논문 99-24-8A-15 한국통신학회논문지 '99-8 Vol.24 No.8A

터보 코드의 복호화를 위한 계산량을 줄인 슬라이딩 윈도우 BCJR 알고리즘

정회원 곽 지 혜*, 양 우 석*, 김 형 명*

Computationally Efficient Sliding Window BCJR Decoding Algorithms For Turbo Codes

Ji-Hye Gwak*, Woo-Seok Yang*, Hyung-Myung Kim* *Regular Members*

요 약

BCJR 알고리즘을 바탕으로 한 슬라이딩 윈도우 BCJR 알고리즘은 Turbo code의 복호시 격자 종결이 필요없으며 적은 메모리를 사용한다는 장점이 있지만, BCJR 알고리즘보다 더 많은 계산량을 필요로 하며 적합한 윈도우 길이에 대한 연구 결과가 없다는 단점이 있다. 이 논문에서는 슬라이딩 윈도우 BCJR 알고리즘의 장점을 유지하면서 계산량을 줄이고 그 성능을 개선시킬 수 있는 효율적인 슬라이딩 윈도우 BCJR 알고리즘을 제안한다. 먼저 적합한 윈도우 길이를 선택하기 위한 지침을 제시하고 윈도우의 이동폭을 크게 하여 그 계산량을 줄이도록 하였다. 모의 실험을 통하여 제안한 방식이 윈도우의 이동폭과 길이를 적절히 선택하면 기존의 슬라이딩 윈도우 방식보다 계산량을 줄이면서 성능은 향상시킬 수 있음을 확인하였다. 구속장 길이가 3인 부호화기에서 BER이 10^{-4}일 때, 제안한 알고리즘을 사용하면 기존의 슬라이딩 윈도우 방식보다 더 적은 계산량으로 0.1dB 정도의 이득을 얻을 수 있다.

ABSTRACT

In decoding the turbo codes, the sliding window BCJR algorithm, derived from the BCJR algorithm, permits a continuous decoding of the coded sequence without requiring trellis fermination of the constituent codes and uses reduced memory span. However, the number of computation required is greater than that of BCJR algorithm and no study on the effect of the window length has been reported. In this paper, we propose an efficient sliding window type scheme which maintains the advantages of the conventional sliding window algorithm, reduces its computational burdens, and improves its BER performance. A guideline is first presented to determine the proper window length and then a computationally efficient sliding window BCJR algorithm is obtained by allowing the window to be forwarded in multi-step. Simulation results show that the proposed scheme outperforms the conventional sliding window BCJR algorithm with reduced complexity. It gains 0.1dB SNR improvements over the conventional method for the constraint length 3 and BER 10^{-4}

I. 서 론

1993년 Berrou et al. [1]에 의해 제안된 터보 코드는 Shannon limit에 접근하는 우수한 성능을 가지는 채널 코딩 기법이다. 정보 비트열은 인터리버를 (interleaver) 사이에 두고 병렬적으로 연결된 두 개의 구성 부호화기로부터 (component encoder) 각각 부호화된다. 구성 부호화기로는 RSC (Recursive Systematic Convolutional) 부호화기가 사용된다.

복호화기에서는 두 개의 구성 복호화기가 (com-

* 한국과학기술원 전기 및 전자공학과
논문번호 : 99089-0305, 접수일자 : 1998년 3월 5일

ponent decoder) 사용된다. 구성 복호화기에서 사용되는 복호 알고리즘은 여러 종류가 있는데, 이 알고리즘들은 모두 소프트 입력을 받아들여 소프트 출력을 내보낸다. 소프트 입출력은 해당하는 비트가 0인지 1인지를 판별할 뿐 아니라, 그 판별의 신뢰도가 어느 정도인지를 함께 나타낸다. 터보 복호화기에서는 두 개의 구성 복호화기가 번갈아 가면서 동작한다. 구성 복호화기는 다른 구성 복호화기의 소프트 출력으로부터 extrinsic 정보를 구해서 이 값을 사전 확률값으로 (a priori probability) 이용하여 복호한다. 이런 과정을 반복하여 터보 코드는 우수한 성능을 얻을 수 있다. Berrou et al.이 터보 코드를 처음 제안했을 때 사용한 구성 복호화기의 복호 알고리즘은 MAP (Maximum A Posteriori) 알고리즘이다.[2] MAP 알고리즘은 1974년 알고리즘을 처음 제안한 저자 네 명의 이름 첫 글자를 따서 BCJR 알고리즘이라고도 불린다. BCJR 알고리즘은 각 입력 정보 비트의 사후 확률을 계산하여 입력 정보 비트의 오류가 최소가 되도록 복호를 한다. BCJR 알고리즘은 격자의 가능한 모든 경로를 고려하므로 계산량이 많으면서도, 대부분의 경우 Viterbi 알고리즘과 성능이 비슷하여 실제로는 거의 쓰이지 않았다. 그러다가 복호화기에서 소프트 출력을 구해야 하는 터보 코드에서 사용되면서 새롭게 주목받기 시작했다.

각 구성 복호화기에서 소프트 출력을 구하기 위해 사용할 수 있는 복호 알고리즘은 여러 가지가 있는데, 크게 두 종류로 나눌 수 있다.

첫번째는, BCJR 알고리즘에 바탕을 두고 있는 알고리즘이다. BCJR 알고리즘은 성능은 우수하지만 계산량이 많아 계산량을 줄이기 위한 노력들이 지금까지 많이 있어 왔다.[3][4][5] 두번째는, Viterbi 알고리즘을 변형한 알고리즘이다. SOVA (Soft-Output Viterbi Algorithm)[6],[7]는 기존의Viterbi 알고리즘을 적용하여 얻을 수 있는 hard-decision 값에 신뢰도 값을 더 계산한다. SOVA는 BCJR 알고리즘보다 성능은 나쁘지만 계산량이 적다는 장점이 있다. SOVA의 복호화의 복잡도는 (decoding complexity) Viterbi 알고리즘의 1.8배 정도이므로 실제 구현에 유리하다.

