Fast Window Decoding of BMST Codes via Step Adjustment and Universal Parity-Check Termination

Viet Hoang Le*, Jasper Brüggmann*, Philipp Mohr*, Gerhard Bauch
Institute of Communications
Hamburg University of Technology
Hamburg, Germany
{viet.le; jasper.brueggmann; philipp.mohr; bauch}@tuhh.de
* These authors contributed equally.

***Abstract*—Block Markov Superposition Transmission (BMST) codes are a class of spatially coupled codes, that achieve excellent error correction performance via superposition of simple constituent codes. A BMST encoder consists of an outer basic code and an inner rate-one block-convolutional shift register. The resulting graph representation of the encoder allows sliding-window decoding, similarly to spatially coupled Low-Density Parity-Check codes. This paper proposes a method to reduce the decoding complexity by adjusting the step size of the sliding window. Furthermore, a low-complexity parity-check-based early termination criterion is introduced, which can be universally applied to recursive BMST structures.**

***Index Terms*—Spatially coupled codes, block Markov superposition transmission, sliding-window decoding.**

I. Introduction

Lasting efforts in pushing boundaries of communication throughput have led to the development of several high-performance code families, such as Turbo codes [1], Low-Density Parity-Check (LDPC) codes [2] and Spatially Coupled LDPC (SC-LDPC) [3] codes. These high-throughput forward error correction schemes rely on graph-based iterative message passing decoding to achieve excellent error rate performance.

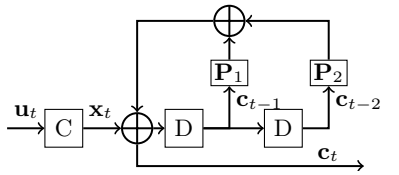
More recently, Block Markov Superposition Transmission (BMST) codes have been introduced as another class of spatially coupled codes, which offers attractive error rate performance with a flexible code construction [4]. BMST codes rely on transmitting superimposed code words of a simple block code, in the following referred to as *basic code*. Spatial coupling of basic code words is achieved by a rate-one block-convolutional encoder as shown in Fig. 1.

Typically, a BMST decoder employs an *iterative sliding-window decoding algorithm* based on message passing over the normal graph representation (see Fig. 2) of the generator matrix [4]. This is similar to sliding-window decoding of SC-LDPC codes [5], where decoding is performed on the parity-check matrix. Three widely studied BMST classes are non-recursive BMST [4], recursive BMST (rBMST) [6], and Doubly-recursive BMST (DrBMST) codes [7]. The non-recursive structure requires large encoding memory and a long decoding window to approach capacity [8]. These requirements are impractical for hardware implementations due to long decoding latency, high memory usage and significant interconnect complexity. Building upon the performance gains achieved by recursive structures in Turbo codes [9], rBMST codes were proposed in [6], where the performance improves substantially with lower encoding memory compared to non-recursive structures. For instance, rBMST codes with memory 3 and window size 11 achieve similar threshold performance to non-recursive BMST codes with memory 12 and window size 24 [6]. Further reduction of required encoding memory and decoding window size was shown in [7] by introducing DrBMST codes. Specifically, a DrBMST ensemble with encoding memory 2 and decoding window size 7 achieves a decoding threshold that is $0.06\,\mathrm{dB}$ better than that of rBMST codes with memory 3 and window size 11 [7].

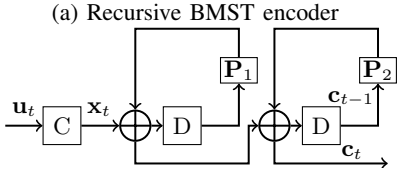
Due to the attractive performance with reduced complexity, rBMST and DrBMST codes are considered. While the asymptotic performance of recursive BMST code classes is analyzed in [6], [7], this paper focuses on the error rate performance in the finite-length code word regime and on the reduction of computational decoding complexity. The main contributions of this work are summarized as follows:

1) We propose the *window step method* to reduce the computational complexity of the conventional sliding-window decoder by moving the window in larger steps than one. This way, the performance-complexity trade-off of the BMST system can be adjusted with a single parameter that determines the step size.
2) We introduce a universal early stopping criterion for the iterative decoder that relies on the parity-check constraints defined by the Single Parity-Check (SPC) nodes present in any recursive BMST graph. This method outperforms the widely-used entropy-based stopping criterion [4] in terms of average computational complexity.
3) In order to compare rBMST and DrBMST codes to 5G LDPC codes in terms of computational complexity and error rate performance, a complexity analysis of the respective decoding algorithms is presented. We show that DrBMST codes can achieve a better performance-complexity trade-off than 5G LDPC codes with layered Belief Propagation (BP) decoding under an equal decoding latency constraint.

This work is supported by Huawei Technologies Sweden AB.



(a) Recursive BMST encoder



(b) Doubly-recursive BMST encoder

Fig. 1: Two variants of recursive BMST encoders with encoding memory $m = 2$.

