Evaluasi Kompleksitas Pendekodean MAP pada Kode BCH Berdasarkan *Trellis* Terbagi

Emir Husni[1], Dimas Pamungkas[2]

***Abstract*— Soft decoding of block codes can be done by representing the block code into the trellis. One method of soft decoding commonly used is the maximum a posteriori probability (MAP). However, the implementation of this method requires a high computational complexity. Reducing the complexity can be done by changing the trellis shape of the block code. This paper shows the process of the block code's trellis formation and the evaluation of computational complexity and bit error ratio for every trellis shape of block codes. The evaluation of codes using the MAP method is compared to the evaluation of the soft output Viterbi algorithm (SOVA) method. The result shows that soft decoding using MAP method is better than soft coding using SOVA method and hard decoding method.**

***Intisari*—Pendekodean secara halus pada kode blok dapat dilakukan dengan cara merepresentasikan kode blok tersebut ke dalam bentuk *trellis*. Salah satu metode pendekodean secara halus yang umum digunakan adalah maksimum probabilitas posteriori (*maximum a posteriori probability*, MAP). Namun implementasi metode ini membutuhkan kompleksitas komputasi yang besar. Pengurangan kompleksitas dapat dilakukan dengan mengubah bentuk *trellis* dari kode blok. Makalah ini menunjukkan proses pembentukan *trellis* dari kode blok dan hasil evaluasi kompleksitas komputasi serta rasio kesalahan bit berdasarkan bentuk *trellis* kode blok tersebut. Evaluasi pengkodean dengan metode MAP dibandingkan dengan metode algoritme Viterbi keluaran halus (*soft output Viterbi algorithm*, SOVA). Hasil simulasi menunjukkan bahwa pendekodean secara halus dengan metode MAP lebih unggul dibandingkan pengkodean secara halus dengan metode SOVA dan pendekodean secara keras.**

***Kata Kunci*—pendekodean secara halus, pendekodean MAP, pembagian *trellis*, kode blok.**

I. PENDAHULUAN

Pendekodean secara halus dapat menghasilkan unjuk kerja rasio kesalahan bit (*bit error rate*, BER) yang lebih baik dibandingkan dengan pendekodean secara keras. Pendekodean secara keras hanya dapat menetapkan dan mengolah bit dalam bentuk 0 atau 1, sedangkan pendekodean secara halus memungkinkan pengolahan bit dengan ukuran ketepatan dalam bilangan real, yang hasilnya disebut keluaran halus.

Dalam awal perkembangannya, kode blok hanya dapat didekode secara keras. Namun, sebuah penelitian menunjukkan bahwa pendekodean secara halus pada kode blok dapat dilakukan [1]. Pencarian algoritme pendekodean secara halus yang mudah diimplementasikan dan efisien pada kode blok masih menjadi hal yang menantang. Cara terbaik untuk mendapatkan pendekodean yang efisien adalah dengan mengubah bentuk *trellis* dari kode tersebut, seperti yang telah diterangkan pada beberapa penelitian [2]-[6].

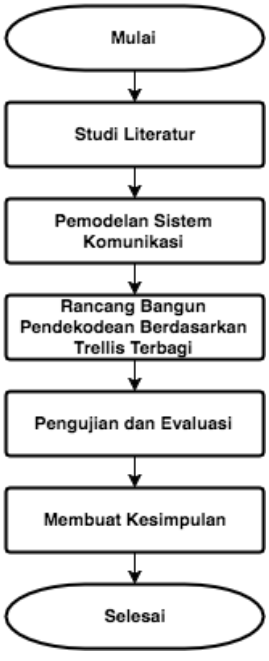
[1,2] *Sekolah Teknik Elektro & Informatika, Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132 Indonesia (e-mail: ehusni@lskk.ee.itb.ac.id)*

Masalah pada pengkodean secara halus adalah implementasi metode ini membutuhkan komputasi yang besar. Oleh karena itu, untuk dapat digunakan secara praktis, kompleksitas komputasi pendekodean harus dikurangi.

Makalah ini meneliti kompleksitas komputasi yang dibutuhkan kode Bose, Chaudhuri, dan Hocquenghem (BCH). Perubahan pada bentuk *trellis* dapat menghasilkan kompleksitas pendekodean yang lebih efisien.

Salah satu cara untuk mendapatkan pendekodean yang efisien adalah dengan mengubah bentuk *trellis* menjadi lebih sederhana. Metode pembagian *trellis* memungkinkan untuk menghasilkan keluaran dengan lebih banyak kode bit dalam satu bagian *trellis*. Hal ini dapat mengurangi banyak komputasi yang dibutuhkan dekoder untuk dapat mengolah sinyal yang diterima. Algoritme pembagian *trellis* optimal menghasilkan *trellis* terbagi dengan banyak komputasi minimum yang dibutuhkan oleh dekoder.

Efisiensi kompleksitas komputasi dan hubungannya dengan BER dari metode pendekodean secara halus, dalam kasus ini maksimum probabilitas posteriori (*maximum a posteriori probability*, MAP), dianalisis dan dibandingkan dengan efisiensi kompleksitas komputasi dan hubungannya dengan BER dari metode algoritme Viterbi keluaran halus (*soft output Viterbi algorithm*, SOVA). Metode pendekodean MAP memiliki kelebihan dalam unjuk kerja kesalahan dibandingkan dengan metode pendekodean SOVA. Sebaliknya, metode pendekodean SOVA memiliki kelebihan dalam efisiensi kompleksitas komputasi dibandingkan dengan metode pendekodean MAP.



Gbr. 1 Alur penelitian.

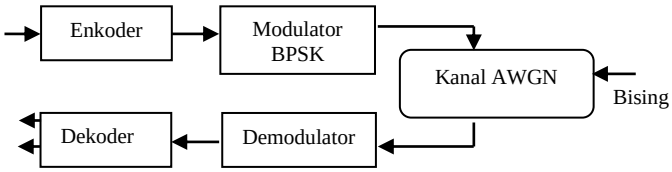
Alur penelitian dalam makalah ini dapat dilihat pada Gbr. 1. Penelitian dimulai dengan studi literatur yang dilanjutkan dengan tahapan pemodelan sistem komunikasi dengan kanal AWGN yang akan digunakan dalam simulasi untuk proses pengujian. Tahapan berikutnya adalah rancang bangun pendekodean MAP dan SOVA berdasarkan *trellis* terbagi. Rancang bangun meliputi konstruksi *trellis* level-bit dan *trellis* terbagi optimal yang digunakan untuk pendekodean MAP dan SOVA. Tahapan akhir adalah pengujian dan evaluasi, yaitu menganalisis hasil simulasi yang telah dilakukan dan membuat kesimpulan.

Hasil pembagian *trellis* berdasarkan kompleksitas masing-masing metode menunjukkan bahwa penurunan kompleksitas komputasi dapat terjadi.

II. PEMODELAN SISTEM

A. Model Sistem Komunikasi

Model sistem komunikasi diilustrasikan pada Gbr. 2. Enkoder bertugas untuk membentuk kode bit dengan panjang N dari informasi dengan panjang K. Modulator *binary phase-shift keying* (BPSK) memetakan urutan kode yang diterima dari enkoder menjadi urutan bipolar, dengan bit 0 diubah menjadi -1 dan bit 1 tetap bernilai 1. Model kanal yang digunakan adalah AWGN. Demodulator mengirimkan urutan sinyal menuju dekoder untuk diproses. *Trellis* digunakan untuk mendekode kode linear dengan menerapkan metode pendekodean MAP.



