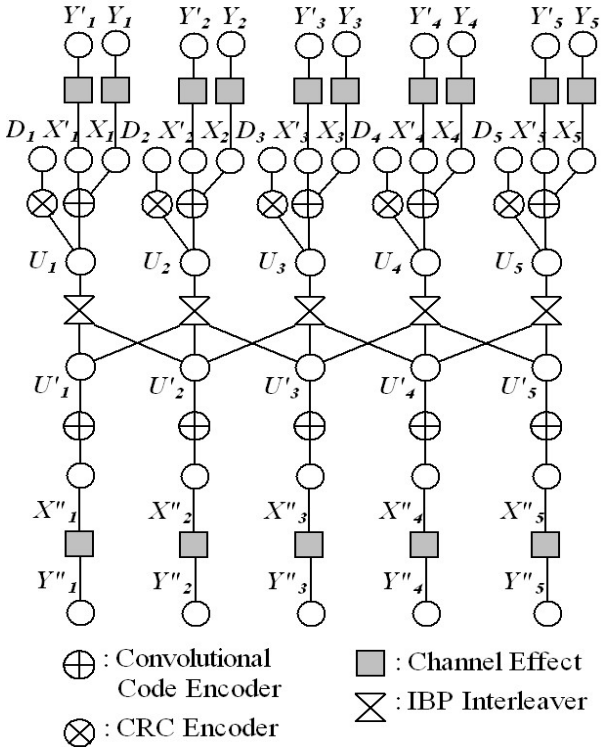

Fig. 4: Factor graph representation of a CRC- and IBPTC-coded communication link.

3 Factor graph representation and schedule

Fig. 4 shows a hierarchical factor graph representation of a coded communication link based on IBPTC and CRC codes. There are five data blocks and interleaver span S is *1*. $\boldsymbol{D}_i$ denotes the ith input data block. $\boldsymbol{U}_i$ and $\boldsymbol{U}'_i$ denote the corresponding CRC encoded sequence and its permuted version. The convolutional-encoded codeword sequences X_i, $\boldsymbol{X}'_i$, $\boldsymbol{X}''_i$ are corrupted by noise before becoming the received sequences $\boldsymbol{Y}_i$, $\boldsymbol{Y}'_i$, $\boldsymbol{Y}''_i$. The decoder (node) performs APP decoding to generate the reliability for the data sequence and the CRC decoder (node) checks the decoded block to make a stop-or-go decision for the ensuing APP decoding and outputs some related information. The IBP interleaver (node) sends the extrinsic information to due positions. The graph indicates relations amongst nodes but does not detailed message-passing flow. We thus introduce a modified graphic representation to describe the behavior of the decoding process.

3.1 Multi-stage factor graph

A hierarchical representation is used to describe of the time evolution of the message-passing process in a factor graph. We use nodes $\boldsymbol{O}_i$ and $\underline{\boldsymbol{O}}_i$ to denote the upper branch that includes the set of nodes $\boldsymbol{D}_i$, $\boldsymbol{U}_i$, $\boldsymbol{X}_i$, $\boldsymbol{X}'_i$, $\boldsymbol{Y}_i$, $\boldsymbol{Y}'_i$ and the lower branch--nodes $\boldsymbol{U}'_i$, $\boldsymbol{X}''_i$, $\boldsymbol{Y}''_i$--respectively; see Fig. 5 (a). Fig. 5 (b) shows the simplified factor graph of the system of Fig. 4.

We extend undirected factor graph to a directed multi-stage factor graph (MFG). Fig. 6 (a) shows a directed factor graph extended from the simplified graph shown in Fig. 5 (b). Nodes $\boldsymbol{O}_i$ and $\underline{\boldsymbol{O}}_i$ are modified as O_i^j and $\underline{O}_i^j$ respectively, where j denotes the jth ADR corresponding to the ith block. The check nodes of interleaver are also substituted by

directed edges. We take the maximum number of iterations as *3* and the number of the stage as *6*. The performance of Fig. 6 (a) is equivalent to the pipeline decoder and the pipeline decoder is the minimum memory storage and the shortest latency realization when the stopping mechanism is not considered.

