A simplified computational kernel for trellis-based decoding

Kamuf, Matthias; Anderson, John B; Öwall, Viktor

Published in:
IEEE Communications Letters

DOI:
10.1109/LCOMM.2004.825743

2004

Link to publication

Citation for published version (APA):

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.
A Simplified Computational Kernel for Trellis-Based Decoding

Matthias Kamuf, Student Member, IEEE, John B. Anderson, Fellow, IEEE, and Viktor Öwall, Member, IEEE

Abstract—A simplified branch metric and add-compare-select (ACS) unit is presented for use in trellis-based decoding architectures. The simplification is based on a complementary property of best feedforward and some systematic feedback encoders. As a result, one adder is saved in every other ACS unit. Furthermore, only half the branch metrics have to be calculated. It is shown that this simplification becomes especially beneficial for rate 1/2 convolutional codes. Consequently, area and power consumption will be reduced in a hardware implementation.

Index Terms—Add-compare-select (ACS), branch metric, logMAP, very large-scale integration (VLSI), Viterbi decoding.

I. INTRODUCTION

Trellis-based decoding is a popular method to recover convolutionally encoded information corrupted during transmission over a noisy channel. For example, the Viterbi algorithm (VA) [1] and the Bahl-Cocke-Jelinek-Raviv (BCJR) algorithm [2] are two schemes that work on an underlying trellis description of the encoded sequence.

Basic computations in either algorithm involve branch metric (BM) calculations and add-compare-select (ACS) operations. In case of the VA, an ACS operation successively discards branches that cannot be part of the survivor path. In case of the BCJR in the logarithmic domain (the logMAP algorithm [3]), this operation corresponds to an add-max* operation [4], which is basically an ACS operation with an added offset (ACSO) to correct for the Jacobian logarithm. Hence, the presented considerations for the ACS hold for the ACSO as well.

Almost all good rate 1/2 convolutional codes, \( n \) an integer, have the property that the code symbol labels on the two branches into each trellis node are complementary. However, simplifications to the BM and ACS units that result have not been published yet, to our knowledge. Consequently, we use this property and present a simplified architecture with reduced complexity for these units, thus saving hardware.

After having introduced a convenient notation in the following section, we address the simplification in Section III and the resulting implications for hardware realization in Section IV.

II. NOTATION

The ACS operation is best described by (1). Let \( \Gamma(s, k+1) \) be the updated metric of state \( s \) at time \( k+1 \), based on the preceding state metrics at time \( k \) and the respective branch metrics \( \lambda() \):

\[
\Gamma(s, k+1) = \min \{ \Gamma(s', k) + \lambda(s', s), \Gamma(s'', k) + \lambda(s'', s) \}
\]  

(1)

A channel symbol received from a soft-output demodulator is quantized with \( q \) bits and denoted \( y_k \). Clearly, there are \( 2^q \) quantization levels and \( y_k \in [0, 2^q - 1] \). This symbol is the output of a discrete memoryless channel with binary input \( x_j \) and transition probabilities \( P(y_k | x_j) \). The expected code symbol \( c_j(s', s) \) along the branch from state \( s' \) to state \( s \) is derived by the mapping \( x_0 \mapsto 0 \) and \( x_1 \mapsto 2^q - 1 \).

In the additive white Gaussian noise channel the optimal distance measure is the squared Euclidean distance

\[
\sum_{j=0}^{n-1} (y_k - c_j(s', s))^2.
\]  

(2)

However, given the preceding symbol constraints, it is shown in [5] that this measure simplifies to

\[
\lambda_j(s', s) = \begin{cases} 
  y_k, & \text{for } c_j(s', s) = 0 \\
  (2^q - 1 - y_k), & \text{for } c_j(s', s) = 2^q - 1
\end{cases}
\]  

(3)

and the complete branch metric is then written as

\[
\lambda(s', s) = \sum_{j=0}^{n-1} \lambda_j(s', s).
\]  

(4)

III. USING THE COMPLEMENTARY PROPERTY

This discussion is restricted to rate 1/2 codes, that is \( n = 2 \), although the considerations can be generalized to \( 1/n \) codes. Rate 1/2 codes play by far the most important role in today’s communication systems since they are a good compromise between achievable coding gain, bandwidth efficiency, and implementation complexity. In practice, high-rate codes are usually obtained by puncturing a basic rate 1/2 code. However, we begin with a general notation that shows that the most beneficial simplification results for \( n = 2 \).

We consider both feedforward encoders and some systematic feedback encoders. The best feedback encoders of memory \( m \) are defined by two shift register tap sets which are delayfree [6]. Some best systematic feedback encoders can be found in [7].

These encoders have one thing in common: The code symbols of merging branches are always complementary. In Fig. 1,
the complementary operation on \( c \) is defined as the complementation of its elements, that is \( \overline{c} = (\overline{c}_0 \ldots \overline{c}_{n-1}) \), where \( c_i + \overline{c}_i = 2^n - 1 \).

From the considerations in Section II it is clear that the branch metrics share this property since they linearly depend on the code symbols. Hence, one branch metric can be expressed by means of the other and we write

\[
\lambda(s''', s) = n(2^n - 1) - \lambda(s', s) = (n-1)\lambda(s', s) + n[(2^n - 1) - \lambda(s', s)].
\]

(W) 5

We define the modified branch metric

\[
\lambda^*(s', s) = n[(2^n - 1) - \lambda(s', s)]
\]

which is a signed number, and (5) becomes

\[
\lambda(s''', s) = (n-1)\lambda(s', s) + \lambda^*(s', s).
\]

Substituting (7) into (1) gives

\[
\Gamma(s, k+1) = \min \{ \Gamma(s', k) + \lambda(s', s), \Gamma(s'', k) + (n-1)\lambda(s', s) + \lambda^*(s', s) \};
\]

Finally, the factor \( \lambda(s', s) \) in the first argument of (8) can be taken out of the comparison and we get

\[
\Gamma(s, k+1) = \lambda(s', s) + \min \{ \Gamma(s', k), \Gamma(s'', k) + (n-2)\lambda(s', s) + \lambda^*(s', s) \},
\]

or, equivalently

\[
\Gamma(s, k+1) = \lambda(s', s) + \Gamma(s, k+1)
\]

where \( \Gamma(s, k+1) \) is the new outcome of the min operation.

There are several things to be observed in (9) and (10). First, considering that the branch metrics are precalculated, there is one addition less needed to carry out the comparison since the first argument in the comparison remains unchanged. Second, for \( n = 2 \) the factor \( \lambda(s', s) \) disappears in the second argument and the comparison solely depends on one (modified) branch metric. Third, in order to retain the numerical relation between interconnected state metrics with different \( \lambda() \) we have to add this factor after having determined \( \Gamma(s, k+1) \). However, one can subtract this factor from all state metrics and it will be shown that in that case half the ACS units do not need this correction, that is \( \Gamma(s, k+1) = \overline{\Gamma}(s, k+1) \). Note that if the butterflies in a trellis were disjoint, this correction could be neglected in all ACS units.

Fig. 1. Complementary property of merging branches.

IV. MODIFIED BM AND ACS UNITS FOR RATE 1/2

We start by noting that the branch metric \( \lambda(s', s) \) can take four different values, namely \( \lambda(x_0|x_1) \) for every possible combination of symbols \( x_j \in \{0,1\} \).

Fig. 2 shows both the conventional and the transformed ACS unit. Both units have the same complexity but the latter needs one adder less to determine the outcome of the comparison.

(a) (b) (c) (d) (e) (f) (g) (h) (i) (j) (k) (l) (m) (n) (o) (p) (q) (r) (s) (t) (u) (v) (w) (x) (y) (z)

\[
\lambda^*(s', s) = \lambda(s', s) - \lambda(00) = \lambda(10) - \lambda(00)
\]

Fig. 2. Conventional and (b) transformed ACS unit for a rate 1/2 code. Both units have the same complexity but the latter needs one adder less to determine the outcome of the comparison.

As stated in [5], if there are \( 2^n \) distinct code sequences, a conventional BM unit requires \( 2^n(n-1) \) additions and \( n \) negations to calculate \( 2^n \) branch metrics. Hence, for a rate 1/2 code

\[
\Delta \lambda = (10) - (00)
\]
we need four adders and two negations to calculate four branch metrics, see Fig. 5(a). The proposed BM unit shown in Fig. 5(b) requires only three additions, one negation of a channel symbol, and two negations of intermediate branch metrics to calculate two branch metrics. Notice that a bit-shift operation comes at negligible cost in a hardware implementation. Furthermore, the difference between the two branch metrics, $\Delta \lambda$, needed to normalize half the state metrics becomes in this case simply $\lambda(00) - \lambda(10)$. This operation can be further simplified on the bit level into a bit-shift followed by a negation (MSB excluded) of $y_0$ and is hence not considered an adder in Table I.

This table shows the number of additions for a BM/ACS unit setup for code rate $1/2$ and memory $m$. The proposed scheme halves the additions in the BM unit and reduces the number of additions for the ACS units by 17%. By software simulation of the hardware circuits, we have verified that decoder error performance stays the same.

V. CONCLUSION

We have shown that the implementation of BM and ACS units in trellis-based decoding architectures can be simplified for a certain class of convolutional codes. For a rate $1/2$ code, half the ACS units save one adder compared to a conventional implementation. Furthermore, only two branch metrics have to be calculated instead of four. These potential hardware savings will also lead to savings in power consumption.

REFERENCES