Gbr. 2 Model sistem komunikasi.

B. Kode BCH

Kode biner BCH ditemukan oleh Hocquenghem pada tahun 1959 dan secara independen oleh Bose dan Chaudhuri pada tahun 1960. Untuk setiap bilangan bulat $m \geq 3$ dan $t < 2^{m-1}$, terdapat kode biner BCH dengan parameter sebagai berikut.

Panjang blok: $n = 2^m - 1$,
Banyak digit cek paritas: $n - k \leq mt$,
Jarak minimum: $d_{min} \geq 2t + 1$.

Kode BCH yang digunakan dalam simulasi ini adalah kode dengan panjang kode N = 7 dan panjang informasi K = 4 yang memiliki matriks pembangkir sebagai berikut.

$$G = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}$$

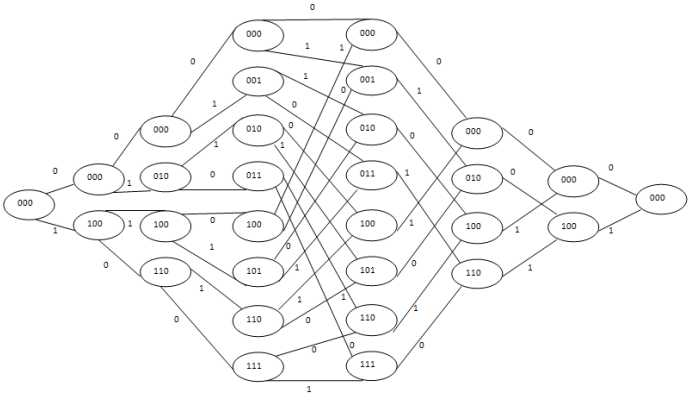
C. Trellis Level-Bit dan Terbagi

Trellis merupakan graf yang merepresentasikan enkoder melalui bentuk diagram keadaan yang diperluas sesuai waktu. Ruang keadaan dari enkoder untuk setiap waktu i ditentukan melalui himpunan bit informasi yang dinotasikan sebagai A_i^s, yang berkorespondensi terhadap baris-baris matriks pembangkit yang dinotasikan sebagai G_i^s.

Konstruksi N-bagian *trellis* T dibangun secara serial bagian per bagian. Misal *trellis* telah dikonstruksi sampai bagian-i, maka selanjutnya akan dikonstruksi bagian ke-$(i+1)$ dari waktu- i sampai waktu-$(i+1)$. Bagian ke-$(i+1)$ dapat dikonstruksi dengan mengikuti langkah berikut.

1. Menentukan G_{i+1}^s dan A_{i+1}^s, dan bentuk ruang keadaan S_{i+1} pada waktu-$(i+1)$.
2. Untuk setiap keadaan $\sigma_i \in S_i$, ditentukan transisi keadaan dan σ_i dihubungkan ke keadaan yang bertetangga pada S_{i+1} oleh suatu garis.
3. Untuk setiap transisi keadaan, ditentukan bit keluaran dan pada garis yang bersesuaian diberi label.

Dengan mengikuti langkah-langkah tadi, maka diperoleh bentuk *trellis*-level bit kode BCH (7,4) seperti pada Gbr. 3.



Gbr. 3 Bentuk *trellis*- level bit kode BCH (7,4).

Trellis terbagi dapat dibentuk dari *trellis*-level bit, yaitu dengan memilih kolom keadaan S_i dan menghubungkan keadaan-keadaannya bersesuaian dengan keterhubungan keadaan tersebut pada *trellis* level-bit.

D. Pendekodean MAP

Algoritme MAP dikembangkan oleh Bahl *et al* pada tahun 1974 [7]. Dalam algoritme ini ada tiga langkah besar yang harus dilakukan, yaitu menghitung peluang cabang, peluang keadaan, dan keluaran halus. Langkah pertama yaitu menghitung semua peluang cabang $b_i(\sigma, \sigma')$ yang menghubungkan keadaan σ menuju keadaan σ' untuk setiap bagian *trellis*, dihitung seperti dalam (1).

$$P\big(b_i(\sigma, \sigma')\big) \propto \exp\{\tfrac{2r_i u_i}{N_0}\} \quad (1)$$

Peluang keadaan σ dihitung melalui dua cara yaitu rekursi maju, seperti pada (2), dan mundur, seperti pada (3), dengan $\Omega_{i-1}(\sigma')$ menotasikan himpunan keadaan yang menghubungkan keadaan dengan keadaan sesudahnya, sedangkan $\psi_i(\sigma)$ menotasikan himpunan keadaan yang menghubungkan suatu keadaan dengan keadaan sebelumnya.

$$\alpha_i(\sigma') = \sum_{\sigma \in \Omega_{i-1}(\sigma')} P\Big(b_i(\sigma, \sigma')\Big)\alpha_{i-1}(\sigma) \quad (2)$$

$$\beta_{i-1}(\sigma) = \sum_{\sigma' \in \psi_i(\sigma)} P\Big(b_i(\sigma, \sigma')\Big)\beta_i(\sigma') \quad (3)$$

Untuk menghasilkan keluaran halus, algoritme ini meninjau semua jalur setiap bagian *trellis* dan membaginya menjadi dua set yang bersesuaian dengan keluaran enkoder. Keluaran halus setiap bit dari MAP didefinisikan sebagai rasio dari peluang dua set tersebut dalam domain logaritmik,

$$\mathrm{L}(\hat{\mathrm{c}}_{\mathrm{i}}) = \log \frac{\sum_{\substack{(\sigma,\sigma') \\ c_i=1}} \alpha_{i-1}(\sigma) P\left(b_i(\sigma,\sigma')\right) \beta_i(\sigma')}{\sum_{\substack{(\sigma,\sigma') \\ c_i=0}} \alpha_{i-1}(\sigma) P\left(b_i(\sigma,\sigma')\right) \beta_i(\sigma')} \quad (4)$$

E. Pengkodean SOVA

Algoritme SOVA merupakan modifikasi dari algoritme Viterbi yang menghasilkan aproksimasi keluaran halus untuk setiap bit kode. Pengkodean SOVA sudah diaplikasikan pada [8]. Semua peluang cabang pada setiap bagian *trellis* dihitung sama seperti pada algoritme Viterbi, seperti dalam (5).

$$\bar{P}(b_i(\sigma,\sigma')) \propto \frac{2}{N_0} r_i u_i \quad (5)$$

dengan $\bar{P}(b_i(\sigma,\sigma')$ merepresentasikan peluang transisi enkoder dari keadaan $\sigma \in S_{i-1}$ menuju keadaan $\sigma' \in S_i$ dengan melewati cabang $b_i(\sigma,\sigma')$ yang bersesuaian dengan kode yang diterima, dinotasikan sebagai r_i, pada setiap bagian *trellis*. Selanjutnya, peluang keadaan untuk rekursi maju $\bar{\alpha}_i(\sigma')$ dalam (6) dan rekursi mundur $\bar{\beta}_{i-1}(\sigma)$ dalam (7).

$$\bar{\alpha}_i(\sigma') = \max_{\sigma \in \Omega_{i-1}(\sigma')} \{\bar{P}\left(b_i(\sigma,\sigma')\right) + \bar{\alpha}_{i-1}(\sigma) \quad (6)$$

$$\bar{\beta}_{i-1}(\sigma) = \max_{\sigma' \in \Psi_i(\sigma)} \{\bar{P}\left(b_i(\sigma,\sigma')\right) + \bar{\beta}_i(\sigma') \quad (7)$$

Untuk mendapatkan keluaran halus, SOVA mempertimbangkan dua jalur pada setiap bagian *trellis*, yaitu jalur kemungkinan maksimum yang bersesuaian dengan bagian *trellis* dan jalur dengan peluang terbesar yang merupakan komplemen dari jalur kemungkinan maksimum (KM). SOVA hanya menjamin satu jalur terbaik yaitu jalur KM. Nilai keluaran halus dihampiri sesuai perbedaan antar cabang pada jalur, seperti dalam (8).

$$L(\hat{c}_i) \approx \bar{P}(c_i = 0|r) - \bar{P}(c_i = 1|r) \quad (8)$$

Jalur peluang terbesar dengan label x merupakan komplemen terhadap estimasi KM diperoleh dari (9).

$$\bar{P}(c_i = x|r) = \max_{\substack{(\sigma,\sigma') \\ c_i=x}} \{\bar{\alpha}_{i-1}(\sigma) + \bar{P}\left(b_i(\sigma,\sigma')\right) + \bar{\beta}_i(\sigma')\} \quad (9)$$

III. KOMPLEKSITAS KOMPUTASI

Kompleksitas komputasi merupakan banyak operasi yang dibutuhkan untuk proses pendekodean. Penghitungan kompleksitas komputasi untuk metode MAP [9] dan metode SOVA [3] diimplementasikan untuk beberapa kode BCH dengan bentuk *trellis* level-bit dan terbagi optimal.

A. Kompleksitas Komputasi untuk Pendekodean Algoritme MAP

1) Peluang Cabang: Peluang cabang, $P\left(b_{h_j}(\sigma,\sigma')\right)$, merepresentasikan peluang transisi enkoder dari keadaan $\sigma \in S_{h_j-1}$ menuju keadaan $\sigma' \in S_{h_j}$ melalui cabang $b_{h_j}(\sigma,\sigma')$ yang berhubungan dengan r_j dalam T_j seperti dalam (10).

$$P\left(b_{h_j}(\sigma,\sigma')\right) = P\left(\sigma', b_{h_j}(\sigma,\sigma'), r_j \middle| \sigma\right) = P\left(\sigma', b_{h_j}(\sigma,\sigma'), \middle|\sigma\right) P(r_j|((\sigma,\sigma'), b_{h_j}(\sigma,\sigma')) \quad (10)$$

$P(\sigma', b_i(\sigma,\sigma')|\sigma)$ untuk saluran AWGN dengan rata-rata nol dan varians $N_0/2$ dapat disederhanakan menjadi (11).

$$P\left(b_{h_j}(\sigma,\sigma')\right) \approx e \exp\left\{\frac{2}{N_0} \sum_{m=h_{j-1}+1}^{h_j} r_m u_m\right\} \quad (11),$$

dengan $u_m = 2c_m - 1$ untuk $c_{h_j} = 0$ atau 1. Banyak operasi yang dibutuhkan untuk menghitung peluang cabang dari waktu h_{j-1} sampai h_j, $l_j = h_j - h_{j-1}$, adalah sebagai berikut:

Penjumlahan : $\left|B_j^d\right| . \left|B_j^p\right| . (l_j - 1)$, untuk $1 \le j \le v$
Perkalian : l_j.

2) Peluang Cabang Komposit: Peluang cabang komposit dihitung apabila keadaan $\sigma \in S_{h_j-1}$ terhubung oleh lebih satu cabang menuju keadaan $\sigma' \in S_{h_j}$. Pada algoritme MAP, peluang cabang komposit, $P\left(L_{h_j}(\sigma,\sigma')\right)$, didefinisikan dalam (12).

$$P\left(L_{h_j}(\sigma,\sigma')\right) = P\left(L_t^+(\sigma,\sigma')\right) + P\left(L_t^-(\sigma,\sigma')\right) \quad (12)$$

untuk suatu nilai t dalam selang $h_{j+1} + 1 \le t \le h_j$. Peluang bit komposit $P\left(L_t^+(\sigma,\sigma')\right)$ merepresentasikan peluang setiap keluaran enkoder untuk setiap bit u_t pada cabang komposit $L_{h_j}(\sigma,\sigma')$ dan didefinisikan dalam (13).

$$P\left(L_t^{\pm}(\sigma,\sigma')\right) = \sum_{\substack{b(\sigma,\sigma') \in L(\sigma,\sigma') \\ u_t = \pm 1}} P(b_{h_j}(\sigma,\sigma') \quad (13)$$

untuk $h_{j+1} + 1 \le t \le h_j$.

Perhitungan persamaan peluang cabang komposit membutuhkan satu operasi penjumlahan untuk setiap cabang komposit berbeda.

Penjumlahan: $\left|B_j^d\right|$

3) Rekursi Maju: Peluang enkoder mencapai keadaan $\sigma' \in S_{h_j}$ dari keadaan $\sigma \in S_{h_j-1}$, pada T_j melalui rekursi maju dari waktu $h_j = 0$ sampai dengan $h_j = N$ untuk algoritme MAP adalah (14).

$$\alpha_{h_j}(\sigma') = \sum_{\sigma \in \Omega_{h_{j-1}}(\sigma')} P\left(L_{h_j}(\sigma,\sigma')\right) \alpha_{h_{j-1}}(\sigma) \quad (14)$$

dengan $\alpha_0(\sigma_0) = 1$. Jika hanya ada satu cabang yang menghubungkan keadaan $\sigma \in S_{h_j-1}$ menuju keadaan $\sigma' \in S_{h_j}$, maka $P\left(L_{h_j}(\sigma,\sigma')\right)$ dapat diganti dengan $P\left(b_{h_j}(\sigma,\sigma')\right)$.

Operasi perkalian diperlukan untuk setiap cabang komposit yang konvergen (masuk) ke keadaan $\sigma' \in S_{h_j}$, kemudian hasil dari perkalian tiap cabang tersebut dijumlahkan. Pada bagian pertama, T_1, tidak perlu ada operasi perkalian karena nilai $\alpha_0(\sigma_0) = 1$. Oleh karena itu, banyak operasi yang dibutuhkan untuk rekursi maju pada setiap T_j adalah:

Penjumlahan : $\left|B_j^c\right| - \left|S_j^c\right|$, untuk $1 \le j \le v$

Perkalian : $\begin{cases} \left|B_j^c\right|, & untuk\ 1 < j \le v \\ 0, & untuk\ j = 1 \end{cases}$.