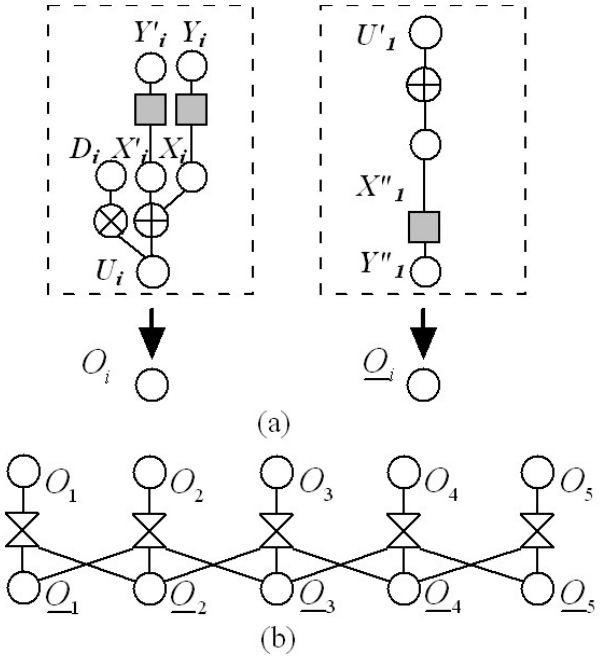

Fig. 5: A systematic hierarchical nodes combining (a) to obtain a simplified factor graphic representation (b).

One can redirect message-passing edge to create a new MFG. We redirect $\underline{O}_i^j \to O_{i+1}^{j+1}$ and $O_i^j \to \underline{O}_{i-1}^{j+1}$ to $\underline{O}_i^j \to O_{i+1}^{j-1}$ and $O_i^j \to \underline{O}_{i-1}^{j+3}$; the intuitive MFG Fig. 6 (a) is substituted by the new MFG Fig. 6 (b).

The redirection enables early decoding of some selected blocks. Consider the two paths in Figs. 6 (a) and (b),

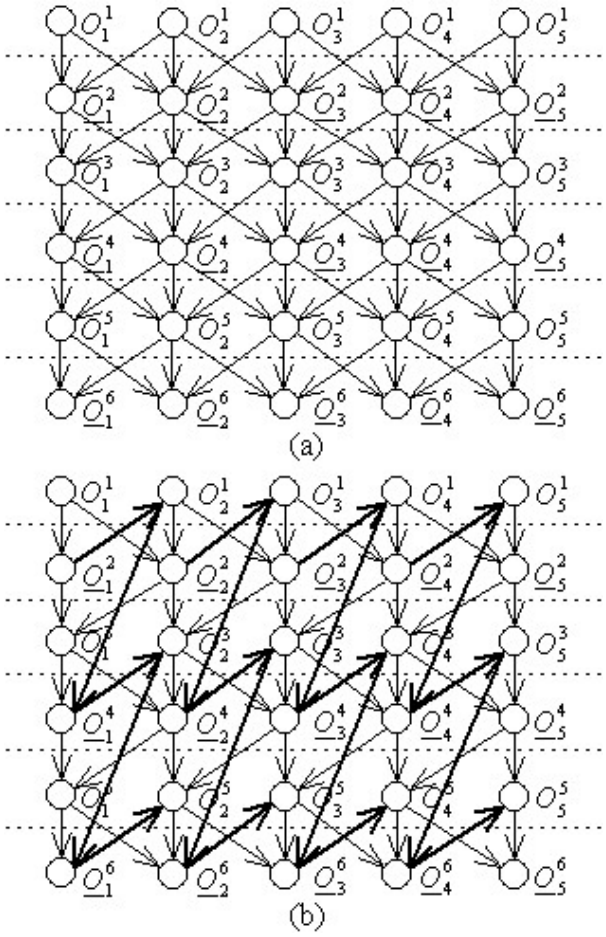

Fig. 6: A multi-stage factor graph representation of an IBPTC decoding procedure.

$\underline{O}_1^6 - O_2^5 - \underline{O}_3^4 - O_4^3 - \underline{O}_5^2 - O_6^1$ and $\underline{O}_1^6 - O_2^3 - \underline{O}_3^2 - O_3^1$. For the first path node $\underline{O}_1^6$ is not activated until the *5*th block is received while the same node is activated as soon as the *4*th block is received. When the total encoded data length is much larger than *5* blocks, the decoding latency for node $\underline{O}_1^6$ of Fig. 6 (a) becomes intolerably long. Note that the message-passing range of a node can be unbounded in Fig. 6 (b) but it is bounded in Fig. 6 (a). Nevertheless, a larger range does not necessarily results in more reliable information collected.

MFG can also be applied to other codes such as LDPC code or repeated accumulated code. One can analyze behavior of the belief evolution for different schedule to reduce required iterations. This graph is a useful tool for computation and complexity reduction.

Remark 1: We count the stage number *j* according to the APP decoding round (ADR) because the computing complexity is proportional to the number of ADRs.

Remark 2: The interleaver check node is originally represented by equality, we replace such node by a directed edges. For irregular turbo codes [8], the equality node may be connected by more than two nodes hence should be kept to describe this effect.

3.2 Causal multi-stage sub-graph

A causal multi-stage sub-graph (CMSG) is used to describe the message-passing procedure for the streaming-type codes such as IBPTC and the LDPC convolutional code [7]. Streaming-type code factor graphs are often a concatenation of small graph duplications and the small graph is enough to demonstrate overall message-passing. Therefore, a CMSG is a subgraph of an MFG to demonstrate a complete graph. Furthermore decoding schedule can be simply demonstrated.

Fig. 7 shows two examples. Assume the *4*th block is input and we extract Fig. 7 (a) and (b) from Fig. 6 (a) and (b). In Fig. 6 (a), the CMSG is composed of $\{ O_4^1, \underline{O}_3^2, O_2^3, \underline{O}_1^4 \}$; real-line denotes message-passing for the instance the *4*th block input and dashed-line denotes message-passing with the other sub-graphs. We eliminate dashed-line and tilt sub-graph to render the corresponding CMSG Fig. 7 (c) composed of $\{ P_{40}^1, \underline{P}_{41}^1, P_{42}^1, \underline{P}_{43}^1 \}$. Similarly, we obtain the CMSG of Fig. 7 (d) from Fig. 6 (b).

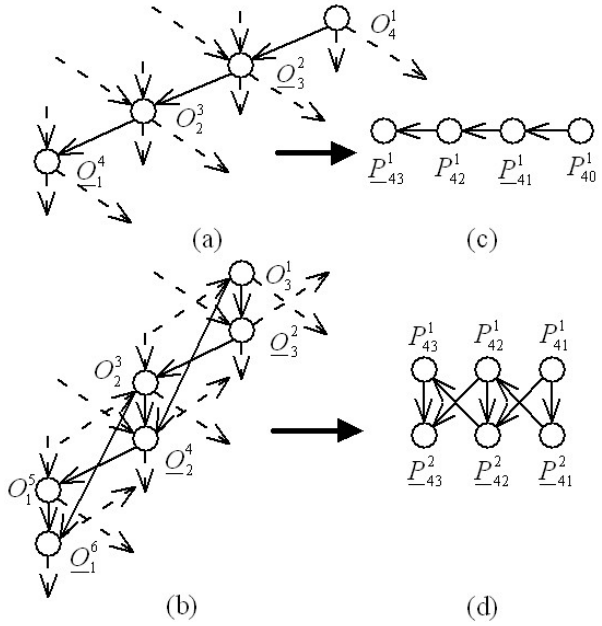

Fig. 7: A multi-stage sub-graph of the MFG derived from Fig. 6.

Decoding schedule for each input instance is also clearly pointed out by these graphs. In Fig. 7 (c), the decoder decodes the aligned nodes $P_{40}^1 \to \underline{P}_{41}^1 \to P_{42}^1 \to \underline{P}_{43}^1$. While in Fig. 7 (d), the decoder decodes the aligned nodes $P_{41}^1 \to \underline{P}_{41}^2 \to P_{42}^1 \to \underline{P}_{42}^2 \to P_{43}^1 \to \underline{P}_{43}^2$.

The ordinal number of the stage of node P_{jl}^i or $\underline{P}_{jl}^i$ for the original MFG can be counted as follows. Define the maximum stage of each column in the CMSG by $N_S(l) = \max_{P_{jl}^i, \underline{P}_{jl}^i} i$.

The ordinal number of P_{jl}^i or $\underline{P}_{jl}^i$ is

$i+\text{sum}_{m=1}^{l-1}N_s(m)$. P_{jl}^{i} and $\underline{P}_{jl}^{i}$ in the CMSG can be simply mapped to $O_{j-1}^{i+\text{sum}_{m=1}^{l-1}N_s(m)}$ and $\underline{O}_{j-1}^{i+\text{sum}_{m=1}^{l-1}N_s(m)}$ in the MFG respectively.

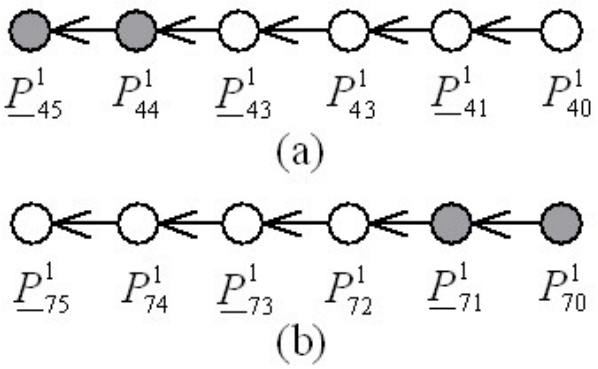

Fig. 8: An illustration of Virtual nodes.

The CMSG is not only useful for describing schedule but also for memory counting. Each CMSG represents the operation range. Fig. 7 (d) requires memory storage for 4 blocks of received samples and temporary extrinsic information. Required memory is easily counted.

Virtual node concept is introduced for finite steaming-type codes to deal with the fact that the beginning and the last blocks have no preceding and ensuing blocks. A virtual node is in a sense a deactivated node. Fig. 8 shows examples from Fig. 7 (a). Fig. 8 (a) and (b) plot virtual nodes P_{44}^{1} and $\underline{P}_{45}^{1}$ for the near beginning block and virtual nodes P_{70}^{1} and $\underline{P}_{71}^{1}$ for the near ending block. The inclusion of virtual nodes in a CMSG enables us to describe the complete decoding schedule by a single graph. At last, we summarize two rules for the virtual nodes: P_{jl}^{i} is a virtual node if $(j\text{-}l)> N$ or $(j\text{-}l)<1$.

4. A memory-saving schedule

Decreasing convergence rate implies that tentative information is to be kept in the memory pool longer and the required memory space increases accordingly. Thus memory shortage is likely to occur at low SNR, leading to more forced early-terminations and deteriorated performance. A candidate solution to avoid forced early-termination is to start a new decoding round before all the information within its span becomes available. Inevitably, such an early-start decoding has to use some non-updated extrinsic information. Invoking the early-start concept, we propose the decoding schedule of Fig. 9 (a). There are three schedule-related parameters: iteration number (IN), repeated decoding rounds (RDR) and decoding window width (DWW). For each block, the standard schedule undergoes IN + 1 = 6 ADRs but the proposed schedule undergoes 25 ADRs. Since the information associated with each block is used more often, the decoder converge with less information (but worse performance) and therefore requires smaller memory space.

The *k*th incoming block can be decoded and interleaved immediately at the pre-permutation node P_{k0}^{1} whose post-permutation counterpart will not be initiated until all related blocks are received. Fig. 9 (b) shows the corresponding partial MFG with 4 stages and one can find the pre-permutation part successively processed by twice at beginning. This benefits the IBPTC performance but pays the computation power.

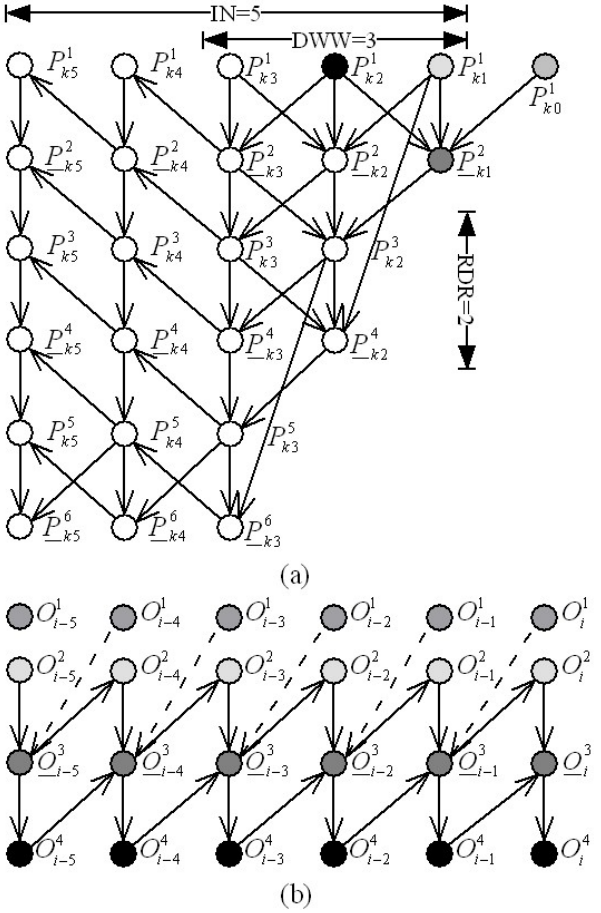

Fig. 9: (a) A memory saving schedule for the IBPTC; (b) multi-stage factor graph representation.

At low SNR, the convergence speed is slow and the required memory is large. The proposed schedule allows more ADRs with less information for each block even if the available memory is limited. This feature provides an alternative to avoid memory shortage-induced performance loss at low SNRs.

5. Numerical Results

Fig. 10 and Fig. 11 show the BER and average iteration number performance. The 3GPP defined interleaver and rate 1/3 turbo code [9] are used with a block length of 400. Tail-biting encoding [10] and Log-MAP decoding with the third-order hybrid CRC and sign-check stopping criterion [3] are assumed. One memory unit (MU) has space for 400 soft-bits. 3 MUs and 1 MU are needed for the received samples and extrinsic information per block. Denote by Schedule A and Schedule B respectively the two schedules shown in Fig. 9 (a) with and without node P_{k0}^1.

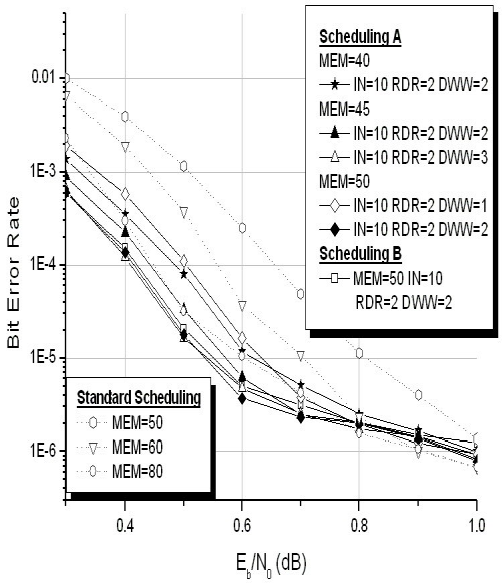

Fig. 10: BER performance as a function of SNR for three decoding schedules.

In Fig. 10, Schedule A with IN=10, ADR=2, DWW=2 and 45 MUs outperforms the standard schedule with 80 MUs, yielding a memory reduction greater than 40%. However, the former needs about six and two more iterations for SNR = 0 dB and SNR > 0.4 dB, respectively. The memory saving is obtained at the expense of higher computing complexity. It is comforting to see that the proposed schedule requires no more than 6 average iterations for SNR > 0.8dB.

Increasing DWW improves the BER performance at the cost of increased computing complexity; see Fig. 11.

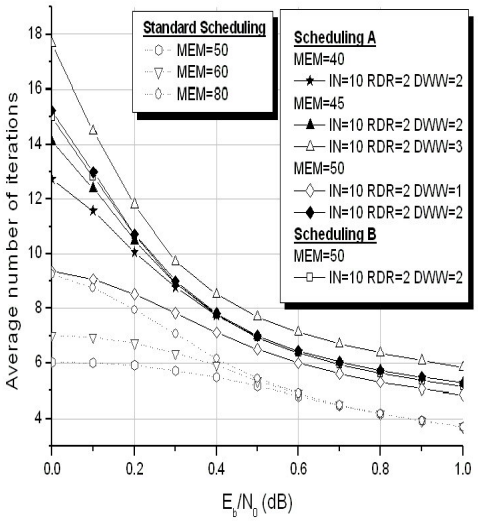

Fig. 11: Average iteration number performance as a function of SNR for three decoding schedules.

For IN = 10, RDR = 2, Schedule A with DWW = 3 and 45 MUs outperforms that with DWW = 2 and 45 MUs. The same figure indicates that Schedule A with DWW = 2 and 50 MUs outperforms that with DWW = 1 and 50 MUs. The increase of memory also helps enhancing the BER performance. With IN = 10, RDR = 2 and DWW = 2 Schedule A performance when the memory size increases from 40 MUs to 50 MUs. However, additional two more iterations are needed at SNR = 0.0 dB. Finally, we note that Schedule A slightly outperforms Schedule B for 50 MUs, IN = 10, RDR = 2 and DWW = 2. The required average iteration number is also slightly higher.

6. Conclusion

In this work, we propose two new graphic representations, MFG and CMSG, to describe the message-passing process associated with an iterative decoding process for IBPTCs.

A new decoding schedule based on the new graphic representations is proposed for decoding IBPTCs. The proposed schedule outperforms the standard schedule, reduces the storage requirement while calling for only a moderate increase in the average iteration number at higher SNR.

The proposed graphic representation can also be used to derive other decoding schedules that offer tradeoffs in complexity, performance and latency. It can also be applied to devise new decoding schedules for other codes like conventional and convolutional LDPC codes.

[References]

[1] **Y.-X. Zheng, Y. T. Su, "On inter-block permutation and turbo codes."** ***Proc. International Symp. Turbo Codes and Related Topics*****, Brest, France, Sep. 2003.**

[2] **C. Berrou, A. Glavieux, and P. L. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo-codes."** ***Proc. ICC'93*****, pp. 1064-1070, May 1993.**

[3] **Y.-X. Zheng, Y. T. Su, "Iterative decoding algorithm and terminating criteria for inter-block permuted turbo codes."** ***Proc. Personal, Indoor and Mobile Radio Communications*****, vol. 2, pp. 1116-1120, 5-8 Sep. 2004.**

[4] **Y.-X. Zheng, Y. T. Su, "High throughput turbo coding system." Wireless World Research Forum Meeting 14, San Diego, California, 7-8 Jul. 2005.**

[5] **F. R. Kschischang, B. J. Frey, H.-A. Loeliger, "Factor graphs and the sum-product algorithm."** ***IEEE Transactions on Information Theory*****, vol. 47, no. 2, pp. 498-519, Feb. 2001.**

[6] **R. G. Galleger,** ***Low-density parity-check codes*****, MIT Press, Cambridge, Mass., 1963.**

[7] **R. M. Tanner, D. Sridhara, A. Sridharan, T. E. Fuja, D. J. Costello, "LDPC block and convolutional codes based circulant matrices."** ***IEEE Transactions on Information Theory*****, vol. 50, no. 12, pp. 2966-2984, Dec. 2004.**

[8] **J. Boutros, G. Caire, E. Viterbo, H. Sawaya, S. Vialle, "Turbo code at 0.03 dB from capacity limit."** ***Proc. International Symposium on Information Theory*****, pp. 56, 30 Jun.-5 Jul. 2002.**

[9] ***TS 25.222 V3.1.1 Multiplexing and channel coding (TDD)*****, 3GPP TSG RAN WG1, Dec. 1999.**

[10] **C. Weiss, C. Bettetter, S. Riedel, D. J. Costello, "Turbo decoding with tail-biting trellis."** ***ISSSE'98*****, Pisa, Italy, pp. 343-348, 1998.**