II. System Overview

A. Recursive BMST Codes

Let $\mathrm{C}[N_\mathrm{C}, K_\mathrm{C}]$ be a binary linear block code of length N_C, dimension K_C, and rate $R_\mathrm{C} = {}^{K_\mathrm{C}}/_{N_\mathrm{C}}$, in the following referred to as *basic code*. Given a long block of information bits $\mathbf{u}$, it is partitioned into a sequence of L sub-blocks denoted as $\mathbf{u} = [\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_{L-1}]$ with $\mathbf{u}_t \in \mathbb{F}_2^{K_\mathrm{C}}$. The parameter L is commonly referred to as the coupling length in the context of spatially coupled codes. To ensure good error protection of the final sub-blocks, T termination blocks are appended at the end of the sequence. These are defined as $\mathbf{u}_t = \mathbf{0}$ for $L \leq t \leq L + T - 1$. The encoding algorithm is given in [6, Algorithm 1].

An rBMST encoder with encoding memory $m = 2$ is shown in Fig. 1a. A basic code word $\mathbf{x}_t \in \mathbb{F}_2^{N_\mathrm{C}}$ results from applying the basic code to $\mathbf{u}_t$. Subsequently, a rate-one block-convolutional shift register with m memory elements superimposes $\mathbf{x}_t$ and m previously transmitted blocks as

$$\mathbf{c}_t = \mathbf{x}_t \oplus \mathbf{P}_1\mathbf{c}_{t-1} \oplus \cdots \oplus \mathbf{P}_m\mathbf{c}_{t-m}, \tag{1}$$

where $\mathbf{P}_i$ are random permutation matrices for $i \in \{1, ..., m\}$.

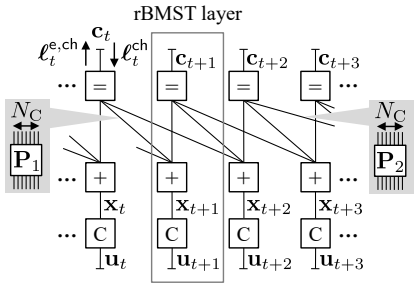
Similarly, the DrBMST encoder consists of the basic encoder and two serially concatenated recursive shift registers with memory 1 as shown in Fig. 1b [7]. The corresponding rBMST and DrBMST graph representations are shown in Fig. 2 in normal graph notation [10]. The BMST graph consists of Repetition (REP) nodes $\boxed{=}$, SPC nodes $\boxed{+}$ and basic code nodes $\boxed{\mathrm{C}}$. Note that we implicitly integrate the permutations $\mathbf{P}_i$ in the edges of the graph for brevity, as they can be considered as routing networks of width N_C.

The overall rate of the BMST system is determined by the rate of the basic code R_C and a rate loss factor R_loss due to termination, and is given as

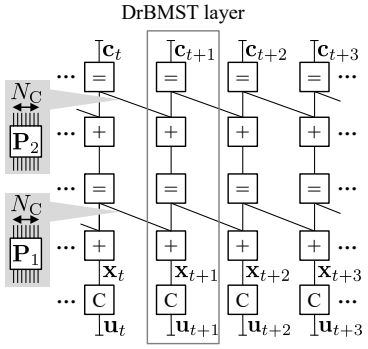
$$R = \frac{K_\mathrm{C}}{N_\mathrm{C}} \cdot \frac{L}{L+T} = R_\mathrm{C} R_\mathsf{loss} \tag{2}$$

with R_loss approaching 1 for a large coupling length L.

The BMST decoder processes received channel Log-Likelihood Ratio (LLR) vectors $\boldsymbol{\ell}_t^\mathsf{ch} = 4\mathbf{y}_t/N_0$ with



(a) Recursive BMST graph



(b) Doubly-recursive BMST graph

Fig. 2: Normal graph representation of two BMST variants with encoding memory $m = 2$.

$\mathbf{y}_t = 1 - 2\mathbf{c}_t + \mathbf{n}_t$, Additive White Gaussian Noise (AWGN) $\mathbf{n}_t \in \mathbb{R}^{N_\mathrm{C}}$ and noise variance $N_0/2$. The n-th element of $\boldsymbol{\ell}_t^\mathsf{ch}$ is defined as $\ell_{t,n}^\mathsf{ch} = \log \frac{p(y_{t,n}|c_{t,n}=0)}{p(y_{t,n}|c_{t,n}=1)}$.

B. Sliding-Window Decoding

Typically, an *iterative sliding-window decoding* algorithm is employed for BMST codes (see [4, Algorithm 3]). In this approach, iterative message passing decoding is performed in a window consisting of W layers to decode an information block $\mathbf{u}_t$, where $\mathbf{u}_t$ corresponds to the first layer in the window.

The decoding process within the window is performed for I_W forward-backward iterations across W layers or until an early stopping criterion is satisfied. Each iteration consists of $W - 1$ forward layer updates followed by $W - 1$ backward layer updates. As described in [6], the update schedule of an rBMST layer is defined as the following node update sequence $\boxed{=} \rightarrow \boxed{+} \rightarrow \boxed{\mathrm{C}} \rightarrow \boxed{+} \rightarrow \boxed{=}$, while a DrBMST layer update is described by $\boxed{=} \rightarrow \boxed{+} \rightarrow \boxed{=} \rightarrow \boxed{+} \rightarrow \boxed{\mathrm{C}} \rightarrow \boxed{+} \rightarrow \boxed{=} \rightarrow \boxed{+} \rightarrow \boxed{=}$ [7]. Algorithm 1 summarizes the computation steps in forward and backward recursion for an rBMST layer with $f_C(\cdot)$ describing a soft-in soft-out decoding function of the basic code. We denote $\boldsymbol{\ell}_\tau^{\mathrm{A}_i\mathrm{B}_j}$ as the LLR messages from node A at layer $\tau + i$ to node B at layer $\tau + j$, and $\boldsymbol{\ell}_\tau^{\mathrm{A}_i}$ as the result of the node operation on all incoming messages of node A at layer $\tau + i$.

The modified decoding algorithm including the contributions of this work is illustrated by Algorithm 2 and will be described further in Section III.

Algorithm 1 Message Updates for rBMST Structure.

procedure Initialize($\boldsymbol{\ell}_0^{\mathsf{ch}}, \ldots, \boldsymbol{\ell}_{L+T-1}^{\mathsf{ch}}$)
  $\boldsymbol{\ell}_t^{\mathsf{s}_i\mathsf{r}_0}, \boldsymbol{\ell}_t^{\mathsf{r}_0\mathsf{s}_i} \leftarrow \mathbf{0} \quad \forall (t,i) \in \{0,...,L+T-1\} \times \{0,...,m\}$
  $\boldsymbol{\ell}_t^{\mathsf{r}_0} \leftarrow \boldsymbol{\ell}_t^{\mathsf{ch}} \quad \forall t \in \{0,...,L+T-1\}$

procedure RepToBasicCodeUpdates(τ)
  $\boldsymbol{\ell}^{\mathsf{r}_0\mathsf{s}_0} \leftarrow \boldsymbol{\ell}^{\mathsf{r}_0} - \boldsymbol{\ell}^{\mathsf{s}_0\mathsf{r}_0}$ ▷ $\boxed{=} \rightarrow \boxed{+}$
  $\boldsymbol{\ell}^{\mathsf{s}_0\mathsf{b}_0} \leftarrow \boldsymbol{\ell}^{\mathsf{r}_0\mathsf{s}_0} \boxplus \mathbf{P}_1\boldsymbol{\ell}^{\mathsf{r}_{-1}\mathsf{s}_0} \boxplus \cdots \boxplus \mathbf{P}_m\boldsymbol{\ell}^{\mathsf{r}_{-m}\mathsf{s}_0}$ ▷ $\boxed{+} \rightarrow \boxed{\mathrm{C}}$
  $\boldsymbol{\ell}^{\mathsf{b}_0\mathsf{s}_0} \leftarrow f_C(\boldsymbol{\ell}^{\mathsf{s}_0\mathsf{b}_0})$ ▷ $\boxed{\mathrm{C}} \rightarrow \boxed{+}$
  $\boldsymbol{\ell}^{\mathsf{s}_0} \leftarrow \boldsymbol{\ell}^{\mathsf{s}_0\mathsf{b}_0} \boxplus \boldsymbol{\ell}^{\mathsf{b}_0\mathsf{s}_0}$ ▷ Compute APP sum for node s_0.

procedure ForwardUpdate(τ)
  RepToBasicCodeUpdates(τ)
  $\boldsymbol{\ell}^{\mathsf{s}_0\mathsf{r}_0} \leftarrow \boldsymbol{\ell}^{\mathsf{s}_0} \boxminus \boldsymbol{\ell}^{\mathsf{r}_0\mathsf{s}_0}$ ▷ $\boxed{+} \rightarrow \boxed{=}$
  $\boldsymbol{\ell}^{\mathsf{r}_0} \leftarrow \boldsymbol{\ell}^{\mathsf{r}_0\mathsf{s}_0} + \boldsymbol{\ell}^{\mathsf{s}_0\mathsf{r}_0}$ ▷ Compute APP sum for node r_0.
  $\boldsymbol{\ell}^{\mathsf{r}_0\mathsf{s}_i} \leftarrow \boldsymbol{\ell}^{\mathsf{r}_0} - \boldsymbol{\ell}^{\mathsf{s}_i\mathsf{r}_0} \quad \forall i \in \{1,...,m\}$ ▷ $\boxed{=} \rightarrow \boxed{+}$

procedure BackwardUpdate(τ)
  RepToBasicCodeUpdates(τ)
  $\boldsymbol{\ell}^{\mathsf{s}_0\mathsf{r}_{-i}} \leftarrow \mathbf{P}_i^{-1}(\boldsymbol{\ell}^{\mathsf{s}_0} \boxminus \mathbf{P}_i\boldsymbol{\ell}^{\mathsf{r}_{-i}\mathsf{s}_0}) \quad \forall i \in \{1,...,m\}$ ▷ $\boxed{+} \rightarrow \boxed{=}$
    ▷ Compute APP sum for nodes r_{-i}.
  $\boldsymbol{\ell}^{\mathsf{r}_{-i}} \leftarrow \boldsymbol{\ell}^{\mathsf{r}_{-i}\mathsf{s}_0} + \boldsymbol{\ell}^{\mathsf{s}_0\mathsf{r}_{-i}} \quad \forall i \in \{1,...,m\}$
  ▷ $\boldsymbol{\ell} \triangleq \boldsymbol{\ell}_\tau, \mathsf{r} \triangleq \mathsf{REP}, \mathsf{s} \triangleq \mathsf{SPC}, \mathsf{b} \triangleq \mathsf{BC}, \psi(x) = \tanh(x/2)$
  ▷ $\ell_1 \boxplus \ell_2 = \psi^{-1}(\psi(\ell_1) \cdot \psi(\ell_2))$
  ▷ $\ell_1 \boxminus \ell_2 = \psi^{-1}(\psi(\ell_1)/\psi(\ell_2))$

Algorithm 2 Sliding-Window Decoding of rBMST Codes.