기존의 길쌈 코드와 달리 터보 코드의 격자 종결은 (trellis termination) 인터리버의 사용으로 인해 쉽지 않은 일이다. 터보 코드에서는 인터리버 길이만큼의 정보 비트열을 한 프레임으로 보고, 프레임 단위로 복호화가 이루어지는데 BCJR 알고리즘은 매 프레임마다 격자 종결이 이루어졌다는 가정 하에 복호를 한다.

1996년 Benedetto et al.[8]에 의해 정보 비트열을 프레임 단위로 나누지 않고, 격자 종결이 필요 없는 연속적인 복호화에 적합한 슬라이딩 윈도우 BCJR 알고리즘이 발표되었다. 하지만 [8]에서 저자는 윈도우 길이가 어느 정도의 값을 가지는지, 그리고 그 값을 어떻게 결정할 것인지에 대한 언급은 하지 않았다. 이 논문에서는 윈도우 길이를 선택하기 위한 지침을 제시하고자 한다.

음성 전송처럼 적은 지연을 (delay) 필요로 하는 경우에 인터리버 길이가 짧아지는데, 인터리버 길이가 짧아질수록 격자 종결이 필요 없는 슬라이딩 윈도우 BCJR 알고리즘이 유리하다. 하지만, 슬라이딩 윈도우 BCJR 알고리즘은, 실제 구현에 있어 계산량이 많다는 것이 가장 큰 단점인 BCJR 알고리즘보다 계산량이 더 많아지므로 계산량을 줄일 필요성이 있다. 이 논문에서는 시간축을 따라 이동하는 윈도우의 이동폭을 크게 해서, 슬라이딩 윈도우 BCJR 알고리즘의 계산량을 줄이는 방법을 제안하였다.

제안한 알고리즘을 적용할 때, 슬라이딩 윈도우 BCJR 알고리즘과 똑같은 길이의 윈도우를 사용한다면 계산량을 줄일 수 있지만 대신 성능이 나빠진다. 제안한 알고리즘과 슬라이딩 윈도우 BCJR 알고리즘의 계산량을 같게 한다면, 제안한 알고리즘에서 사용하는 윈도우 길이를 크게 할 수 있으므로 성능을 향상시킬 수 있다.

2장에서는 BCJR 알고리즘과 슬라이딩 윈도우 BCJR 알고리즘에 대해 기술하고 BCJR 알고리즘의 문제점을 지적한다. 3장에서는 슬라이딩 윈도우 BCJR 알고리즘에서 윈도우의 길이를 선택하기 위한 지침을 제시한다. 그런 뒤, 슬라이딩 윈도우 BCJR 알고리즘의 문제점을 지적하고, 이를 해결하기 위한 알고리즘을 제안하다. 또, 각 알고리즘의 계산량과 성능, 그리고 메모리 사용량을 비교한다. 각 알고리즘들에 대한 모의 실험의 결과와 그에 대한 분석은 4장에서 하기로 한다. 5장에서 결론을 내린다.

Ⅱ. BCJR 알고리즘과 슬라이딩 윈도우 알고리즘

알고리즘을 기술하기에 앞서 이 논문에서 사용할 표기들을 먼저 정의하기로 한다. 코드율이 1/3인 부호화기와 길이 N의 인터리버를 사용한다고 가정

한다. 인터리버의 사용으로 인해, 부호화와 복호화는 인터리버 길이 N의 프레임을 기준으로 이루어진다. 시간 k가 1에서 N까지의 값을 가질 때, 입력정보 력 비트열 $\mathbf{d}=(d_1, d_2, \cdots, d_N)$, $d_k \in \{0,1\}$이고, 각 구성 부호화기로부터의 출력인 패리티 비트열은 $\mathbf{v}_1=(v_{11}, v_{12}, \cdots, v_{1N})$, $v_{1k} \in \{0,1\}$, $\mathbf{v}_2=(v_{21}, v_{22}, \cdots, v_{2N})$, $v_{2k} \in \{0,1\}$이다. $\mathbf{v}_2$는 인터리버를 통과한 정보 비트열 $\mathbf{d}'=(d_1', d_2', \cdots, d_N')$, $d_k' \in \{0,1\}$을 부호화한 것이다. 그리고, 각 구성 부호화기의 코드워드는 $c_{1k}=(d_k, v_{1k})$, $c_{2k}=(d_k', v_{1k})$이다.

부호화기의 출력이 가산성 백색 정규 잡음 채널을 통과하였다고 가정하면, 이 때 복호화기의 입력은 $x_k=(2 \cdot d_k-1)+n_{0k}$, $y_{1k}=(2 \cdot v_{1k}-1)+n_{1k}$, $y_{2k}=(2 \cdot v_{2k}-1)+n_{2k}$, $k=1,2,\cdots,N$ 이고, n_{0k}, n_{1k}, n_{2k}는 평균이 0이고, 분산이 σ^2인 독립적인 정규 잡음이다.

구성 복호화기는 채널을 통과한 시스터메틱 비트와 패리티 비트, 그리고 다른 복호화기로부터 얻은 정보를 사전 확률값으로 이용하며 이 값을 extrinsic 정보라고 한다.[1] 두 구성 복호화기를 DEC 1, DEC 2라 할 때 DEC 1이 DEC 2로부터 얻는 extrinsic 정보열을 $\mathbf{L}_{2e}=(L_{2e}(d_1), L_{2e}(d_2), \cdots, L_{2e}(d_k), \cdots, L_{2e}(d_N))$이라 하고,DE C 2가 DEC 1으로부터 얻는 extrinsic 정보열을 $\mathbf{L}_{2e}=(L_{1e}(d_1'), L_{1e}(d_2'), \cdots, L_{1e}(d_k'), \cdots, L_{1e}(d_N'))$이라 한다.

Extrinsic 정보를 얻기 위해서는 구성 복호화기가 소프트 출력을 계산해야만 한다. 소프트 출력은 입력 정보 비트의 사후 확률의 (a posteriori probability) 비를 구해 로그값으로 변환한 LLR로 (Log Likelihood Ratio) 나타낸다.

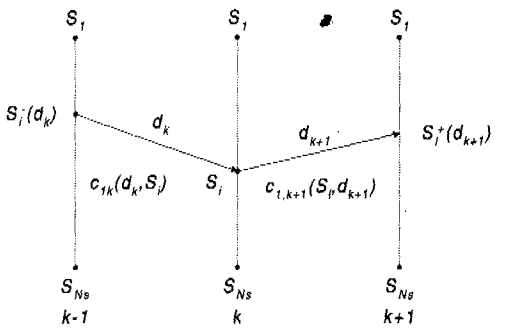

그림 2.1 논문에서 사용되는 표기법

터보 코드의 부호화기에서 사용하는 레지스터의 상태의 수가 N_s일 때, DEC 1에서 k번째 시간에 대해 그림 2.1과 같은 표기법을 사용하기로 한다.

- S_i는 집합 $\mathbf{S}=\{S_1, \cdots, S_{N_s}\}$에 속하며 시간 k에서의 상태이다.
- $S_i^-(d_k)$는 천이 $S_i^-(d_k) \rightarrow S_i$ 에 해당하는 입력 비트 d_k에 의해 정해 지는 S_i의 precursor이다.
- $S_i^+(d_{k+1})$은 천이 $S_i \rightarrow S_i^+(d_{k+1})$에 해당하는 입력 비트 d_{k+1}에 의해 정해지는 S_i의 postcursor이다.
- $c_{1k}(d_k, S_i)$는 해당되는 천이가 S_i에서 끝나는 경우이고, $c_{1,k+1}(S_i, d_{k+1})$은 해당되는 천이가 S_i에서 시작되는 경우를 의미한다.
- 상태열 $\mathbf{s}=(s_0, s_1, \cdots, s_N)$이다. 상태열은 시간에 따른 상태의 변화를 나타낸다. 즉, 소문자 s_k는 시간 k번째의 상태를 의미하고, 대문자 S_i는 집합 S에 속하는 특정한 상태를 의미한다.

1. BCJR 알고리즘

BCJR 알고리즘에서는 입력 정보 비트열의 사후 확률열을 계산하여 소프트 출력 LLR을 얻는다. DEC 1의 LLR은 다음과 같이 정의한다.[1]

$$\Lambda_1(d_k)=\ln \frac{P\{d_k=1 \mid \text{observations}\}}{P\{d_k=0 \mid \text{observations}\}} \tag{2.1}$$

여기서 $P\{d_k=d \mid \text{observations}\}$, $d=0,1$은 채널 출력 $\mathbf{x}, \mathbf{y}_1$과 사전 확률값이 주어졌을 때의 사후 확률이다.

먼저 시간 $k=1,2,\cdots,N$ 이고, 상태의 수는 N_s라 한다. 그리고, 상태열 $\mathbf{s}=(s_0, \cdots, s_N)$에서 처음과 마지막 상태 s_0, s_N을 수신단에서 알고 있다고 가정을 한다. 그리고, DEC 1에 대해 시간 k, 상태 S_i에서 사후 천이 확률을 (a posteriori transition probability) 다음과 같이 정의한다.

$$\sigma_k(S_i, d)=P\{d_k=d, s_{k-1}=S_i \mid \mathbf{x}, \mathbf{y}_1, \mathbf{L}_{2e}\}, \quad d=0,1 \tag{2.2}$$

그러면 APP는 (a posteriori probability) 사후 천이 확률을 이용하여 구할 수 있다.

$$P(d_k=d \mid \mathbf{x}, \mathbf{y}_1, \mathbf{L}_{2c})=\mathrm{P_k}(\mathrm{d} \mid \mathbf{x}, \mathbf{y}_1, \mathbf{L}_{2e}) \tag{2.3}$$

$$= \sum_{S_i} \sigma_k(S_i, d) \tag{2.4}$$

따라서, APP를 구하기 위해서는 사후 천이 확률을 구하면 되고, 이 값은 아래 식으로부터 구할 수 있다.

$$\sigma_k(S_i, d) = h_\sigma \alpha_{k-1}(S_i)\gamma_k(c(S_i, d))\beta_k(S_i^+(d)) \tag{2.5}$$

여기서 h_σ는 $\sum_{S_i, d}\sigma_k(S_i, d) = 1$이 되도록 하는 정규화 상수이다. $\gamma_k(c(S_i, d))$는 결합 확률로 다음과 같이 정의되며,

$$\gamma_k(c(S_i, d)) = P(x_k, y_{1k}, c_{1k} = c) \tag{2.6}$$

$$= P(x_k, y_{1k} \mid c_{1k} = c)P(c_{1k} = c) \tag{2.7}$$

채널 입력 c의 사전 확률과 채널의 천이 확률 $P(x_k, y_{1k} \mid c_k = c)$를 이용하여 계산한다.

$\alpha_k(S_i)$는 시간 k에서, k와 k 이전의 수신 신호가 주어졌을 때 각 상태에 머물러 있을 확률이다. 즉, $\alpha_k(S_i) = P(s_k = S_i \mid x_i^k, y_{1,1}^k)$이고, $y_{1,1}^k$는 $y_{1,1}$, $y_{1,2}, \cdots, y_{1,k}$열을 나타낸다. $\alpha_k(S_i)$는 forward recursion에 의해 계산할 수 있다.

$$\alpha_k(S_i) = h_\alpha \sum_d \alpha_{k-1}(S_i^-(d))\gamma_k(c(d, S_i)) \tag{2.8}$$

h_α는 앞에서와 마찬가지로 $\sum_{S_i}\alpha_k(S_i) = 1$을 만족시키는 정규화 상수이며, 초기 상태를 알고 있으므로 forward recursion의 초기화는 다음과 같이 한다.

$$\alpha_0(S_i) = \begin{cases} 1 & \text{if } S_i = s_0 \\ 0 & \text{otherwise} \end{cases} \tag{2.9}$$

$\beta_k(S_i)$는 시간 k에서, k 이후의 수신 신호가 주어졌을 때 각 상태에 머물러 있을 확률이다. 즉, $\beta_k(S_i) = P(s_k = S_i \mid x_{k+1}^N, y_{1,k+1}^N)$이고 $\beta_k(S_i)$는 backward recursion에 의해 계산할 수 있다.

$$\beta_k(S_i) = h_\beta \sum_d \beta_{k+1}(S_i^+(d))\gamma_{k+1}(c(S_i, d)) \tag{2.10}$$

h_β 역시 $\sum_{S_i}\beta_k(S_i) = 1$을 만족시키는 정규화 상수이며, backward recursion의 초기화는

$$\beta_N(S_i) = \begin{cases} 1 & \text{if } S_i = s_N \\ 0 & \text{otherwise} \end{cases} \tag{2.11}$$

와 같이 한다.

$\gamma_k, \alpha_k, \beta_k$들을 식 (2.7), 식 (2.8), 식 (2.10)으로부터 각각 구한 후 식 (2.5)에 대입하여 사후 천이 확률값을 계산한다.

2. 슬라이딩 윈도우 BCJR 알고리즘

터보 코드의 부호화기에서는 정보 비트열을 인터리버 길이에 해당하는 프레임 단위로 나누어 프레임마다 격자 종결을 해 주어 송신하며 복호화도 프레임 단위로 이루어진다. 복호화기에서 사용하는 BCJR 알고리즘은 backward recursion을 포함하고 있으므로 한 프레임 전체를 다 받고 난 뒤에야 복호를 시작할 수 있다 (그림 2.2 참조). 그러나, 터보 코드는 구성 부호화기가 인터리버로 연결되어 있으므로 두 개의 구성 부호화기를 동시에 격자 종결시키는 것은 쉽지가 않다. 프레임마다 격자 종결을 시킬 경우, 몇 비트가 추가적으로 필요하므로 전체 코드율이 낮아진다. 그리고, 한 프레임 길이만큼의 α_k와 γ_k 값들을 저장해 두어야하므로 메모리를 많이 사용하게 된다.

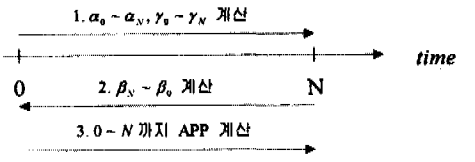

그림 2.2 BCJR 알고리즘의 계산 순서

앞서 말한 BCJR 알고리즘의 문제점을 보완하기 위해 격자 종결을 하지 않으며, 적은 메모리를 사용하는 슬라이딩 윈도우 BCJR 알고리즘이 1996년 Benedetto et al.[8]에 의해 제안되었다. 그림 2.3에서처럼 슬라이딩 윈도우 BCJR 알고리즘은 수신 데이터를 프레임 단위로 나누지 않고, 길이가 D인 윈도우가 시간축을 따라 이동하면서 LLR을 계산한다.

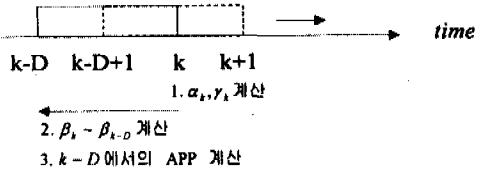

그림 2.3 슬라이딩 윈도우 BCJR 알고리즘의 계산 순서

슬라이딩 윈도우 BCJR 알고리즘에서는, 시간 k에서의 α_k값을 이용하여 β_k를 초기화시켜 D번 back-

ward recursion 한 후 $k-D$에서의 APP를 계산하고, 윈도우가 한 칸 이동해서 $k+1$에서도 마찬가지의 계산을 하므로 프레임에 상관없이 복호화가 이루어진다. 따라서, 슬라이딩 윈도우 BCJR 알고리즘에서는 격자 종결이 필요 없으며, 한 프레임 전체에 대해서가 아니라 윈도우가 위치한 곳의 α_k와 γ_k만 저장해 두면 되므로 적은 메모리를 사용한다.

한 프레임의 길이는 대개 인터리버의 길이와 같게 되는데, 터보 코드에서 사용하는 인터리버의 길이는 100비트 정도의 짧은 길이부터 10000비트 이상까지 넓은 범위의 값을 가진다. 무선 통신 시스템의 음성 전송에서는 인터리버에 의한 지연값이 적어야 하므로 일반적으로 사용되는 음성 프레임의 길이는 200비트 이내이다.[9] 따라서, 음성 전송처럼 지연이 적어야 하는 경우일수록 격자 종결을 하지 않는 슬라이딩 윈도우 BCJR 알고리즘이 유리하다.

Ⅲ. 계산량을 줄인 슬라이딩 윈도우 BCJR 알고리즘

1. 윈도우 길이 D의 선택

[8]에서 Benedetto et al.은 슬라이딩 윈도우 BCJR 알고리즘을 제안하였으나 윈도우 길이 D가 어느 정도의 값을 가지는지, 그리고 그 값을 어떻게 결정할 것인지에 대한 언급은 하지 않았다. 이 논문에서는 D를 선택하기 위한 지침을 제시하고자 한다.

길쌈 코드의 복호화에서 적은 메모리를 사용해 ML 복호화기를 구현하고자 할 때, 입력 정보 τ 비트에 해당하는 경로만을 고려해서 복호를 하는 경로 절단 방법을 사용하기도 한다. 이 때, 경로 절단에 의해 부가적인 오류가 발생하게 된다.

한편, 길쌈 코드의 WEF은 (Weight Enumerating Function) 다음과 같이 정의한다.

$$T(X,Y)=\sum_{i,j}A_{i,j}X^iY^j \tag{3.1}$$

여기서 $A_{i,j}$는 입력 정보 비트열의 weight가 j이고, 해당되는 코드워드의 weight가 i인 코드워드의 개수이다.

WEF이 $T(X,Y)$인 길쌈 코드의 ML 복호화에서 경로 절단을 할 때, 가산성 백색 정규 잡음 채널에 대한 비트 오류율의 상한은 아래의 식으로 주어진다.[10]

$$P_b<\frac{1}{k}\left[\frac{\partial T(X,Y)}{\partial Y}+\sum_{i=1}^{2^{K-1}-1}T_i^{\tau}(X,Y)\right]\Bigg|_{X=e^{-REb/No},\,Y=1} \tag{3.2}$$

여기서 $T_i^{\tau}(X,Y)$는 WEF에서 절단에 의해 오류를 발생시킬 수 있는 경로이고, τ는 절단된 경로의 길이이다. 그리고, R과 K는 각각 코드율과 구속장 길이를 뜻한다.

식 (3.2)에서 첫 번째 항은 ML 복호화에 의해 발생하는 비트 오류율이며, 두 번째 항은 절단에 의해 발생하는 비트 오류율이다. 이 식으로부터 Eb/No이 클 때, 절단에 의해 발생하는 비트 오류율이 ML 복호화에 의해 발생하는 비트 오류율에 비해 무시할 수 있을 만큼 작아지는 절단 경로의 길이를 구할 수 있다.

이 때의 $\tau_{\min}\approx 4\times(K-1)$이며, 작은 값의 Eb/No에 대해서는 $\tau_{\min}\approx 5.8\times(K-1)$이면 절단에 의해 더해지는 비트 오류율 값을 무시하기에 충분하다.[11] 여러 모의 실험의 결과와 실제 경험에 의해 절단된 경로의 길이는 부호화기에서 사용하는 메모리의 개수 $K-1$의 4~5배 정도의 τ면 충분하다는 것이 밝혀졌다.[11]

길쌈 코드의 ML 복호화기에서 경로 절단에 의해 발생하는 오류를 무시할 수 있는 τ 값이 어떻게 결정되어지는지 살펴보았다. 이 때, 식 (3.2)로부터 구한 절단 경로의 길이는 터보 복호화기의 각 구성 복호화기에 그대로 적용할 수 있으므로, 슬라이딩 윈도우 BCJR 알고리즘에서도 마찬가지 방법으로 윈도우의 길이를 정할 수 있다.

슬라이딩 윈도우 BCJR 알고리즘에서 윈도우 길이 D가 $5\times(K-1)$ 이상이면 BCJR 알고리즘과 비슷한 성능이 보장되므로 $D_{\min}=5\times(K-1)$로 두고, $D\geq D_{\min}$을 만족시키도록 선택한다. 하지만, β_k의 계산량이 BCJR 알고리즘보다 D배 증가하므로 경우에 따라 성능과 계산량을 고려하여 D를 결정한다.

2. 계산량을 줄인 슬라이딩 윈도우 BCJR 알고리즘

슬라이딩 윈도우 BCJR 알고리즘은 격자 종결이 필요 없으며 BCJR 알고리즘보다 적은 메모리를 사용한다는 장점이 있지만, 계산량은 오히려 BCJR 알고리즘보다 많다. 전체 입력 정보 비트열의 길이가 N일 때, 두 알고리즘의 α_k와 γ_k의 계산량은 동일

하지만 β_k의 계산량은 BCJR 알고리즘이 N번, 슬라이딩 윈도우 BCJR 알고리즘이 $N\times D$ 번으로 BCJR 알고리즘보다 D배가 더 많기 때문이다. 계산량이 많다는 것이 가장 큰 단점인 BCJR 알고리즘보다 계산량이 더 많으므로 계산량을 줄일 필요가 있다.

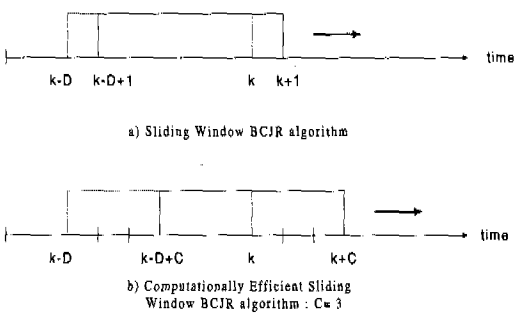

그림 3.1 윈도우의 이동

그림 3.1의 a)에서처럼 슬라이딩 윈도우 BCJR 알고리즘에서는 β_k값을 얻기 위해 윈도우가 이동하면서, 윈도우의 길이 D만큼 backward recursion을 한다. 이 논문에서는 윈도우의 이동폭을 증가시켜 β_k의 계산량을 줄이는 방법을 제안한다 (그림 3.1의 b) 참조). 즉, 시간 k에서부터 D번 backward recursion하여 구한 β_k값을 $k-D$에서 $k-D+C-1$까지 $C-1$개 사용한다.

시간 k가 0부터 ∞까지의 값을 가진다고 가정하고, 제안한 방법을 슬라이딩 윈도우 BCJR 알고리즘에 다음과 같이 적용한다. 알고리즘에서 $\beta_l^{(k)}$의 위첨자는 시간축에서 윈도우의 오른쪽 끝을 의미하고, 아래 첨자는 윈도우의 오른쪽 끝이 k에 정지하고 있을 때, l번째의 backward recursion을 나타낸다.

<계산량을 줄인 슬라이딩 윈도우 BCJR 알고리 즘>

1] $\alpha_0(S_i)$, $0\le i\le N_s-1$를 식 (2.9)에 따라 초 기화한다.

2] $k\ge 1$이면 x_k, y_{1k}를 수신하여 식 (2.7)에 의해 $\gamma_k(c)$를 구한 뒤,식 (2.8)에 따라 $\alpha_k(S_i)$를 구한다. 모든 S_i, c에 대해 $\alpha_k(S_i)$와 $\gamma_k(c)$를 저장한다.

3] $k\ge D$이면 $\beta_k^{(k)}(S_i)$를 다음과 같이 초기화시킨다.

$$\beta_k^{(k)}(S_i)=\alpha_k(S_i),\quad \forall S_i$$

4] (2.10)을 이용하여 $\beta_l^{(k)}(S_i)$, $k-D\le l\le k-1$ 를 구한다.

5] 시간 k-D 에서 k-D+C-1 까지 $\gamma_{k-D}(c)\sim\gamma_{k-D+C-1}(c)$, $\alpha_{k-D}(S_i)\sim\alpha_{k-D+C-1}(S_i)$, $\beta_{k-D}(S_i)\sim\beta_{k-D+C-1}(S_i)$ 값을 식 (2.5)에 대입하여 $\sigma_{k-D}(S_i,u)\sim\sigma_{k-D+C-1}(S_i,u)$를 계 산 한다.

6] 식 (2.4)를 이용하여 APP값 $P_{k-D}(u\mid \mathbf{y})\sim P_{k-D+C-1}(u\mid \mathbf{y})$를 계산한다. $k=k+1$로 증가시키고 1]로 간다.

슬라이딩 윈도우 BCJR 알고리즘과 제안한 알고리즘의 윈도우 길이가 같고 C가 1보다 크다면, 제안한 알고리즘의 계산량은 슬라이딩 윈도우 BCJR 알고리즘보다 줄어들고 성능은 나빠질 것이다. $\beta_k(S_i)$의 초기값을 정확하게 줄 수 없으므로, backward recursion을 할수록 β_k값이 정확해지기 때문이다. 하지만, 슬라이딩 윈도우 BCJR 알고리즘과 제안한 알고리즘의 계산량을 같게 한다면, 제안한 알고리즘에서는 윈도우의 길이를 더 증가시킬 수 있으므로 성능을 향상시킬 수 있다. 따라서, 윈도우 길이와 이동폭은 다음과 같이 선택한다.

기존의 슬라이딩 윈도우 BCJR 알고리즘을 사용할 때보다 성능을 저하시키고 싶지 않은 경우에는, 슬라이딩 윈도우 BCJR 알고리즘에서 윈도우 길이 D를 앞 절에서 제안한 $D_{\max}$이내의 값으로 선택한 뒤 $D+C$ 를 제안한 알고리즘에서 사용하는 윈도우 길이로 정한다. 그리고, 기존의 슬라이딩 윈도우 BCJR 알고리즘을 사용할 때보다 성능은 나빠지더라도 계산량을 줄이고 싶다면 윈도우 길이를 슬라이딩 윈도우 BCJR 알고리즘에서와 같도록 정한다. C는 필요한 계산량을 고려하여 선택한다.

(k_0, n_0) 부호이고, 상태의 수가 N_s, 코드워드의 수가 M일때, BCJR 알고리즘과 슬라이딩 윈도우 BCJR 알고리즘, 계산량을 줄인 슬라이딩 윈도우 BCJR 알고리즘의 계산량을 비교해 보자. 세 알고리즘 모두 α_k, γ_k의 계산량은 같고 β_k의 계산량에서 차이를 보인다. 앞서 언급했듯이 전체 입력 정보 비트열의 길이가 N일 때 BCJR 알고리즘의 β_k 계산량은 N번, 슬라이딩 윈도우 BCJR 알고리즘에서는 $N\times D$번이고 제안한 알고리즘에서는 $N\times\frac{D}{C}$번을 계산한다. α_k와 β_k를 갱신하기 위해 필요한 세 알고리즘의 계산량을 표 3.1에서 비교하였다.

표 3.1 각 알고리즘의 메모리 사용량 비교

	저장해야 할 γ의 수	저장해야 할 α의 수
BCJR	$N \times M$	$N \times N_s$
SW-BCJR	$D \times M$	$D \times N_s$
PSW-BCJR	$D \times M$	$D \times N_s$

표 3.1에서 SW-BCJR은 슬라이딩 윈도우 BCJR 알고리즘을, PSW-BCJR은 계산량을 줄인 제안한 알고리즘을 의미한다. 윈도우의 이동폭 C는 1과 $D+1$ 사이의 값을 가진다. $C=1$일 경우, 제안한 알고리즘은 슬라이딩 윈도우 BCJR 알고리즘과 동일하다. 세 알고리즘의 메모리 사용량은 표 3.2에서 비교하였다.

표 3.2 k마다 β를 갱신하기 위해 필요한 각 알고리즘의 계산량 비교

	2^{k_0}개의 덧셈의 계산량	곱셈의 계산량
BCJR	N_s	$N_s \times 2^{k_0}$
SW-BCJR	$D \times N_s$	$D \times N_s \times 2^{k_0}$
PSW-BCJR	$\frac{D}{C} \times N_s$	$\frac{D}{C} \times N_s \times 2^{k_0}$

Ⅳ. 모의실험 결과 및 분석

이 장에서는 슬라이딩 윈도우 BCJR 알고리즘과 계산량을 줄인 슬라이딩 윈도우 BCJR 알고리즘의 비트 오류율을 모의실험을 통해 비교해 보고, 그 결과를 분석해 보고자 한다.

모의실험에서는 코드율 1/3, 구속장 길이 $K=3,4$인 부호화기를 사용하였다. $K=3$일 때는 $\{7,5\}_8$, $K=4$일 때는 $\{17,15\}_8$ 인 생성 다항식을 사용하였다. 인터리버는 10×10의 블록 인터리버를 사용하였다. 그림 4.1~4.3에서 여러 경우에 대해 Eb/No에 대한 비트 오류율을 나타내었다.

각 그림들에서 SW는 슬라이딩 윈도우 BCJR 알고리즘을 의미하고 PSW는 제안한 알고리즘을 의미한다. D,C는 각각 윈도우의 길이와 이동폭이며, i는 터보 복호화기의 반복의 횟수를 의미한다. 1회의 반복은 DEC 1의 입력으로 들어간 수신 데이터가 DEC 1을 거쳐 DEC 2의 출력으로 나올 때까지이다.

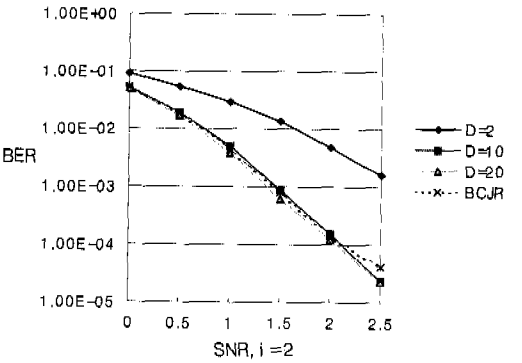

그림 4.1 SW1; D=2, D=10, D=20, BCJR

그림 4.1에서는 $K=3$일 때 윈도우 길이가 2, 10, 20인 SW-BCJR 알고리즘의 비트 오류율을 비교하였다. 윈도우 길이가 D_{min}보다 짧을 때는 성능이 많이 나쁘고 윈도우 길이를 증가시킬수록 성능이 좋아지지만, 윈도우 길이가 D_{min}이상이면 D_{min}일 때와 비교해 크게 성능이 좋아지지 않음을 알 수 있다. 즉 윈도우 길이가 D_{min}이 되면 경로 절단에 의한 오류가 무시할 수 있을 만큼 작아짐을 확인할 수 있다.

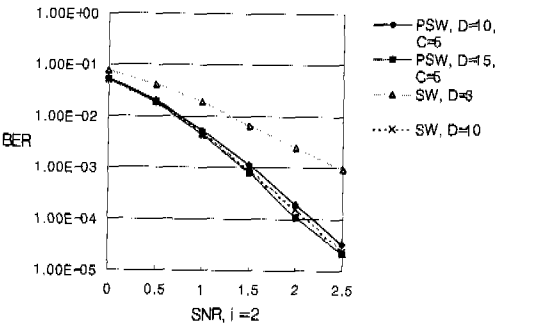

그림 4.2 SW; D=3, D=10 PSW; D=10, C=5 and D=15, C=5

그림 4.2에서는 $K=3$일 때, SW-BCJR 알고리즘과 PSW-BCJR 알고리즘의 비트 오류율을 비교하였다. SW-BCJR 알고리즘에서 사용한 윈도우의 길이는 3, 10이다. 제안한 알고리즘에서는 D_{min}을 기준으로 윈도우 길이를 결정하였다. 따라서 SW-BCJR

알고리즘과 비교하여 성능이 나빠지지 않도록 하기 위해서는 윈도우 길이를 15로 결정하고, 성능이 조금 나빠지더라도 계산량을 최대한 줄이기 위해서는 윈도우 길이를 10으로 결정할 수 있다.

그림 4.2에서 아래쪽의 세 선 중 성능이 가장 좋은 것이 제안한 알고리즘에서 길이가 15인 윈도우를 사용했을 때이고, 성능이 가장 나쁜 것이 제안한 알고리즘에서 길이가 10인 윈도우를 사용했을 때이다. 윈도우 길이가 10일 때는 $D_{\min}$보다 적은 개수의 데이터를 사용하여 복호화를 하게 되는 경우가 발생하므로 성능이 나빠지는 현상이 생긴다. 한편 SW-BCJR 알고리즘에서는 길이가 3인 윈도우를 사용한 경우가 성능이 가장 나쁜데, 이 경우는 $D=10, C=5$일 때와 계산량이 같다.
그림 4.3에서는 $K=4$일 때 그림 4.2에서와 마찬가지 경우에 대해 비트 오류율을 비교하였다.

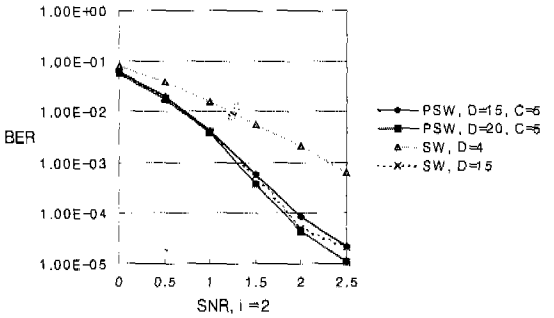

그림 4.3 SW; D=4, D=15 PSW; D=20, C=5 and D=15, C=5

V. 결 론

지금까지 터보 코드의 복호화기에서 사용할 수 있는 소프트 출력 복호 알고리즘들을 살펴보았다. Berrou et al.이 터보 코드를 처음 제안했을 때 사용한 BCJR 알고리즘은 성능은 우수하지만 계산량이 많다는 단점이 있다. 그리고, 복호화가 프레임 단위로 이루어지고, 매 프레임마다 격자 종결을 해 주어야 한다.

이에 반해, 슬라이딩 윈도우 BCJR 알고리즘은 복호화를 프레임 단위로 나누지 않고, 윈도우가 시간축을 따라 이동하면서 윈도우 구간의 수신 데이터를 이용하여 복호를 한다. 이 알고리즘을 사용할 경우 격자 종결이 필요 없으며 적은 메모리를 필요로 한다. 하지만, 실제 구현에 있어 계산량이 많다는 것이 가장 큰 걸림돌인 BCJR 알고리즘보다 계산량이 더 많아지므로 계산량을 줄일 필요성이 있다.

이 논문에서는 먼저 윈도우 길이 D를 선택하기 위한 지침을 제시한 뒤, 슬라이딩 윈도우 BCJR 알고리즘의 계산량을 줄이기 위해 윈도우의 이동폭을 크게 하는 방식을 제안하였다. 윈도우의 이동폭과 길이를 적절히 선택하여 제안한 알고리즘을 사용할 경우, 계산량을 줄이면서 성능도 향상시킬 수 있다. 슬라이딩 윈도우 BCJR 알고리즘과 제안한 알고리즘의 성능을 모의실험을 통해 비교하여 그 결과를 살펴보았다. 이 논문에서 윈도우의 길이를 선택하기 위한 지침으로 길쌈 보드의 절단 경로를 사용하였는데, 터보 코드의 전반적 특성을 반영한 윈도우 길이 선택에 관한 연구가 더 필요한 것으로 여겨진다.

참 고 문 헌

[1] C. Berrou, A. Glavieux, and P. Thitimajshma, "Near Shannon limit error- correction coding and decoding : Turbo- codes," *Proc. ICC*, pp. 1064-107
0, Geneva, Switzerland, May 1993.

[2] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. Inform. Theory*, vol. 42, pp. 409-428, Mar. 1996.

[3] P.Robertson, "Illuminating the structure of code and decoder of parallel concatenated recursive systematic (turbo) codes," *Proc. GLOBECOM*, pp. 1298-1303, San Francisco, U.S.A., Nov. 1994.

[4] P. Robertson, E. Villebrun, and P. Hoeher, "A comparison of optimal and sub-optimal MAP decoding algorithms operating in the log domain," *Proc. ICC*, pp. 1009-1013, Seattle, U.S.A., June 1995.

[5] V. Franz and J. B. Anderson, "Concatenated decoding with a reduced -search BCJR algorithm," *IEEE J. Select. Areas Commun.*, vol. 16, pp. 186-195, Feb. 1998.

[6] J. Hagenauer and P. Hoeher, "A Viterbi algorithm with soft-decision outputs and its applications," *Proc. GLOBECOM*, pp. 1680-

1686, Dallas, U.S.A., Nov. 1989.

[7] L. Papke and P. Robertson, "Improved decoding with the SOVA in a parallel concatenated (turbo-code) scheme," *Proc. ICC*, pp. 102-106, Dallas, U.S.A., June 1996.

[8] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "Soft-output decoding algorithms for continuous decoding of parallel concatenated convolutional codes," *Proc. ICC*, pp. 112-117, Dallas, U.S.A., June 1996.

[9] P. Jung, "Comparison of turbo-code decoders applied to short frame transmission system," *IEEE J. Select. Areas Commun.*, vol. 14, pp. 530-537, Apr. 1996.

[10] F. Hemmati and D. J. Costello, Jr., "Truncation error probability in Viterbi decoding," *IEEE Trans. Commun.*, vol. 25, pp. 530-532, May 1977.

[11] S. Lin and D. J. Costello, Jr., *Error Control Coding*, Prentice-Hall, 1983.

곽 지 혜(Ji-Hye Gwak) 정회원

1997년 2월 : 한국과학기술원 전기 및 전자공학과 학사과정 졸업

1999년 2월 : 한국과학기술원 전기 및 전자공학과 석사과정 졸업

1999년 3월~현재 : 한국전자통신 연구원 무선 방송 기술 연구소 위성통신 시스템 연구부

<주관심 분야> 전자공학, 통신공학

양 우 석(Woo-Seok Yang) 정회원

1998 2월 : 연세대학교 전자공학과 학사과정 졸업

1998 3월~현재 : 한국과학기술원 전기 및 전자공학과 석사 과정

<주관심 분야> 전자공학, 통신공학

김 형 명(Hyung-Myung Kim) 정회원

1974년 2월 : 서울대학교 공학사

1982년 4월 : 미국 Pittsburgh 대학 전기공학과 석사

1985년 12월 : 미국 Pittsburgh 대학 전기공학과 공학박사

1986년 4월~1992년 8월 : 한국과학기술원 전기 및 전자공학과 조교수

1992년 9월~현재 : 한국과학기술원 전기 및 전자공학과 부교수

<주관심 분야> 디지탈 신호와 영상처리 다차원 시스템 이론, 비디오신호 전송통신 이론, 이동통신 기술 분야