Berlekamp–Massey algorithm

From Wikipedia, the free encyclopedia

The **Berlekamp–Massey algorithm** is an algorithm that will find the shortest linear feedback shift register (LFSR) for a given binary output sequence. The algorithm will also find the minimal polynomial of a linearly recurrent sequence in an arbitrary field. The field requirement means that the Berlekamp–Massey algorithm requires all non-zero elements to have a multiplicative inverse.[1] Reeds and Sloane offer an extension to handle a ring.[2]

Elwyn Berlekamp invented an algorithm for decoding Bose–Chaudhuri–Hocquenghem (BCH) codes.[3][4] James Massey recognized its application to linear feedback shift registers and simplified the algorithm.[5][6] Massey termed the algorithm the LFSR Synthesis Algorithm (Berlekamp Iterative Algorithm),[7] but it is now known as the Berlekamp–Massey algorithm.

Contents

Description of algorithm

The Berlekamp–Massey algorithm is an alternate method to solve the set of linear equations described in Reed–Solomon Peterson decoder, which can be summarized as:

$$S_{i+\nu} + \Lambda_1 S_{i+\nu-1} + \cdots + \Lambda_{\nu-1} S_{i+1} + \Lambda_\nu S_i = 0.$$

In the code examples below, C(x) is a potential instance of Λ(x). The error locator polynomial C(x) for *L* errors is defined as:

$$C(x) = C_L\ x^L + C_{L-1}\ x^{L-1} + \cdots + C_2\ x^2 + C_1\ x + 1$$

or reversed:

$$C(x) = 1 + C_1\ x + C_2\ x^2 + \cdots + C_{L-1}\ x^{L-1} + C_L\ x^L.$$

The goal of the algorithm is to determine the minimal degree *L* and C(x) which results in:

$$S_n + C_1\ S_{n-1} + \cdots + C_L\ S_{n-L} = 0$$

for all syndromes, $n = L$ to (N-1).

Algorithm: C(x) is initialized to 1, *L* is the current number of assumed errors, and initialized to zero. *N* is the total number of syndromes. *n* is used as the main iterator and to index the syndromes from 0 to (N-1). B(x) is a copy of the last C(x) since *L* was updated and initialized to 1. *b* is a copy of the last discrepancy *d* (explained below) since *L* was updated and initialized to 1. *m* is the number of iterations since *L*, B(x), and *b* were updated and initialized to 1.

Each iteration of the algorithm calculates a discrepancy *d*. At iteration *k* this would be:

$$d = S_k + C_1\ S_{k-1} + \cdots + C_L\ S_{k-L}.$$

If *d* is zero, the algorithm assumes that C(x) and L are correct for the moment, increments *m*, and continues.

If *d* is not zero, the algorithm adjusts C(x) so that a recalculation of *d* would be zero:

$$C(x) = C(x)\ -\ (d/b)\ x^m\ B(x).$$

The x^m term *shifts* B(x) so it follows the syndromes corresponding to 'b'. If the previous update of L occurred on iteration *j*, then $m = k - j$, and a recalculated discrepancy would be:

$$d = S_k + C_1\ S_{k-1} + \cdots - (d/b)(S_j + B_1\ S_{j-1} + \cdots).$$

This would change a recalculated discrepancy to:

$$d = d - (d/b)b = d - d = 0.$$

The algorithm also needs to increase *L* (number of errors) as needed. If *L* equals the actual number of errors, then during the iteration process, the discrepancies will become zero before *n* becomes greater than or equal to (2 *L*). Otherwise L is updated and algorithm will update B(x), b, increase L, and reset m = 1. The formula L = (n + 1 - L) limits L to the number of available syndromes used to calculate discrepancies, and also handles the case where L increases by more than 1.