Input: $[\boldsymbol{\ell}_0^{\mathsf{ch}}, \ldots, \boldsymbol{\ell}_{L+T-1}^{\mathsf{ch}}], W, I_{\mathsf{W}}, \tau_{\mathsf{S}}, S_{\mathsf{W}}$.
Output: $[\hat{\mathbf{u}}_0, \ldots, \hat{\mathbf{u}}_{L-1}]$
Initialize($\boldsymbol{\ell}_0^{\mathsf{ch}}, \ldots, \boldsymbol{\ell}_{L+T-1}^{\mathsf{ch}}$)
$t \leftarrow 0$ ▷ Index of first layer in a window.
$w \leftarrow 2$ ▷ Initial window size is 2.
while $t \leq L-1$ **do**
  $\iota \leftarrow 0$ ▷ Iteration counter for current window position.
  while $\iota < I_{\mathsf{W}}$ and not ConvCheck($t + \tau_{\mathsf{S}}$) **do**
    for $\omega \in (0, \ldots, w-1)$ **do**
      ForwardUpdate($t + \omega$)
    for $\omega \in (w, \ldots, 1)$ **do**
      BackwardUpdate($t + \omega$)
    $\iota \leftarrow \iota + 1$
  ▷ Manage window size at start and end layers.
  if $w < W$ **then** ▷ Increase w at beginning of graph.
    $w \leftarrow \min(w + S_{\mathsf{W}}, W)$
  else if $t + w \geq L + T - 1$ **then** ▷ Reduce w at end of graph.
    $w \leftarrow w - S_{\mathsf{W}}$
    ▷ Compute hard decisions and advance window by S_{W} layers.
  if $w = W$ or $t > 0$ **then**
    Compute $\left[\hat{\mathbf{u}}_t, \ldots, \hat{\mathbf{u}}_{\min(t+S_{\mathsf{W}}, L-1)}\right]$.
    $t \leftarrow t + S_{\mathsf{W}}$

C. Multi-Rate Code Construction via Time-Sharing Method

A simple method of achieving multi-rate rBMST code construction was introduced in [6]. The core idea of the *time-sharing* approach is to construct a basic code $\mathrm{C}[N_{\mathrm{C}}, K_{\mathrm{C}}]$ with a desired rate $R_{\mathrm{C}} = {}^{K_{\mathrm{C}}}/_{N_{\mathrm{C}}}$ by combining two constituent codes of similar structure but different rates. For rates that cannot be achieved using a repetition code or a single parity-check code, i.e., $R_{\mathrm{C}} = {}^1/_N$ or $R_{\mathrm{C}} = {}^{(N-1)}/_N$ for $N \in \mathbb{Z}^+$, we construct the basic code by encoding a fraction of the information bits using a lower-rate code, and the remaining bits with a higher-rate code. A target rate $R_{\mathrm{C}} \in (0, {}^1/_2]$ can be obtained by using two repetition codes, whereas rates $R_{\mathrm{C}} \in [{}^1/_2, 1)$ can be obtained from two single parity-check codes. This multi-rate construction achieves excellent performance while the decoding complexity of the basic code remains very low [6]. In this paper, the time-sharing method will be used to demonstrate the proposed decoder modifications for several code rates.

III. Proposed Low-Complexity Window Decoding

As briefly discussed in Section II-B, the sliding-window decoder for BMST codes performs iterative message passing in a decoding window spanning across W layers until a stopping criterion is satisfied or the maximum number of iterations I_{W} is reached. Then, the decoding window is shifted by one layer position. Especially at high E_b/N_0, the stopping criterion significantly reduces the average computational complexity without performance degradation as shown in Section IV. Without early termination, most iterations do not yield any further benefit in high E_b/N_0 region, shown in Fig. 4. Doubling I_{W} from 5 to 10 provides an additional $0.02\,\mathrm{dB}$ gain, while further increasing to $I_{\mathsf{W}} = 15$ offers $0.01\,\mathrm{dB}$ improvement but increases the complexity by 50%. In other words, many computations are effectively wasted, thus avoiding them by early stopping is crucial for reducing complexity.

A. Window Step Method

To reduce computational complexity and explore new performance-complexity trade-off regimes with a predictable decoding schedule, we propose modifying the sliding-window decoder such that the window can advance in steps $S_{\mathsf{W}} > 1$ as illustrated in Algorithm 2. Thus, S_{W} information block estimates $\hat{\mathbf{u}}_t$ are computed per window position and the maximum number of layer updates is reduced by the factor S_{W}.

Sliding-window decoding of BMST codes relies on wave-like convergence [11]. The decoding wave denotes the boundary between the already-converged layers and the yet-to-be-decoded layers, i.e., the layer position where bit error probability rises. Due to the known encoder initialization, the decoding wave starts at the first layer and propagates through the graph. In order to provide low error rates, the window must not lose track of the decoding wave. Otherwise, the remaining layers will inhibit high error probabilities.

A poorly chosen stopping criterion may lead to premature termination and insufficient convergence within the decoding window. Consequently, the window may proceed faster than the decoding wave propagates, especially for larger S_{W}. To avoid this, we propose to perform the convergence check for a window position t at delayed layer position $t+\tau_{\mathsf{S}}$, where τ_{S} is referred to as the convergence check position within a window. This is indicated in Algorithm 2. We observed in Section IV-B that this significantly improves the error rate performance by

allowing slightly more iterations and improving the decoders ability to track the decoding wave.

B. Entropy-based Early Termination

In [4], [6], early termination during sliding-window decoding is achieved using an entropy-based stopping criterion. Decoding is stopped once the changes of the exchanged messages from one iteration to the next are sufficiently small. For this purpose, the entropy rate $h_\iota(\mathbf{Y}_\tau)$ with random vector $\mathbf{Y}_\tau$ corresponding to the received vector $\mathbf{y}_\tau$ at layer τ, is estimated at every iteration ι as shown in $\mathrm{ConvCheckEntropy}$ of Algorithm 3. Here, $\boldsymbol{\ell}_\tau^{\mathsf{e},\mathsf{ch}}$ refers to the extrinsic LLR vector computed by the repetition nodes $\boxed{=}$ towards the channel half edges (see Fig. 2a).

The window decoding for a target layer t is stopped, if $|h_\iota(\mathbf{Y}_{t+\tau_\mathsf{S}}) - h_{\iota-1}(\mathbf{Y}_{t+\tau_\mathsf{S}})| < \epsilon$ for a fixed threshold ϵ. In [4], the entropy rate is initialized as $h_0(\mathbf{Y}_t) = 0$ with $\tau_\mathsf{S} = 0$ before decoding a window position t, leading to at least two iterations even at high E_b/N_0. We propose to initialize $h_0(\mathbf{Y}_{t+\tau_\mathsf{S}})$ using the computation in $\mathrm{ConvCheckEntropy}$ of Algorithm 3, such that one window iteration can be sufficient at high E_b/N_0 to fulfill the stopping criterion.

C. Proposed Early Termination using SPC Syndromes

Looking at $\mathrm{ConvCheckEntropy}$ of Algorithm 3, the high computational complexity of the entropy-based stopping criterion is evident, especially for large N_C due to the high number of additions, exponential functions and logarithms. Moreover, it relies on monitoring the similarity of messages between successive iterations, leading to at least one required iteration.

In [4], if an outer Cyclic Redundancy Check (CRC) code is available, it can also be used for error detection purposes. However, this outer code would inherently introduce a rate loss. Alternatively, a stopping criterion using the parity-check matrix of the basic code can be employed [11]. However, such a mechanism depends on the choice of the basic code.

To reduce computational complexity and avoid any rate loss caused by an outer error-detecting code, we propose a simple early termination criterion that is universally applicable to recursive BMST structures by evaluating the constraints of SPC nodes available from the convolutional encoder. This convergence check is illustrated by $\mathrm{ConvCheckSPC}$ in Algorithm 3 and applied in Algorithm 2. After each iteration ι, the N_C syndromes of the SPC node at layer t are computed. If all N_C syndromes are fulfilled, the current window decoding is terminated and the window shifts to the next layer. Note that this method does not require comparing results of different iterations. Thus, the number of required iterations can reach zero at high E_b/N_0, if the first convergence check is done before the first iteration.

Similarly to the entropy-based method, the SPC-based criterion is agnostic to the choice of the basic code. For DrBMST, we observed that it is sufficient to evaluate the constraints of only one SPC node per layer. Neither criterion is used as a proper error detection mechanism, but serves as an indicator for convergence of the decoding window.

Algorithm 3 Early Stopping Criteria of rBMST Codes.

function $\mathrm{ConvCheckSPC}(\tau)$
  $\mathbf{s} \leftarrow \mathrm{sgn}(\boldsymbol{\ell}_\tau^{\mathsf{SPC}_0\mathsf{BC}_0}) \circ \mathrm{sgn}(\boldsymbol{\ell}_\tau^{\mathsf{BC}_0\mathsf{SPC}_0})$
  return $\mathbf{s} = +\mathbf{1}$

function $\mathrm{ConvCheckEntropy}(\tau)$
  $\boldsymbol{\ell}^{\mathsf{h}} \leftarrow \boldsymbol{\ell}_\tau^{\mathsf{ch}} + \boldsymbol{\ell}_\tau^{\mathsf{e},\mathsf{ch}} - \log\big(1+\exp\boldsymbol{\ell}_\tau^{\mathsf{ch}}\big) - \log\big(1+\exp\boldsymbol{\ell}_\tau^{\mathsf{e},\mathsf{ch}}\big)$
  $h_\iota(\mathbf{Y}_\tau) \leftarrow -\frac{1}{N_\mathrm{C}}\sum_{n=0}^{N_\mathrm{C}-1}\ell_n^{\mathsf{h}}$
  return $|h_\iota(\mathbf{Y}_\tau) - h_{\iota-1}(\mathbf{Y}_\tau)| < \epsilon$

TABLE I: Complexity metrics for rBMST and LDPC codes. d_j^{v}, d_i^{c} denote the j-th variable and i-th check node degree in a 5G LDPC base graph with lift size Z, respectively.

Avg. count / info. bit	Operations per layer update		Operations per iteration
Metrics	**rBMST**	**DrBMST**	**LDPC**
Addition	$\frac{(m+1)N_\mathrm{C}}{K_\mathrm{C}}$	$\frac{4N_\mathrm{C}}{K_\mathrm{C}}$	$\frac{Z}{K_{5\mathsf{G}}}\sum_j d_j^{\mathsf{v}}\cdot\begin{cases}2 & d_j^{\mathsf{v}}>2\\ 1 & d_j^{\mathsf{v}}=2\\ 0 & d_j^{\mathsf{v}}=1\end{cases}$
Boxplus	$\frac{(m+1)N_\mathrm{C}}{K_\mathrm{C}}$	$\frac{5N_\mathrm{C}}{K_\mathrm{C}}$	$\frac{Z}{K_{5\mathsf{G}}}\sum_i\left(2\,d_i^{\mathsf{c}}-1\right)$
Basic Code	$\frac{1}{K_\mathrm{C}}$		–

D. Decoding Complexity

A fair code comparison should include a complexity indicator for achieving a given target error rate. We consider computational complexity as *operation counts* (e.g. additions and boxplus) over the decoding process, normalized by the number of information bits.

Table I summarizes the complexity metrics for rBMST, DrBMST under sliding-window decoding and LDPC with row-layered decoding. The computational complexity of rBMST and DrBMST decoding is expressed as the number of operations per layer update and information bit, while the LDPC complexity is considered in terms of operations per iteration and information bit. The computational complexity of updating an rBMST layer is derived directly from counting operations in Algorithm 1. A similar approach can be done for DrBMST codes.

To compute the overall BMST decoding complexity, the normalized operation counts must be multiplied by the number of times a single BMST layer is updated on average. Neglecting the first and last layers, the number of updates per layer is upper-bounded by $2I_\mathsf{W}(W-1)/S_\mathsf{W}$. Note that for BMST systems with a rate-$1/2$ repetition basic code, the complexity of the basic code decoder is zero during forward and backward recursions and thus does not contribute to the overall computational complexity.

Similarly to BMST, the average number of iterations must be included in case of LDPC decoding. The maximum number of LDPC iterations is denoted as I_L. For LDPC decoding under a row-layered schedule, the number of operations per iteration depends on the degree distributions, the lifting factor Z, and is normalized by the information word length $K_{5\mathsf{G}}$.

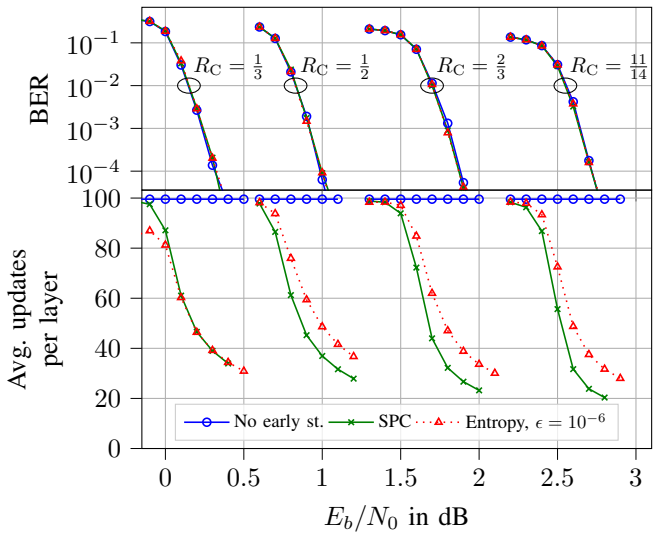


Fig. 3: Multi-rate performance of rBMST with and without early stopping. Time-sharing basic codes with $K_C \approx 1000$ are used, with $L = 110$, $T = 3$, $m = 3$, $I_W = 5$, $W = 11$, $S_W = 1$, $\tau_S = 2$.

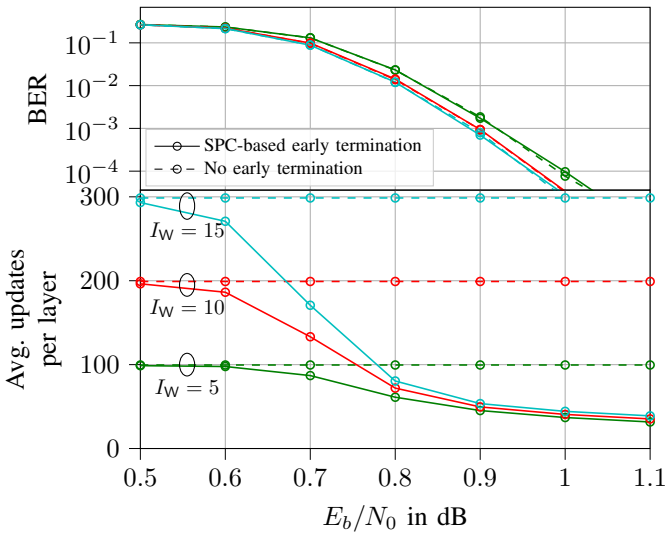


Fig. 4: rBMST with and without early stopping and varying maximum number of window iterations I_W. Parameters of Fig. 3 with $R_C = 1/2$, $K_C = 1000$ are used.

IV. Performance Evaluation

This section evaluates the methods proposed in Section III regarding Bit Error Rate (BER) performance and computational complexity.

A. SPC-based Early Stopping

In Fig. 3, the proposed SPC-based early stopping is compared to the entropy-based stopping criterion with $\epsilon = 10^{-6}$ [6] and a sliding-window decoder without early termination for time-sharing basic codes with four different rates R_C in terms of BER and average updates per layer versus E_b/N_0. Both stopping criteria achieve virtually identical BER without any noticeable performance degradation to the decoder not using early stopping.

Furthermore, SPC-based early stopping outperforms entropy-based stopping at high E_b/N_0 in terms of layer updates for the tested rates $R_C \geq 1/2$. This is due to the requirement of at least one full window iteration in the entropy-based case as explained in Section III-B. At a low rate $R_C = 1/3$, we observed that using entropy-based stopping can achieve on average less layer updates than for higher rates. Compared to SPC-based stopping, using entropy-based stopping results in approximately the same amount of layer updates for $R_C = 1/3$.

The influence of choosing certain maximum iteration counts I_W is investigated in Fig. 4. Only a minor benefit in terms of BER is observed when increasing I_W from 5 to 10 and 15. This is reflected in measured computational complexity, as the average number of layer updates quickly shrinks to similar values in the waterfall region. Consequently, at medium to high E_b/N_0, the maximum number of iterations is rarely required.

B. Window Step and Convergence Check Position

As explained in Section III-A, selecting the convergence check position τ_S within a window can influence the BER and

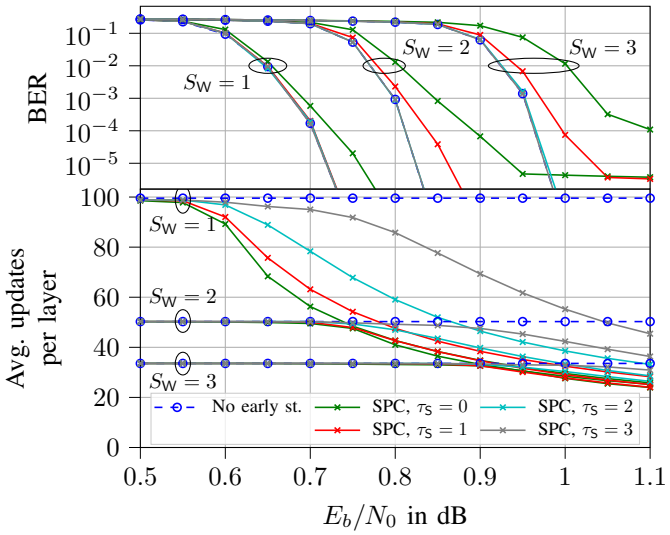


Fig. 5: rBMST with varying convergence check position τ_S. Parameters of Fig. 3 with $R_C = 1/2$, $K_C = 4000$ are used.

average number of layer updates. Fig. 5 evaluates different τ_S using SPC-based stopping for varying S_W.

Increasing the window step S_W from 1 to 2 and 2 to 3 with a properly chosen τ_S introduces a BER degradation of 0.11 dB and 0.15 dB, respectively. As explained in Section III-D, the maximum number of updates per layer scales with $1/S_W$. In the waterfall region, early stopping quickly reduces the average number of layer updates for $S_W = 1$, while the reduction is less significant for larger window steps S_W.

Clearly, performing convergence check too early at a small τ_S can degrade the BER, especially for larger S_W. In Fig. 5, $\tau_S = 1$ is sufficient for $S_W = 1$, while $\tau_S = 2$ is required for $S_W \in \{2, 3\}$ to achieve identical BER as without early stopping. Choosing an even larger τ_S, thus enforcing a higher

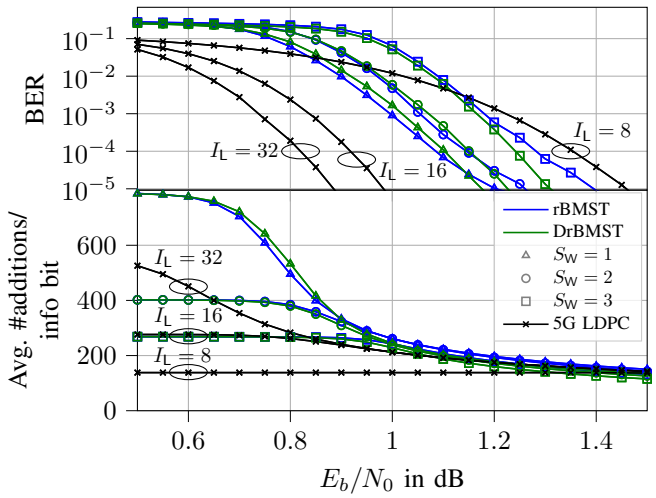


Fig. 6: Performance and average additions of rBMST and DrBMST using window step methods for rate R_C=1/2. BMST parameters: see Fig. 3, K_C=768; 5G LDPC parameters: R=1/2, K_{5G}=WK_C=8448.

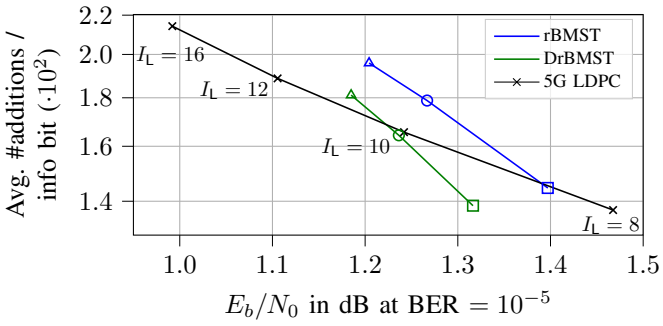


Fig. 7: Performance-complexity trade-offs of rBMST, DrBMST and 5G LDPC codes. BMST parameters: see Fig. 6, $S_W \in \{1, 2, 3\}$. 5G LDPC parameter: K_{5G}=WK_C=8448. Early termination is enabled.

degree of convergence in the window leads to an increased number of layer updates without influencing the BER in the considered range.

C. Computational Complexity Evaluation

In hardware implementations, addition operations are often more costly than the boxplus operations, which can be approximated using the min-sum rule. Thus, we evaluate the computational complexity as the average number of additions required during decoding. Fig. 6 illustrates the BER and average number of additions of rBMST codes with $m = 3$ and DrBMST codes employing the window step method with τ_S chosen as in Section IV-B. Measured at BER= 10^{-5}, the DrBMST structure achieves better error-rate performance and requires fewer additions compared to rBMST.

We consider the decoding latency as the number of received samples that must be collected before decoding an information block. For a BMST decoder, the latency equals WN_C bits, whereas for a 5G LDPC decoder it is N_{5G} bits. Accordingly, we define the equal-decoding-latency configuration of BMST and 5G LDPC codes as $N_{5G} = WN_C$. Fig. 7 depicts E_b/N_0 and the number of additions per information bit required to achieve a target BER $= 10^{-5}$. Here, $K_C = 768$ is used in BMST systems with $W = 11$ for a fair comparison, thus $K_{5G} = 8448$ holds for LDPC.

We observe that DrBMST codes achieve performance comparable to that of the rBMST codes while offering lower overall complexity. This advantage is due to the fact that DrBMST requires on average fewer layer updates, even though each layer update involves the same number of additions as the rBMST code with $m = 3$ (see Table I and Fig. 6).

Furthermore, DrBMST codes can provide a 0.13 dB coding gain over 5G LDPC codes at a similar computational complexity. The performance-complexity curves of DrBMST and 5G LDPC intersect at $I_L = 10$, beyond which the 5G LDPC code outperforms DrBMST for higher iteration counts.

V. Conclusions

This paper introduced a new way of achieving certain performance-complexity trade-off domains in BMST by allowing larger steps in the sliding-window decoder. Furthermore, the effectiveness of a novel early stopping criterion was shown, that exploits the constraints of the SPC nodes in the BMST graph. This mechanism can be universally applied to recursive BMST structures.

It was also shown that DrBMST codes employing the window step decoding method with SPC-based early stopping outperform rBMST and 5G LDPC codes under equal decoding latency constraints, while requiring similar number of additions.

References

[1] C. Berrou and A. Glavieux, "Near optimum error correcting coding and decoding: turbo-codes," *IEEE Transactions on Communications*, vol. 44, no. 10, pp. 1261–1271, Oct. 1996.

[2] R. Gallager, "Low-density parity-check codes," *IRE Transactions on Information Theory*, vol. 8, no. 1, pp. 21–28, Jan. 1962.

[3] A. Jimenez Felstrom and K. Zigangirov, "Time-varying periodic convolutional codes with low-density parity-check matrix," *IEEE Transactions on Information Theory*, vol. 45, no. 6, pp. 2181–2191, Sep. 1999.

[4] X. Ma, C. Liang, K. Huang, and Q. Zhuang, "Block Markov Superposition Transmission: Construction of Big Convolutional Codes From Short Codes," *IEEE Transactions on Information Theory*, vol. 61, no. 6, pp. 3150–3163, Jun. 2015.

[5] A. E. Pusane, A. J. Feltstrom, A. Sridharan, M. Lentmaier, K. S. Zigangirov, and D. J. Costello, "Implementation aspects of LDPC convolutional codes," *IEEE Transactions on Communications*, vol. 56, no. 7, pp. 1060–1069, Jul. 2008.

[6] S. Zhao, X. Ma, Q. Huang, and B. Bai, "Recursive block markov superposition transmission of short codes: Construction, analysis, and applications," *IEEE Transactions on Communications*, vol. 66, no. 7, pp. 2784–2796, 2018.

[7] ——, "Doubly-Recursive Block Markov Superposition Transmission: A Low-Complexity and Flexible Coding Scheme," *IEEE Transactions on Communications*, vol. 68, Aug. 2020.

[8] K. Huang and X. Ma, "Performance Analysis of Block Markov Superposition Transmission of Short Codes," *IEEE Journal on Selected Areas in Communications*, vol. 34, pp. 362–374, 2016.

[9] W. Ryan and S. Lin, *Channel codes: classical and modern*. Cambridge university press, 2009.

[10] G. Forney, "Codes on graphs: normal realizations," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 520–548, Feb. 2001.

[11] Q. Wang, S. Cai, W. Lin, S. Zhao, L. Chen, and X. Ma, "Spatially Coupled LDPC Codes via Partial Superposition and Their Application to HARQ," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 4, pp. 3493–3504, Apr. 2021.