4) Rekursi Mundur: Peluang enkoder mencapai keadaan $\sigma \in S_{h_j-1}$ dari keadaan $\sigma' \in S_{h_j}$, pada T_j melalui rekursi mundur dari waktu $h_j = N$ sampai dengan $h_j = 0$ untuk algoritma MAP adalah (15).

$$\beta_{h_{j-1}}(\sigma) = \sum_{\sigma' \in \psi_{h_j}(\sigma)} P\left(L_{h_j}(\sigma, \sigma')\right) \beta_{h_j}(\sigma') \quad (15)$$

dengan $\beta_v(\sigma_f) = 1$. Jika hanya ada satu cabang yang menghubungkan keadaan $\sigma' \in S_{h_j}$ menuju keadaan $\sigma \in S_{h_j-1}$, maka $P\left(L_{h_j}(\sigma, \sigma')\right)$ dapat diganti dengan $P\left(b_{h_j}(\sigma, \sigma')\right)$.

Rekursi mundur dapat dilihat sebagai pencerminan dari rekursi maju. Operasi perkalian diperlukan untuk setiap cabang komposit yang divergen (keluar) dari keadaan $\sigma \in S_{h_{j-1}}$ dan hasil dari perkalian tiap cabang kemudian dijumlahkan. Pada bagian terakhir, T_v, tidak perlu ada operasi perkalian karena nilai $\beta_v(\sigma_f) = 1$. Oleh karena itu, banyak operasi yang dibutuhkan untuk rekursi mundur pada setiap T_j adalah sebagi berikut:

Penjumlahan : $\left|B_j^c\right| - \left|S_{h_{j-1}}\right|$, untuk $1 \le j \le v$

Perkalian : $\begin{cases} \left|B_j^c\right|, & untuk\ 1 \le j < v \\ 0, & untuk\ T_v \end{cases}$

5) Peluang Bit Komposit: Seperti yang sudah dijelaskan sebelumnya, apabila terdapat cabang komposit yang terdiri atas lebih dari satu cabang, maka perlu dihitung peluang setiap kemungkinan keluaran enkoder untuk setiap bit u_t dengan $h_{j-1} + 1 \le t \le h_j$, pada T_j. Untuk algoritme MAP, $P\left(L_t^{\pm}(\sigma, \sigma')\right)$ didefinisikan dalam (16).

$$P(L_t^{\pm}(\sigma, \sigma')) = \sum_{\substack{b(\sigma,\sigma') \in L(\sigma,\sigma') \\ u_t = \pm 1}} P(b_{h_j}(\sigma, \sigma')) \quad (16)$$

untuk $h_{j-1} + 1 \le t \le h_j$.

Untuk setiap kemungkinan keluaran enkoder di antara setiap cabang komposit yang berbeda, diperlukan penjumlahan peluang semua cabang yang berkaitan dengan keluaran enkoder untuk semua l_j bit pada setiap bagian. Oleh karena itu, banyak operasi penjumlahan yang dibutuhkan untuk menghasilkan $P(L_t^{\pm}(\sigma, \sigma'))$ pada setiap T_j adalah:

Penjumlahan: $2 \cdot \left|B_j^d\right| \cdot \left(\frac{\left|B_j^p\right|}{2} - 1\right) \cdot l_j$, untuk $1 \le j \le v$

6) Keluaran Halus: Algoritme MAP memberikan nilai keluaran halus, $\mathrm{L}(\mathrm{c_t})$, untuk setiap estimasi bit kode, $\mathrm{c_t}$, dengan $h_{j-1} + 1 \le t \le h_j$ pada T_j dalam bentuk *Log Likelihood Ratio* seperti dalam (17).

$$\mathrm{L}(\mathrm{c_t}) = \log \frac{\sum_{\substack{(\sigma,\sigma') \\ u_t = +1}} \alpha_{h_{j-1}}(\sigma) P\left(L_t^{+}(\sigma,\sigma')\right) \beta_{h_j}(\sigma')}{\sum_{\substack{(\sigma,\sigma') \\ u_t = -1}} \alpha_{h_{j-1}}(\sigma) P\left(L_t^{-}(\sigma,\sigma')\right) \beta_{h_j}(\sigma')} \quad (17)$$

Perlu diperhatikan nilai keluaran halus yang dihasilkan dari algoritme MAP berbeda untuk bagian *trellis* yang memiliki cabang paralel dan tidak. Kasus pertama, jika ukuran cabang komposit pada T_j sama dengan satu, maka menghitung LLR dari masing-masing penyebut dan pembilang dari (17) membutuhkan dua perkalian untuk setiap cabang komposit pada setiap bagian, kecuali bagian pertama dan terakhir yang hanya membutuhkan satu perkalian, karena $\alpha_0(\sigma_0) = \beta_v(\sigma_f) = 1$. Operasi perkalian dilakukan terhadap 1 bit dalam setiap bagian dan hasil perkaliannya ditambahkan dengan hasil dari cabang komposit lain pada posisi bit yang sama. Operasi terakhir pada LLR adalah pembagian. Untuk setiap bagian dengan panjang l_j diperlukan l_j operasi pembagian. Dari evaluasi tersebut dapat diketahui banyak operasi yang dibutuhkan untuk menghasilkan keluaran halus dari setiap bit pada T_j, yaitu hanya terdapat satu cabang yang menghubungkan keadaan $\sigma \in S_{h_j-1}$ dengan keadaan $\sigma' \in S_{h_j}$ adalah:

Penjumlahan: $2 \cdot \left(\frac{\left|B_j^c\right|}{2} - 1\right) \cdot l_j$, untuk $1 \le j \le v$

Perkalian: $\begin{cases} 2 \cdot \left|B_j^c\right| + l_j, & untuk\ 1 < j < v \\ \left|B_j^c\right| + l_j, & untuk\ j = 1, v \end{cases}$.

Kasus kedua, jika ukuran cabang komposit lebih dari satu, cara yang efisien untuk mengevaluasi persamaan dari $\mathrm{L}(\mathrm{c_t})$ adalah dengan menghitung (18).

$$S = \sum_{(\sigma,\sigma')} \alpha_{h_{j-1}}(\sigma) P\left(L_{h_j}(\sigma, \sigma')\right) \beta_{h_j}(\sigma') \quad (18)$$

Persamaan (18) membutuhkan dua perkalian untuk setiap cabang komposit di setiap T_j, kecuali T_1 dan T_v yang hanya membutuhkan satu perkalian, karena $\alpha_0(\sigma_0) = \beta_v(\sigma_f) = 1$. Hasil setelah melewati operasi tadi kemudian dijumlahkan, maka banyak operasi yang dibutuhkan untuk setiap T_j adalah:

Penjumlahan : $\left|B_j^c\right| - 1$, untuk $1 \le j \le v$

Perkalian : $\begin{cases} 2 \cdot \left|B_j^c\right| + l_j, & untuk\ 1 < j < v \\ \left|B_j^c\right| + l_j, & untuk\ j = 1, v \end{cases}$.

Selanjutnya, penyebut pada (17) dievaluasi untuk setiap bit dalam satu bagian. Hanya dibutuhkan satu perkalian untuk setiap cabang komposit dan kemudian hasilnya dijumlahkan. Banyak operasi penjumlahan dan perkalian yang dibutuhkan adalah:

Penjumlahan : $\left(\left|B_j^c\right| - 1\right) \cdot l_j$, untuk $1 \le j \le v$

Perkalian : $\left|B_j^c\right| \cdot l_j$, untuk $1 \le j \le v$.

Pembilang (17) didapatkan melalui selisih dari hasil yang diperoleh dari evaluasi penyebut dan hasil (18) untuk setiap bit pada T_j. Operasi pengurangan yang dibutuhkan sebanyak l_j. Langkah terakhir adalah melakukan operasi pembagian sebanyak l_j untuk mendapatkan nilai LLR dari setiap bit pada T_j.

B. Kompleksitas Komputasi untuk Pendekodean SOVA

Pada bagian ini kompleksitas komputas diperiksa untuk pendekodean satu bagian T_j, dari waktu h_{j-1} menuju waktu h_j,

dengan panjang $l_j = h_j - h_{j-1}$, pada *trellis* yang terbagi menjadi v bagian, dengan batas bagian $\{h_0, h_1, \dots, h_v\}$. Total kompleksitas komputasi tiap bagian T_j menentukan kompleksitas komputasi untuk SOVA pada *trellis* terbagi. Kompleksitas komputasi untuk SOVA pada *trellis* terbagi terdiri atas pencarian peluang setiap cabang dan setiap cabang komposit, peluang keadaan rekursi maju dan mundur, dan aproksimasi keluaran halus untuk setiap bit kode. Peluang tersebut dihitung dalam domain logaritmik dan dinotasikan sebagai $\bar{P}$, $\bar{\alpha}$ dan $\bar{\beta}$.

1) Peluang Cabang: Peluang cabang, $\bar{P}(b_{h_j}(\sigma,\sigma'))$, merepresentasikan peluang enkoder mencapai keadaan $\sigma' \in S_{h_j}$ dari keadaan $\sigma \in S_{h_{j-1}}$ melewati cabang pada bagian T_j didefinisikan dalam (19).

$$\bar{P}(b_{h_j}(\sigma,\sigma')) \propto \frac{2}{N_0}\sum_{m=h_{j-1}+1}^{h_j} r_m u_m. \quad (19)$$

Dari (19), terlihat SOVA cukup memerhatikan bagian $\sum_{m=h_{j-1}+1}^{h_j} r_m u_m$, karena $\frac{2}{N_0}$ selalu bernilai konstan. Perlu dipertimbangkan bahwa tidak ada operasi yang dibutuhkan untuk menghitung $r_m u_m$, karena nilai $u_m = \{-1, +1\}$. Perhitungan $\bar{P}(b_{h_j}(\sigma,\sigma'))$ membutuhkan $l_j - 1$ penjumlahan untuk setiap cabang paralel, $|B_j^p|$, dengan setiap cabang komposit, $|B_j^d|$, berbeda. Dari evaluasi tersebut, maka banyak operasi penjumlahan untuk setiap T_j adalah:

$$\text{Penjumlahan:} |B_j^d| \cdot |B_j^p| \cdot (l_j - 1), untuk\ 1 \le j \le v.$$

2) Peluang Cabang Komposit: Peluang cabang komposit, $\bar{P}(L_{h_j}(\sigma,\sigma'))$, perlu dihitung apabila ada lebih dari satu cabang yang menghubungkan keadaan $\sigma \in S_{h_{j-1}}$ menuju keadaan $\sigma' \in S_{h_j}$. Persamaan peluang cabang komposit adalah seperti dalam (20).

$$\bar{P}\left(L_{h_j}(\sigma,\sigma')\right) = \max_{b(\sigma,\sigma') \in L(\sigma,\sigma')}(\bar{P}(b_{h_j}(\sigma,\sigma')). \quad (20)$$

Perhitungan (20) membutuhkan $|B_j^d| \cdot (|B_j^p| - 1)$ pembandingan. Karena tujuan SOVA adalah untuk memaksimalkan peluang untuk seluruh jalur maka banyak operasi pembandingan dapat dikurangi menjadi $|B_j^d| \cdot (|B_j^p|/2 - 1)$, jika cabang-cabang paralel pada setiap cabang komposit di T_j saling komplemen satu sama lain.

3) Rekursi Maju: Peluang keadaan untuk rekursi maju dari waktu $h_j = 0$ sampai $h_j = N$ didefinisikan dalam (21) dengan $\bar{\alpha}_0(\sigma_0) = 0$.

$$\bar{\alpha}_{h_j}(\sigma') = \max_{\sigma \in \Omega_{h_{j-1}}(\sigma')}\left\{\bar{P}\left(L_{h_j}(\sigma,\sigma')\right) + \bar{\alpha}_{h_{j-1}}(\sigma)\right\}. \quad (21)$$

Apabila ukuran cabang komposit tidak lebih dari satu, maka $\bar{P}\left(L_{h_j}(\sigma,\sigma')\right)$ diganti menjadi $\bar{P}\left(b_{h_j}(\sigma,\sigma')\right)$. Untuk menghitung persamaan tersebut, untuk setiap bagian T_j, dibutuhkan operasi sebanyak:

$$\text{Pembandingan} : |B_j^c| - |S_{h_j}|\ untuk\ 1 \le j \le v$$
$$\text{Penjumlahan} : \begin{cases} |B_j^c| \text{ untuk } 1 < j \le v \\ 0 \text{ untuk } T_1 \end{cases}.$$

4) Rekursi Mundur: Peluang keadaan untuk rekursi mundur dari waktu $h_j = N$ sampai $h_j = 0$ didefinisikan dalam (22) dengan $\bar{\beta}_v(\sigma_f) = 0$.

$$\bar{\beta}_{h_{j-1}}(\sigma) = \max_{\sigma' \in \Psi_{h_j}(\sigma)}\left\{\bar{P}\left(L_{h_j}(\sigma,\sigma')\right) + \bar{\beta}_{h_j}(\sigma')\right\}. \quad (22)$$

Sama seperti pada rekursi maju, apabila ukuran cabang komposit tidak lebih dari satu, maka $\bar{P}\left(L_{h_j}(\sigma,\sigma')\right)$ diganti menjadi $\bar{P}\left(b_{h_j}(\sigma,\sigma')\right)$. Untuk menghitung persamaan tersebut, untuk setiap bagian T_j dibutuhkan operasi sebanyak:

$$\text{Pembandingan} : |B_j^c| - |S_{h_{j-1}}|\ untuk\ 1 \le j \le v$$
$$\text{Penjumlahan} : \begin{cases} |B_j^c|\ untuk\ 1 \le j < v \\ 0\ untuk\ T_v \end{cases}.$$

5) Peluang Bit Komposit: Jika ukuran cabang komposit melebih satu, maka penghitungan untuk peluang bit komposit perlu dilakukan. Peluang bit komposit, $\bar{P}(L_t^{\pm}(\sigma,\sigma')$, merepresentasikan peluang setiap kemungkinan keluaran enkoder untuk setiap bit u_t pada cabang komposit $L_{h_j}(\sigma,\sigma'))$, didefinisikan dalam (23) untuk $h_{j-1} + 1 \le t \le h_j$.

$$\bar{P}(L_t^{\pm}(\sigma,\sigma') = \max_{\substack{b(\sigma,\sigma') \in L(\sigma,\sigma') \\ u_t = \pm 1}}\{\bar{P}(b_{h_j}(\sigma,\sigma'))\}. \quad (23)$$

Bergantung pada nilai dari setiap bit pada cabang komposit, peluang cabang komposit $\bar{P}(L_{h_j}(\sigma,\sigma'))$ disusun berdasarkan $\bar{P}(L_t^{\pm}(\sigma,\sigma')$ yang sesuai. Tidak diperlukan komputasi untuk melakukan hal tersebut karena hanya dibutuhkan operasi Boolean. Banyak operasi yang dibutuhkan untuk menghitung $\bar{P}(L_t^{\pm}(\sigma,\sigma')$ yaitu:

$$\text{Pembandingan:} \left(\frac{|B_j^p|}{2} - 1\right) \cdot |B_j^d| \cdot l_j\ untuk\ 1 \le j \le v.$$

6) Keluaran Halus: Pada SOVA, setiap pendekodean bit $\bar{c}_t$ untuk $h_{j-1} + 1 \le t \le h_j$ pada T_j diaproksimasi dengan (24).

$$L(\hat{c}_t) \approx \bar{P}(c_t = 0|r) - \bar{P}(c_t = 1|r). \quad (24)$$

Untuk setiap bit pada T_j, peluang dari jalur KM, $\bar{\alpha}_v(\sigma_f)$, disusun ke dalam bagian dari (24) sesuai dengan estimasi KM untuk bit $\bar{c}_t$. Peluang dari jalur kemungkinan terbesar dengan label, x, yang merupakan komplemen dari estimasi KM untuk bit $\bar{c}_t$ didefinisikan dalam (25).

$$\bar{P}(c_t = x|\boldsymbol{r}) = \max_{\substack{(\sigma,\sigma') \\ c_t = x}}\{\bar{\alpha}_{h_{j-1}}(\sigma) + \bar{P}\left(L_t^{sign(x)}(\sigma,\sigma')\right) + \bar{\beta}_{h_j}(\sigma')\}. \quad (25)$$

Apabila hanya ada satu cabang yang menghubungkan keadaan $\sigma \in S_{h_{j-1}}$ ke keadaan $\sigma' \in S_{h_j}$, $\bar{P}(L_t^{\pm}(\sigma,\sigma')$ digantikan dengan peluang cabang $\bar{P}(b_{h_j}(\sigma,\sigma'))$ yang

mempunyai label bernilai x untuk bit c_t. Dari evaluasi terhadap (25), dibutuhkan dua penjumlahan untuk semua cabang sesuai dengan komplemen estimasi KM, kecuali pada bagian T_1 dan T_v. Untuk bit pertama, dibutuhkan $2.|B_j^c|/2$ penjumlahan. Namun, untuk subsekuen bit lain pada bagian itu tidak perlu dilakukan pengulangan perhitungan, mengingat semua perhitungan telah disimpan, kecuali untuk cabang dengan bit yang merupakan komplemen terhadap estimasi KM.

Perlu diperhatikan bahwa untuk seluruh bagian, semua cabang perlu diperhitungkan, kecuali cabang yang merepresentasikan estimasi KM, yaitu sebanyak $|B_j^c| - |B_j^c|/|B_j^d|$ cabang. Bagian T_1 dan T_v membutuhan satu penjumlahan untuk banyak cabang yang sama. Setelah dilakukan penjumlahan, langkah selanjutnya adalah pembandingan untuk mendapatkan nilai yang maksimum. Perhitungan LLR pada (24) untuk bagian dengan panjang l_j, membutuhkan l_j operasi pengurangan. Secara keseluruhan, untuk mendapatkan keluaran halus setiap bit pada T_j dengan hanya satu cabang yang menghubungkan keadaan $\sigma \in S_{h_{j-1}}$ ke keadaan $\sigma' \in S_{h_j}$ dibutuhkan operasi sebanyak:

Pembandingan: $\left(\frac{|B_j^c|}{2} - 1\right) \cdot l_j \; untuk \; 1 \le j \le v$

Penjumlahan: $\begin{cases} 1 \cdot \left(|B_j^c| - \frac{|B_j^c|}{|B_j^d|}\right) + l_j, untuk \; j = 1, v \\ 2 \cdot \left(|B_j^c| - \frac{|B_j^c|}{|B_j^d|}\right) + l_j \; , untuk \; 1 < j < v \end{cases}$.

Untuk kasus lain dengan ukuran cabang komposit melebihi satu, evaluasi dari LLR berdasarkan (25) pada semua bagian, kecuali bagian pertama dan terakhir, membutuhkan dua penjumlahan untuk bit pertama dari setiap cabang komposit. Bit lain yang tersisa pada T_j, yaitu sebanyak $l_j - 1$ bit, cukup membutuhkan satu penjumlahan. Maka untuk menghitung keluaran halus dari kode bit dengan panjang bagian l_j, dibutuhkan operasi sebanyak:

Pembandingan: $\left(|B_j^c| - 1\right) \cdot l_j \; untuk \; 1 \le j \le v$

Penjumlahan:

$$\begin{cases} |B_j^c|.\, l_j + l_j, untuk \; j = 1, v \\ 2 \cdot |B_j^c| + |B_j^c|.\,(l_j - 1) + l_j \; , untuk \; 1 < j < v \end{cases}.$$

IV. SIMULASI

Bagian ini dibagi menjadi dua bagian, dengan bagian pertama menunjukkan, menganalisis, dan membandingkan kompleksitas komputasi dari BCH untuk pendekodean algoritme MAP dengan SOVA berdasarkan bentuk *trellis* level bit dan terbagi. Bagian kedua menunjukkan dan membandingkan hasil simulasi unjuk kerja BER melalui saluran AWGN dari masing-masing kode dengan bentuk *trellis* terbagi optimal dengan pendekodean MAP dan SOVA.

Dalam simulasi ini penulis melakukan pencarian nilai kompleksitas dengan asumsi bahwa operasi perkalian lima kali lebih kompleks daripada operasi penjumlahan. Selanjutnya *trellis* yang mencapai nilai kompleksitas terkecil dengan asumsi tersebut dikatakan *trellis* terbagi optimal. Selain mencari *trellis* terbagi optimal, pada simulasi ini juga ditunjukkan *trellis* dengan total banyak operasi terkecil atau disebut *trellis* terbagi minimal. Perlu diperhatikan bahwa *trellis* terbagi minimal dapat diperoleh dengan menganggap kompleksitas operasi perkalian sama dengan penjumlahan.

Penghitungan kompleksitas komputasi untuk metode MAP [9] dan metode SOVA [3] diimplementasikan untuk kode BCH (7,4) dan BCH (15,11) dengan bentuk *trellis* level-bit dan terbagi optimal. Hasil penghitungan kompleksitas komputasi untuk kode BCH (7,4) dengan metode MAP dapat dilihat pada Tabel I, hasil penghitungan kompleksitas komputasi untuk kode BCH (15,11) dengan metode MAP dapat dilihat pada Tabel II, hasil penghitungan kompleksitas komputasi untuk kode BCH (7,4) dengan metode SOVA dapat dilihat pada Tabel III, dan hasil penghitungan kompleksitas komputasi untuk kode BCH (15,11) dengan metode SOVA dapat dilihat pada Tabel IV.

TABEL I
KOMPLEKSITAS KOMPUTASI KODE BCH (7,4) DENGAN METODE MAP

	Level-bit MAP	***Trellis* Terbagi Optimal MAP**	***Trellis* Terbagi Minimal MAP**
Batas Optimum		{0,7}	{0,1,6,7}
Penjumlahan	60	202	137
Perkalian	182	23	58
# Operasi	242	225	195
Kompleksitas	970	319	427

TABEL II
KOMPLEKSITAS KOMPUTASI KODE BCH (15,11) DENGAN METODE MAP

	Level-bit MAP	***Trellis* Terbagi Optimal MAP**	***Trellis* Terbagi Minimal MAP**
Batas Optimum		{0,6,7,8,9,15}	{0,1,2,3,5,6,7, 8,9,10,12,15}
Penjumlahan	524	1494	630
Perkalian	1222	702	994
# Operasi	1746	2196	1624
Kompleksitas	6634	5004	5600

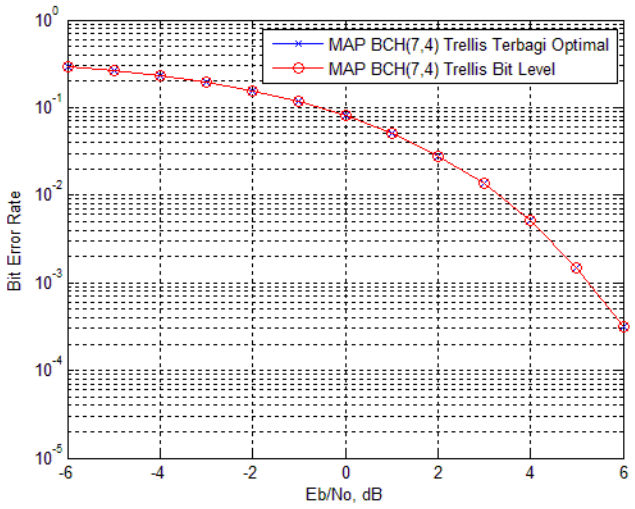
TABEL III
KOMPLEKSITAS KOMPUTASI KODE BCH (7,4) DENGAN METODE SOVA

	Level-bit SOVA	***Trellis* Terbagi Optimal SOVA**
Batas Optimum		{0,1,6,7}
Pembandingan	45	58
Penjumlahan	163	104
Kompleksitas	208	162

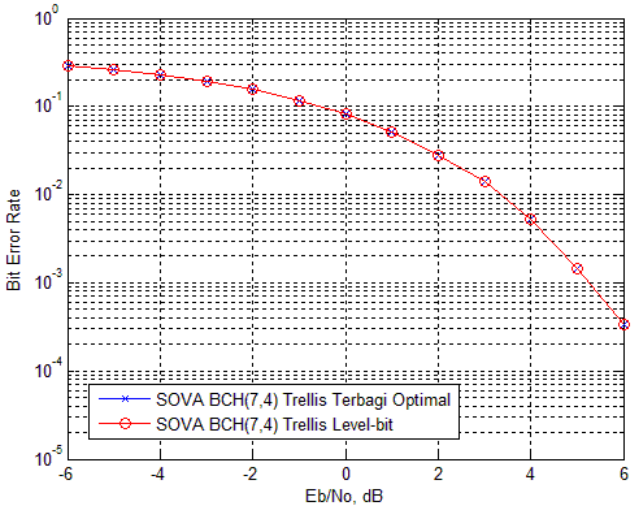
TABEL IV
KOMPLEKSITAS KOMPUTASI KODE BCH (15,11) DENGAN METODE SOVA

	Level-bit SOVA	***Trellis* Terbagi Optimal SOVA**
Batas Optimum		{0,1,2,3,5,6,7,8,9,10,12,15}
Pembandingan	381	402
Penjumlahan	1115	1039
Kompleksitas	1496	1441

Dari Tabel I terlihat kompleksitas optimal untuk kode BCH (7,4) dengan metode MAP tercapai pada saat batas bagian {0,7} dan mencapai total banyak operasi minimal ketika batas bagian adalah {0,1,6,7}. Perlu diperhatikan bahwa cabang yang terbentuk dari batas bagian {0,7} memiliki label yang sama dengan semua kemungkinan kode yang dihasilkan oleh kode BCH (7,4). Dari Tabel I terlihat perbedaan kompleksitas kode untuk *trellis* level-bit dengan *trellis* terbagi optimal dan minimal. Penggunaan *trellis* terbagi optimal menghasilkan penurunan kompleksitas sebesar 651 (970-319) atau 67,11%, sedangkan penggunaan *trellis* terbagi minimal menghasilkan penurunan kompleksitas sebesar 543 (970-427) atau 55,97%. Sementara, banyak operasi mengalami penurunan sebesar 7,02% untuk *trellis* terbagi optimal dan 19,42% untuk *trellis* terbagi minimal.

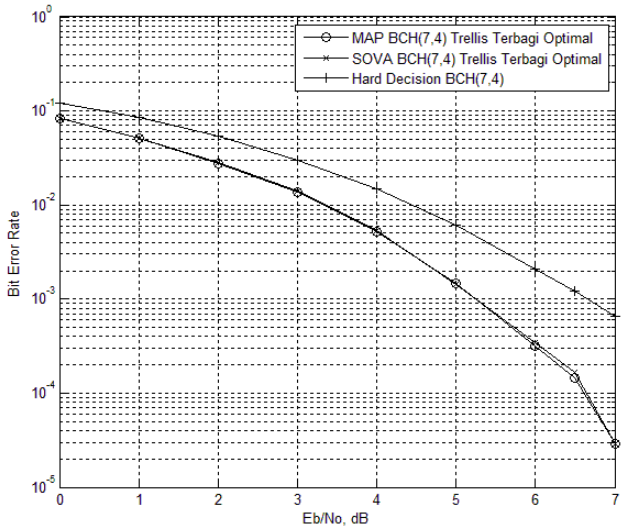


Gbr. 4 Unjuk kerja kesalahan bit untuk kode BCH (7,4) berdasarkan *trellis* terbagi optimal dan level-bit dengan pendekodean MAP.

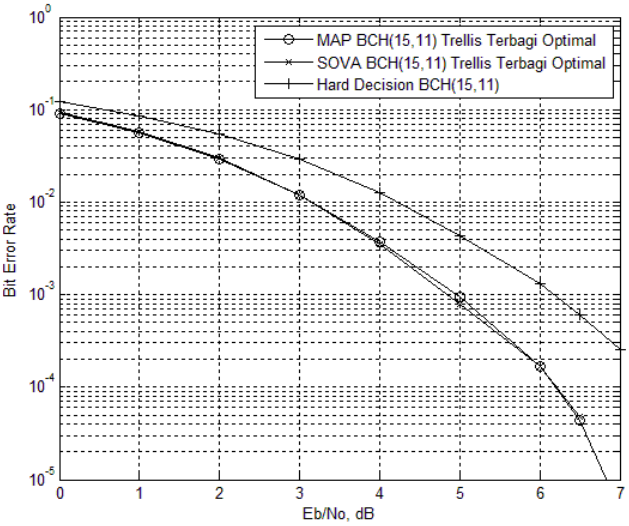


Gbr. 5 Unjuk kerja kesalahan bit untuk kode BCH (7,4) berdasarkan *trellis* terbagi optimal dan level-bit dengan pendekodean SOVA.

Dari Tabel II terlihat kode BCH (15,11) mencapai kompleksitas optimum pada batas bagian {0,6,7,8,9,15} dan minimal pada batas bagian {0,1,2,3,5,6,7,8,9,10,12,15}. Pada saat optimum, kompleksitas mengalami penurunan sebesar 24,57%, tetapi mengalami kenaikan pada banyak operasi sebesar 25,77%. Banyak operasi minimal pada kode ini adalah sebanyak 1624 operasi atau turun sebesar 6,9% dibanding banyak operasi pada *trellis* level-bit.



Gbr. 6 Unjuk kerja kesalahan bit untuk kode BCH (7,4) dengan metode pendekodean MAP, SOVA, dan secara keras.



Gbr. 7 Unjuk kerja kesalahan bit untuk kode BCH (15,11) dengan metode pendekodean MAP, SOVA, dan secara keras.

Pada pencarian kompleksitas komputasi SOVA, diasumsikan kebutuhan proses operasi pembandingan sama dengan operasi penjumlahan. Pada simulasi ini, batas optimum pada *trellis* terbagi untuk SOVA sama dengan batas optimum pada *trellis* terbagi minimal untuk MAP. Ini dikarenakan asumsi operasi yang berlaku pada kedua metode adalah sama. Dari hasil perhitungan pada Tabel I sampai Tabel IV, secara umum kompleksitas SOVA pada *trellis* level-bit lebih unggul dibanding MAP. Pada kode BCH (7,4) perbedaan kompleksitas mencapai 762 dan untuk kode (15,11) perbedaan kompleksitas sebanyak 5138.

Perubahan bentuk *trellis* menjadi *trellis* terbagi optimal juga berdampak pada pengurangan kompleksitas pada SOVA. Dari Tabel III, kode BCH (7,4) mengalami penurunan kompleksitas sebesar 22,11% dan dari Tabel IV, kode BCH (15,11) mengalami penurunan sebesar 3,6%. Jika dibandingkan dengan penurunan kompleksitas dari bentuk *trellis* terbagi optimal ke *trellis* level bit, pendekodean MAP mengalami penurunan kompleksitas yang lebih tinggi dibandingkan dengan pendekodean SOVA.

Selanjutnya, hasil pengujian BER dari masing-masing kode dengan pendekodean algoritme MAP dibandingkan dan dievaluasi berdasarkan dua model sistem, yaitu kode blok dan kode blok tersambung serial. Pengujian ini merupakan pengukuran dasar dari unjuk kerja sistem untuk mengetahui seberapa tepat bit yang dikirim dari awal sampai tujuan. BER adalah banyak bit galat yang diterima dibagi dengan total banyak bit yang diterima. Sebagai contoh, transmisi yang memiliki BER sebesar 10^{-5} memiliki arti bahwa rata-rata untuk setiap 1 dari 100.000 bit yang ditransmisikan mengalami kesalahan. Jika BER semakin tinggi maka hal tersebut mengindikasikan bahwa sistem komunikasi data tersebut tidak handal.

Pendekodean secara halus membutuhkan bentuk *trellis* dari suatu kode. Oleh karena itu, pada simulasi BER pertama ini algoritme MAP dan SOVA diterapkan sesuai bentuk *trellis* level-bit dan *trellis* terbagi optimal untuk kode BCH (7,4). Gbr. 4 menunjukkan unjuk kerja BER dari kode BCH (7,4) dengan pendekodean MAP. Dari pengamatan diketahui bahwa *trellis* terbagi dan *trellis* level-bit menghasilkan nilai BER yang sama untuk algoritme MAP. Begitu pula pada Gbr. 5, ditunjukkan bahwa *trellis* terbagi dan level-bit menghasilkan nilai BER yang sama untuk SOVA. Untuk selanjutnya, simulasi BER dilakukan terhadap *trellis* terbagi optimal saja.

Hasil simulasi pada Gbr. 6 dan Gbr. 7 menunjukkan bahwa pendekodean MAP pada kode blok jauh lebih optimal dibandingkan dengan pendekodean SOVA dan pengkodean secara keras. Unjuk kerja BER untuk semua kode BCH yang diujikan lebih optimal dibanding pendekodean secara keras sejak $E_b/N_0 = 0$ sampai $E_b/N_0 = 7$. Untuk kode BCH (7,4) dihasilkan keunggulan sebesar 1,5dB untuk BER pada 10^{-3} dibandingkan dengan metode pengkodean SOVA dan pengkodean secara keras. Sedangkan pada kode BCH (15,11) dicapai keunggulan dengan nilai 1,25dB untuk nilai BER pada 10^{-3}.

V. KESIMPULAN

Dari hasil penghitungan kompleksitas komputasi dapat ditarik kesimpulan bahwa penggunaan bentuk *trellis* terbagi yang optimal dapat mengurangi kompleksitas komputasi. Kompleksitas komputasi untuk metode MAP masih tinggi jika dibandingkan dengan metode SOVA.

Dalam hal unjuk kerja BER, perubahan bentuk *trellis* dari level-bit menjadi terbagi tidak mempengaruhi unjuk kerja BER. Hasil simulasi menunjukkan bahwa pendekodean secara halus dengan metode MAP lebih unggul dibandingkan pengkodean secara halus dengan metode SOVA dan pendekodean secara keras.

Dapat dikatakan bahwa pendekodean secara halus dengan metode MAP lebih unggul dibanding dengan pengkodean secara halus dengan metode SOVA dan pendekodean secara keras, dengan harga berupa perhitungan kompleksitas komputasi yang lebih tinggi. Penghitungan kompleksitas komputasi dapat dikurangi dengan penggunaan bentuk *trellis* terbagi yang optimal.

REFERENSI

[1] F.R. Kschischang, V. Sorokine, "On the Trellis Structure of Block Codes," *IEEE Transactions on Information Theory*, Vol. 41, No. 6, 1995.

[2] J. Hagenauer, E. Offer, L. Papke, "Iterative Decoding of Binary Block and Convolutional Codes," *IEEE Transactions on Information Theory*, Vol. 42, No. 2, 1996.

[3] F. Labeau, "Low-complexity nonbinary SOVA for sectionalized trellises," *Proceedings of Wireless Communications and Networking Conference*, 2004.

[4] A. Lafourcade, A. Vardy, "Optimal Sectionalization of a Trellis," *IEEE Transactions on Information Theory*, Vol. 42, No. 3, 1996.

[5] T.H. Chen, K.C. Chen, M.C. Lin, C.F. Chang, "On A* Algorithms for Decoding Short Linear Block Codes," *IEEE Transactions on Communications*, Vol. 63, No. 10, 2015.

[6] X. Li, W. Zhang, Y. Liu, "Efficient architecture for algebraic soft-decision decoding of Reed–Solomon codes," *IET Communications*, Vol. 9, No. 1, 2015.

[7] L. Bahl, J. Cocke, F. Jelinek, J. Raviv, "Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate," *IEEE Transactions on Information Theory*, Vol. 20, No. 2, 1974.

[8] D. Chandra, B. Setiyanto, S.S. Kusumawardani, "Implementasi pada FPGA atas Soft-Output Viterbi Algorithm (SOVA) untuk Pengawasandian Turbo," *Jurnal Nasional Teknik Elektro dan Teknologi Informasi*, Vol. 2, No. 4, 2013.

[9] Y. Liu, S. Lin, M.P.C. Fossorier, "MAP Algorithms for Decoding Linear Block Codes Based on Sectionalized Trellis Diagrams," *IEEE Transactions on Communications*, Vol. 48, No. 4, 2000.