Code sample

The algorithm from Massey (1969, p. 124).

```
polynomial(field K) s(x) = ... /* coeffs are s_j; output sequence as N-1 degree polynomial) */
/* connection polynomial */
polynomial(field K) C(x) = 1;  /* coeffs are c_j */
polynomial(field K) B(x) = 1;
int L = 0;
int m = 1;
field K b = 1;
int n;

/* steps 2. and 6. */
for (n = 0; n < N; n++)
  {
    /* step 2. calculate discrepancy */
    field K d = s_n + \Sigma_{i=1}^L c_i * s_{n-i};

    if (d == 0)
      {
        /* step 3. discrepancy is zero; annihilation continues */
        m = m + 1;
      }
    else if (2 * L <= n)
      {
        /* step 5. */
        /* temporary copy of C(x) */
        polynomial(field K) T(x) = C(x);

        C(x) = C(x) - d b^{-1} x^m B(x);
        L = n + 1 - L;
        B(x) = T(x);
        b = d;
        m = 1;
      }
    else
      {
        /* step 4. */
        C(x) = C(x) - d b^{-1} x^m B(x);
        m = m + 1;
      }
  }
return L;
```

The algorithm for the binary field

The following is the Berlekamp–Massey algorithm specialized for the typical binary finite field F_2 and GF(2). The field elements are 0 and 1. The field operations + and − are identical and become the exclusive or operation, XOR. The multiplication operator * becomes the logical AND operation. The division operator reduces to the identity operation (i.e., field division is only defined for dividing by 1, and x/1 = x).

1. Let $s_0, s_1, s_2 \cdots s_{n-1}$ be the bits of the stream.
2. Initialise two arrays b and c each of length n to be zeroes, except $b_0 \leftarrow 1, c_0 \leftarrow 1$
3. assign $L \leftarrow 0, m \leftarrow -1$.
4. **For** $N = 0$ **step** 1 **while** $N < n$:
   - Let d be $s_N + c_1 s_{N-1} + c_2 s_{N-2} + \cdots + c_L s_{N-L}$.
   - **if** $d = 0$, **then** c is already a polynomial which annihilates the portion of the stream from $N - L$ to N.
   - **else**:
     - Let t be a copy of c.
     - Set $c_{N-m} \leftarrow c_{N-m} \oplus b_0, c_{N-m+1} \leftarrow c_{N-m+1} \oplus b_1, \ldots$ up to $c_{n-1} \leftarrow c_{n-1} \oplus b_{n-N+m-1}$ (where $\oplus$ is the Exclusive or operator).
     - If $L \leq \frac{N}{2}$, set $L \leftarrow N + 1 - L$, set $m \leftarrow N$, and let $b \leftarrow t$; otherwise leave L, m and b alone.

At the end of the algorithm, L is the length of the minimal LFSR for the stream, and we have $c_L s_a + c_{L-1} s_{a+1} + c_{L-2} s_{a+2} + \cdots = 0$ for all a.

Code sample for the binary field in Java

The following code sample is for a binary field.

```
    public static int runTest(int[] array) {
        final int N = array.length;
        int[] b = new int[N];
        int[] c = new int[N];
        int[] t = new int[N];
        b[0] = 1;
        c[0] = 1;
        int l = 0;
        int m = -1;
        for (int n = 0; n < N; n++) {
            int d = 0;
            for (int i = 0; i <= l; i++) {
                d ^= c[i] * array[n - i];
            }
            if (d == 1) {
                System.arraycopy(c, 0, t, 0, N);
```

```
            int N_M = n - m;
            for (int j = 0; j < N - N_M; j++) {
                c[N_M + j] ^= b[j];
            }
            if (l <= n / 2) {
                l = n + 1 - l;
                m = n;
                System.arraycopy(t, 0, b, 0, N);
            }
        }
    }
    return l;
}
```

See also

- Reeds–Sloane algorithm, an extension for sequences over integers mod *n*
- Berlekamp–Welch algorithm
- NLFSR, Non-Linear Feedback Shift Register

References

1. Reeds & Sloane 1985, p. 2
2. Reeds, J. A.; Sloane, N. J. A. (1985), "Shift-Register Synthesis (Modulo n)" (PDF), *SIAM Journal on Computing* **14** (3): 505–513, doi:10.1137/0214038
3. Berlekamp, Elwyn R. (1967), *Nonbinary BCH decoding*, International Symposium on Information Theory, San Remo, Italy
4. Berlekamp, Elwyn R. (1984) [1968], *Algebraic Coding Theory* (Revised ed.), Laguna Hills, CA: Aegean Park Press, ISBN 0-89412-063-8. Previous publisher McGraw-Hill, New York, NY.
5. Massey, J. L. (1969), "Shift-register synthesis and BCH decoding" (PDF), *IEEE Trans. Information Theory*, IT-15 (1): 122–127
6. Ben Atti, Nadia; Diaz-Toca, