Error Structure Aware Parallel BP-RNN Decoders for Short LDPC Codes

Joachim Rosseel, Valérian Mannoni, Valentin Savin, Inbar Fijalkow

Error Structure Aware Parallel BP-RNN Decoders for Short LDPC Codes

Joachim Rosseel*†, Valérian Mannoni*, Valentin Savin*, Inbar Fijalkow†
*CEA-Leti, Université Grenoble Alpes, F-38000 Grenoble, France
{Joachim.Rosseel, Valerian.Mannoni, Valentin.Savin}@cea.fr
†ETIS, CY Cergy Paris Univ., ENSEA, CNRS F-95000, France
{Joachim.Rosseel, Inbar.Fijalkow}@ensea.fr

***Abstract*—This article deals with the decoding of short block length Low Density Parity Check (LDPC) codes. It has already been demonstrated that Belief Propagation (BP) can be adjusted to the short coding length, thanks to its modeling by a Recurrent Neural Network (BP-RNN). To strengthen this adaptation, we introduce a new training method for the BP-RNN. Its aim is to specialize the BP-RNN on error events sharing the same structural properties. This approach is then associated with a new decoder composed of several parallel specialized BP-RNN decoders, each trained on correcting a different type of error events. Our results show that the proposed specialized BP-RNNs working in parallel effectively enhance the decoding capacity for short block length LDPC codes.**

I. Introduction

Short-packet machine-to-machine communications, central to the emerging Internet of Things (IoT) technology, have revitalized interest in research and practice of efficient error correcting codes, for messages ranging from a few tens up to a few hundred bits. While important progress has been made over the last years in understanding the limits of coding at short block lengths [1], the design of efficient short codes and decoding algorithms still raises many challenges [2].

Low Density Parity Check (LDPC) codes [3] are a class of error correcting codes defined by sparse bipartite graphs [4]. They are well-known for their excellent error correction performance at long block lengths, achieving near Shannon channel capacity performance under Belief Propagation (BP) decoding, in the asymptotic limit of the code length [5]. For codes defined by cycle-free bipartite graphs, BP decoding outputs the maximum a posteriori estimates of the coded bits [6]. Although bipartite graphs associated with practical codes contain cycles, BP decoding may still take effective advantage of sparse, long enough graphs, devoid of short cycles. However, for short codes, short cycles may not be avoidable, thus significantly degrading the BP performance. This is even more pronounced for High Density Parity Check (HDPC) codes, defined by higher density bipartite graphs [7].

To reduce the impact of short cycles, a weighted BP decoding has been introduced in [8], where the weights are optimized using a Neural Network (NN). The topology of the NN mimics the BP decoding process, with unwrapped decoding iterations. The approach may use either a feedforward (FF) or a Recurrent NN (RNN). The corresponding decoders are termed as BP-FF and BP-RNN. It has been shown in [8] that the BP-RNN is able to outperform the usual BP decoder for short Bose-Chaudhuri-Hocquenghem (BCH) codes, belonging to the class of HDPC codes. Subsequently, several variants of NN-based BP decoding have been proposed in the literature. [9] proposed the design of new decoding rules for finite-alphabet iterative decoders, based on a quantized NN model. [10] developed a pruning method of irrelevant check nodes in a neural BP model, aimed at jointly optimizing the code construction and the decoding. In [11], a neural BP decoding approach was proposed for cyclic redundancy check (CRC)-assisted polar codes.

In this paper, we focus on BP-RNN decoding of short block length LDPC codes. To improve the decoding performance, our approach aims at specializing BP-RNN decoders to difficult error events. To do so, we first propose a classification of the error events, according to the structure of the induced sub-graph. The classification is driven by the impact of the induced sub-graph on the BP decoding performance. Then, a parallel construction, comprising several BP-RNN decoders running in parallel, is described, where each BP-RNN is specialized to a specific error class. Finally, we discuss the training of the parallel BP-RNN decoders, and provide a method to reduce their number, without jeopardizing the decoding performance, by introducing a similarity rate metric.

The paper is organized as follow. Section II introduces the notations and recalls the BP-RNN decoding algorithm. Section III defines the classification of the error events, the parallel construction of BP-RNN decoders, the training of the BP-RNN decoders, and the similarity metric used to reduce their number. Finally, Section IV presents the numerical results, and Section V concludes the paper.

II. Neural BP decoding

We consider an LDPC code defined by a Tanner (bipartite) graph with N variable-nodes and M check-nodes, denoted respectively by $n \in \{1, \dots, N\}$ and $m \in \{1, \dots, M\}$. We further denote by $\mathcal{N}(m)$ the set of variable-nodes connected to a check-node m, and by $\mathcal{M}(n)$ the set of check-nodes connected to a variable-node n.

BP decoding consists of an iterative exchange of messages along the edges of the Tanner graph, where each message

This work was partially supported by the ECSEL Joint Undertaking (JU) programme, under grant number N°826276 (CPS4EU project).

provides an estimation of the incident variable-node. BP-RNN and BP-FF decoding algorithms are weighted variants of the BP decoding, where exchanged messages are multiplied by weights learned through an either RNN or FF-NN approach. The underlying NN contains three types of *neural layers*, each one corresponding to a step of the BP algorithm. The *check-pass layer* and the *data-pass layer* carry out the computation of messages outgoing from check-nodes and variable-nodes, respectively. Each one of them contains a number of neurons equal to the number of edges of the Tanner graph. In addition, the *a posteriori Log Likelihood Ratio (LLR) layer* consists of N neurons, computing the a posteriori LLR values of the N variable-nodes. The three layers of the NN are connected such that a check pass layer, followed by a data pass layer and an a posteriori LLR layer model one iteration of the BP decoding. In particular, it is worth stressing out the differences between the edges of the Tanner graph (corresponding to neurons in the check-pass and data-pass layers), and the edges of the NN.

The formulas below detail the calculation of messages within each layer. We denote by $\beta_{m\to n}$ and $\alpha_{n\to m}$ the messages computed by the check-pass and data-pass layers (where (m,n) is an edge of the Tanner graph), and by $\tilde{L}_n$ the messages computed by the a posteriori LLR layer. The observed (channel) LLR values are denoted by $L_{\mathrm{ch},n}$, and are used to initialize $\alpha_{n\to m}$ messages prior to the first iteration.

$$\beta_{m\to n} = 2\tanh^{-1}\left(\prod_{n'\in\mathcal{N}(m)\setminus n}\tanh\left(\frac{\alpha_{n'\to m}}{2}\right)\right) \tag{1}$$

$$\alpha_{n\to m} = L_{\mathrm{ch},n} + \sum_{m'\in\mathcal{M}(n)\setminus m} w_{m'\to n\to m}\beta_{m'\to n} \tag{2}$$

$$\tilde{L}_n = L_{\mathrm{ch},n} + \sum_{m\in\mathcal{M}(n)} \tilde{w}_{m\to n}\beta_{m\to n} \tag{3}$$

It can be observed that weights are applied only on the NN edges incoming to the data-pass (2) and a posteriori LLR (3) layers. Each weight corresponds to one specific edge of the NN. In (2) the weights are denoted by $w_{m'\to n\to m}$, where the subscript indicates both the corresponding neuron $n\to m$ in the data-pass layer, and the incoming NN edge from neuron $m'\to n$ in the check-pass layer. In (3) the weights are denoted by $\tilde{w}_{m\to n}$, where the subscript indicates the corresponding neuron n in the a posteriori LLR layer, and the incoming NN edge from neuron $m\to n$ in the check-pass layer. For the BP-RNN, the weights only depend on the corresponding edges of the NN, while for the BP-FF they also depend on the iteration number (note that, to simplify notation, we have not indicated the iteration number on the above formulas). It is worth noticing that despite the reduced number of trained weights, the BP-RNN achieves similar performance to the BP-FF [8]. An alternative approach suggested in [8] to further reduce the number of weights is based on the following data-pass layer,

$$\alpha_{n\to m} = L_{\mathrm{ch},n} + w_{n\to m}\sum_{m'\in\mathcal{M}(n)\setminus m}\beta_{m'\to n}, \tag{4}$$

where the applied weight only depends on the data-pass neuron. This simplification reduces the training complexity and makes it possible to reuse conventional BP decoding architectures for efficient hardware implementation. The BP-RNN using (4) will be referred to as BP-RNN Hardware Friendly Implementation (BP-RNN-HFI).

To train the BP-RNN, we use the following Bit Error Rate (BER) loss function, assuming without loss of generality that the zero codeword is transmitted:

$$\mathrm{Loss}(\tilde{L}) = \frac{-1}{N}\sum_{n=0}^{N-1}\log(\sigma(\tilde{L}_n)) \tag{5}$$

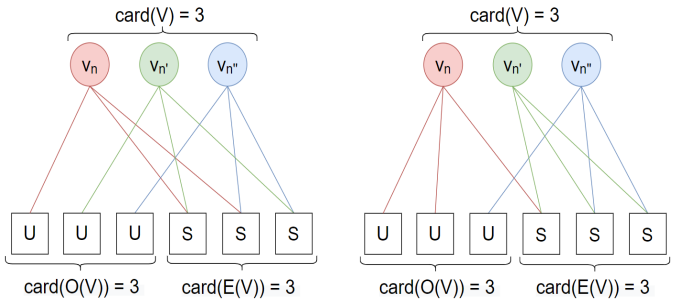
where $\sigma(x) = (1+\exp(-x))^{-1}$ is the sigmoid function, converting the LLRs into probability values. The loss function is minimized during the NN training, thus improving the BER of the trained decoder.

III. Specializing BP-RNN Decoders according to an Error Events Classification

A. Absorbing-type classification of error events

Let V be a set of variable-nodes, and C be the set of check-nodes connected to at least one variable-node in V. We denote by $O(V)\subset C$ the set of check-nodes connected an odd number of times to V (that is, they have odd degree in the sub-graph induced by V). Thus, $E(V) := C\backslash O(V)$ is the set of check-nodes connected an even number of times to V. The set V is said to be an *absorbing set* [12], if each variable-node in V has fewer neighbors in $O(V)$ than in $E(V)$. Fig. 1(a) shows an example of absorbing set, where each variable-node of V is connected to one check-node in $O(V)$ and two check-nodes in $E(V)$. While absorbing sets are combinatorial substructures of the Tanner graph, defined independently of the particular decoding algorithm, they are known to be particularly harmful to BP or other forms of message-passing decoding. Indeed, assuming that the set of errors V is an absorbing set, then each variable-node in V has less neighbor check-nodes indicating an error (unsatisfied), than indicating no error (satisfied). Consequently, V represents a *difficult error event*, yielding a decoding failure with high probability.

We are interested in classifying error events with a given number of errors ν. Let V a set of variable-nodes, with $\mathbf{card}(V)=\nu$. We define the *absorbing type* of V as the pair (ω,ε), where $\omega := \mathbf{card}(O(V))$ and $\varepsilon := \mathbf{card}(E(V))$. We shall sometimes denote the absorbing type as ν-(ω,ε), to also account for the cardinality of V. Note that the absorbing type indicates the total number of (unsatisfied, satisfied) check-nodes, in case the variable-nodes in V are in error. However, variable-node sets of same absorbing type may induce different (precisely, *non-isomorphic*) sub-graphs. Such an example is illustrated in Fig. 1, for two variable-node sets of absorbing type 3-$(3,3)$, the first of which is an absorbing set (a), but not the second (b). For the set V in (b), it can be seen that variable-node n is connected to two unsatisfied check-nodes and one satisfied (most favorable case among the three variable-nodes), n' to only satisfied check-nodes (worst case),



(a) Absorbing set case (b) Non-absorbing set case

Fig. 1. Example of two sets V, with $\mathbf{card}(O(V)) = \mathbf{card}(E(V)) = 3$. If variable-nodes in V are in error, check-nodes marked by an U are unsatisfied, while those marked by an S are satisfied.

and n'' to one unsatisfied and two satisfied check-nodes. If variable-node n gets corrected, V reduces to an absorbing set of type 2-$(2,2)$, determined by n' and n''. However, in general there is no guarantee that variable-node n can be decoded by the BP decoder (this will depend on the noise model, and the actual noise realization), thus we consider the case (b) as an intermediate case, lying between the 3-$(3,3)$ absorbing set case (a) and the 2-$(2,2)$ absorbing set case.

Accordingly, we define the *error class* ν-(ω,ε) as comprising all the error events, whose underlying variable-node error set V has absorbing type ν-(ω,ε). We further partition the above error class into *error sub-classes*, with each sub-class corresponding to variable-node error sets V of absorbing type ν-(ω,ε), and inducing isomorphic sub-graphs. Sub-classes are denoted by ν-(ω,ε,s), where s denotes the sub-class index. Accordingly, the variable-node sets illustrated in Fig. 1 are associated with the sub-classes 3-$(3,3,1)$ and 3-$(3,3,2)$. In the sequel, we shall simply refer to ν-(ω,ε,s) as error classes (rather than sub-classes), since no confusion is possible.

B. Proposed parallel BP-RNN decoders

To improve the decoding performance of LDPC codes at short coding length, we propose to *specialize* (*i.e.*, train) a BP-RNN decoder for each error class ν-(ω,ε,s), according to the classification from the previous subsection. The number of different error classes, and thus of BP-RNN decoders, depend on the particular Tanner graph defining the LDPC code, and the value of ν. For a given short LDPC codes, we determine all the possible error classes, by considering all the variable-node subsets V of cardinality ν. In this work, we consider $\nu = 2, 3$, thus the proposed approach is particularly relevant to high coding rate LDPC codes, correcting a small number of errors.

Once a BN-RNN decoder has been trained for each error class (the training procedure will be detailed in next subsection), we consider a parallel decoding architecture, where all the trained decoders are run in parallel (note that different architectural choices are possible and not discussed in this paper). We include the conventional BP decoder in the parallel structure. If none of the parallel decoders outputs a codeword (which is verified by computing the syndrome), decoding fails. Otherwise, among the decoded codewords, we select the one that has been outputted the most often (in case different decoders output different codewords). Decoding is successful if the selected codeword is equal to the transmitted one.

C. Training of the parallel BP-RNN decoders

We propose in this section a construction of the training set, used to train the BP-RNN decoder for a particular error class. We assume that coded bits are mapped to ± 1 modulated symbols, which undergo real additive white Gaussian noise (AWGN). Since both the noise model and the BP-RNN decoder are symmetric [8], we may assume the all-zero codeword is transmitted, corresponding to an all $+1$ modulated signal. Hence, under the AWGN model, received symbols are given by $y_n = 1 + z_n$, $n = 1, \ldots, N$, where z_n denotes a real-valued normal distributed random variable, with mean 0, and variance σ^2.

To generate a random error event in a given error class ν-(ω,ε,s), we first consider an underlying variable-node error set V, randomly chosen from those corresponding to the given error class, and then generate received symbols y_n, by

$$y_n = 1 + z_n, \quad \forall n = 1, \ldots, N \tag{6}$$

$$\text{where } z_n \sim \begin{cases} \mathcal{N}(0,\sigma^2,-\infty,-1), & \text{if } n \in V \\ \mathcal{N}(0,\sigma^2,-1,\infty), & \text{otherwise} \end{cases} \tag{7}$$

where $\mathcal{N}(0,\sigma^2,a,b)$ denotes the truncated normal distribution with mean 0 and variance σ^2, taking values in the interval (a,b). The training set is obtained by repeating the above procedure multiple times, for each variable-node set V in the given error class. In this way, the training set is representative of the error class, and thus the trained BP-RNN decoder becomes specialized to error events in the class.

D. Complementary selection of trained BP-RNNs

In practical applications, it is desirable to reduce the number of decoders running in parallel. To this end, in this subsection we propose a complementarity metric between the trained decoders. Intuitively, two decoders are complementary if the probability to fail on the same error event is low. This maximizes the gain when the two decoders are run in parallel.

We define the *similarity rate* between two decoders $\mathcal{D}_p$ and $\mathcal{D}_q$, $p \neq q$, as the probability that both decoders fail, when either one of them fails. Precisely, we define

$$S(\mathcal{D}_p,\mathcal{D}_q) := \Pr\left(\mathcal{D}_p \text{ and } \mathcal{D}_q \text{ fail} \mid \text{either } \mathcal{D}_p \text{ or } \mathcal{D}_q \text{ fails}\right) \tag{8}$$

In practice, this metric can be numerically estimated by Monte-Carlo simulation, using

$$S(\mathcal{D}_p,\mathcal{D}_q) \approx \frac{N_{p,q}}{N_p + N_q - N_{p,q}}, \tag{9}$$

where N_p (resp. N_q) is the number of times the decoder $\mathcal{D}_p$ (resp. $\mathcal{D}_q$) failed, and $N_{p,q}$ is the number times they both failed. Consequently, $S(\mathcal{D}_p,\mathcal{D}_q)$ measures the effectiveness of specialized training in producing decoders able to correct different error events. Put differently, it characterizes the level of complementary between the two decoders.

TABLE I
PARAMETERS OF THE CONSTRUCTED CODES

	$\mathbf{N}$	$\mathbf{K}$	$\mathbf{R_c}$	$\mathbf{d_v}$	$\mathbf{d_c}$	$\mathbf{n_{4\text{-cycles}}}$
Code-1	64	46	0.71	3	10-11	47
Code-2	128	105	0.81	3	16-17	1130

An overall similarity coefficient is then calculated for each decoder $\mathcal{D}_p$, by averaging the similarity rate between $\mathcal{D}_p$ and the other decoders.

$$\overline{S}(\mathcal{D}_p) = \frac{1}{D-1} \sum_{q \neq p} S(\mathcal{D}_p, \mathcal{D}_q) \tag{10}$$

where D denotes the total number of parallel BP-RNN decoders. Subsequently, given a similarity threshold value S_{th}, we keep only the decoders with overall similarity coefficient $\overline{S}(\mathcal{D}_p)$ less than S_{th}. Hence, only the most complementary decoders are maintained in the final parallel structure. We note that the choice of the S_{th} value may yield different trade-offs between complexity and decoding performance.

IV. NUMERICAL RESULTS

A. Simulation Settings

Two LDPC codes with regular variable-node degree have been considered in our simulations. Code parameters are provided in Table I, where N denotes the code length, K the number of information bits, $R_c := K/N$ the coding rate, d_v the variable nodes degree, d_c the check nodes degree, and $n_{4\text{-cycles}}$ the number of length-4 cycles. The Tanner graphs of the two codes have been constructed by using the Progressive Edge Growth (PEG) algorithm [13], a greedy algorithm making the best-effort to reduce the number of short cycles in the constructed graph. Yet, for small code length, short cycles, including cycles of length-4, cannot be completely avoided.

The number of decoding iterations was set to ten, for all the decoders. Since we consider codes with high coding rate, the error events classification was conducted for error sets V of size $\nu = 2, 3$ [1]. Applying the error classification procedure described in Section III-A, we found three 2-(ω, ε, s) classes, for both Code-1 and Code-2, and ten (resp. eleven) 3-(ω, ε, s) error classes for Code-1 (resp. Code-2). Thus, a total of thirteen (resp. fourteen) BP-RNNs were used in parallel, alongside the BP decoder. Each BP-RNN was trained independently with the training set construction technique described in Section III-C.

The same procedure was repeated for the BP-RNN-FHI decoder, for both Code-1 and Code-2, in order to assess the weight reduction in (4). In addition, we also trained a single BP-RNN decoder according to the procedure described in [8], to provide a benchmark for our parallel BP-RNNs approach.

To train the BP-RNN decoders, we used the Keras library, with the hyper parameters shown in Table II. All BP-RNNs were trained for each signal-to-noise ratio (SNR) value ranging from $1\,\text{dB}$ to $8\,\text{dB}$, with a step of $1\,\text{dB}$, thus providing eight optimized weight sets for each decoder. While this increases the training complexity, it also improves the decoding performance, as compared to the case where only one training is performed, mixing together with all SNR values. During the simulations (*i.e.*, test stage, after the training was performed), each weight set was used for the corresponding SNR value, except near 9 and $8.5\,\text{dB}$ where the weights of $8\,\text{dB}$ were used.

[1]For comparison, a BCH code with either $(N, K) = (63, 45)$ or $(N, K) = (127, 106)$ has minimum distance $d = 7$, thus may correct 3 errors

TABLE II
KERAS PARAMETERS

Parameters	**Parameters values**
Optimizer (Gradient descent)	RMSprop [14] (initialized at a learning rate of 10^{-3})
Epoch number	10
Training batch size	8192
Testing batch size	16384

Finally, we used the similarity metric introduced in Section III-D, in order to reduce the number of parallel decoders. The similarity coefficient has been numerically estimated by using (9) and (10), based on the first simulation results. We fixed a similarity threshold value $S_{\text{th}} = 0.6$ (resp. $S_{\text{th}} = 0.56$) for Code-1 (resp. Code-2), at a FER of 10^{-4}, which led to the selection of only eight (resp. nine) trained BP-RNN decoder alongside the conventional BP. Then, new independent simulations were run for both resulting parallel structures, to assess their FER performance.

B. FER performance

We compare the different decoding strategies discussed in the previous section, in terms of FER. Reported SNR gains are evaluated at a FER of 10^{-4}.

Simulation results for Code-1 are shown in Fig. 2. The BP-RNN trained as in [8] yields an SNR gain of $0.18\,\text{dB}$, with respect to conventional BP. Using the proposed structure, with specialized BP-RNNs, the SNR gain is increased to $0.42\,\text{dB}$. The parallel BP-RNN-FHIs exhibit only negligible performance degradation compared to the parallel BP-RNNs, due to the weight reduction constraint in (4). Finally, it can be observed that the parallel BP-RNNs construction with complementary selection shows virtually the same FER performance as the original complete construction of parallel BP-RNNs. Therefore, choosing $S_{\text{th}} = 0.6$ leads to a selection of eight decoders which effectively complement each others.

Simulation results for Code-2 are shown in Fig. 3. Despite the increase in the length of the code, it should be noted that Code-2 exhibits an increased number of length-4 cycles, due to its higher coding rate. However the parallel BP-RNN decoder yields a similar SNR gain, of $0.42\,\text{dB}$, with respect to the conventional BP. The parallel BP-RNN-FHIs tend to be a little less efficient. Furthermore, Fig. 3 also corroborates the relevance of the complementary selection, the corresponding parallel BP-RNNs construction yielding again almost the same performance as the original complete construction of parallel BP-RNNs.

Finally, Fig. 4 provides a comparison in terms of the number of decoding failures, for various values of ν (size of the error event), for an SNR of $8\,\text{dB}$. It demonstrates that for both

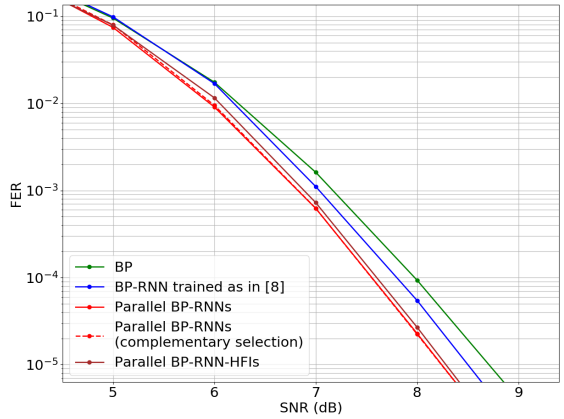


Fig. 2. FER results for Code-1.

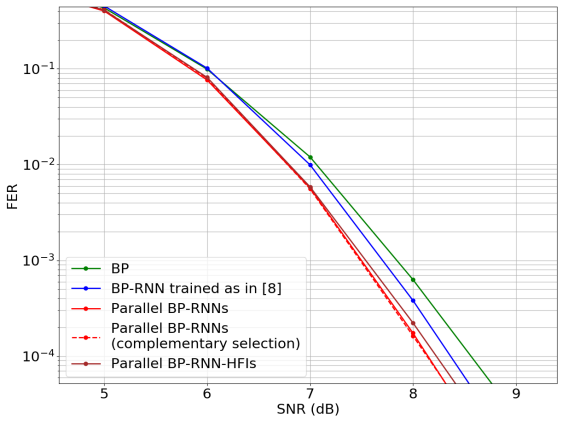
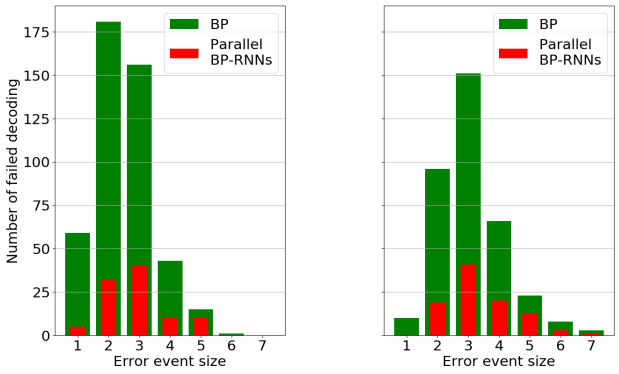


Fig. 3. FER results for Code-2.



(a) Code 1 (at 8dB). (b) Code 2 (at 8dB).

Fig. 4. Number of failed decoding according to the size of an error event.

Code-1 and Code-2, the parallel BP-RNNs are clearly able to correct numerous error events which are not decoded by the BP, especially for the two and three error events ($\nu = 2, 3$). Therefore, we conclude that the specialization of the training for the 2-(ω, ε, s) and the 3-(ω, ε, s) error classes effectively induces an understanding of the BP-RNNs of how to decode several type of size two and three error events. Furthermore, some error events of size four and five are also successfully decoded thanks to the previous specialization.

V. Conclusion and Perspectives

In this paper, we addressed the problem of enhancing the BP-RNN performance at short coding length. To this end, we studied and classified error events according to the impact of the induced sub-graphs on the BP decoding performance. Then, we proposed a new decoding strategy consisting of parallel specialized BP-RNN decoders where each BP-RNN was trained for a specific error class. In addition, we introduced a complementary selection method of the trained BP-RNN decoders, proven to be efficient in keeping the most relevant trained decoders in the final parallel structure.

This work is a first step towards a framework of specialized neural BP decoders, and we believe that further work may reveal alternative specialization strategies. The final aim would be to approach maximum likelihood decoding performance at short to moderate code-length, for which we will probably need to rely on a bunch of practical decoders, rather than a unique one.

References

[1] Y. Polyanskiy, H. V. Poor, and S. Verdú, "Channel coding rate in the finite blocklength regime," *IEEE Transactions on Information Theory*, vol. 56, no. 5, pp. 2307–2359, 2010.

[2] M. C. Coşkun, G. Durisi, T. Jerkovits, G. Liva, W. Ryan, B. Stein, and F. Steiner, "Efficient error-correcting codes in the short blocklength regime," *Physical Communication*, vol. 34, pp. 66–79, 2019.

[3] R. G. Gallager, "Low density parity check codes," MIT Press, Cambridge, 1963, research Monograph series.

[4] R. Tanner, "A recursive approach to low complexity codes," *IEEE Transactions on Information Theory*, vol. 27, no. 5, pp. 533–547, 1981.

[5] T. Richardson, M. Shokrollahi, and R. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Trans. on Information Theory*, vol. 47, no. 2, pp. 619–637, 2001.

[6] N. Wiberg, "Codes and decoding on general graphs," Ph.D. dissertation, Department of Electrical Engineering, Linköping University, Sweden, 1996.

[7] I. Dimnik and Y. Be'ery, "Improved random redundant iterative HDPC decoding," *IEEE Transactions on Communications*, vol. 57, no. 7, pp. 1982–1985, 2009.

[8] E. Nachmani, E. Marciano, L. Lugosch, W. J. Gross, D. Burshtein, and Y. Be'ery, "Deep learning methods for improved decoding of linear codes," *IEEE Journal of Selected Topics in Signal Processing*, vol. 12, no. 1, pp. 119–131, 2018.

[9] X. Xiao, B. Vasić, R. Tandon, and S. Lin, "Designing finite alphabet iterative decoders of ldpc codes via recurrent quantized neural networks," *IEEE Trans. on Communications*, vol. 68, no. 7, pp. 3963–3974, 2020.

[10] A. Buchberger, C. Häger, H. D. Pfister, L. Schmalen, and A. G. i Amat, "Pruning neural belief propagation decoders," in *IEEE International Symposium on Information Theory (ISIT)*, 2020, pp. 338–342.

[11] N. Doan, S. A. Hashemi, E. N. Mambou, T. Tonnellier, and W. J. Gross, "Neural belief propagation decoding of crc-polar concatenated codes," in *IEEE Int. Conference on Communications (ICC)*, 2019, pp. 1–6.

[12] L. Dolecek, P. Lee, Z. Zhang, V. Anantharam, B. Nikolic, and M. Wainwright, "Predicting error floors of structured ldpc codes: Deterministic bounds and estimates," *IEEE Journal on Selected Areas in Communications*, vol. 27, no. 6, pp. 908–917, 2009.

[13] X.-Y. Hu, E. Eleftheriou, and D.-M. Arnold, "Regular and irregular progressive edge-growth Tanner graphs," *IEEE Transactions on Information Theory*, vol. 52, no. 51, pp. 386–398, 2005.

[14] T. Tieleman and G. Hinton, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural networks for machine learning*, vol. 4, no. 2, pp. 26–31, 2012.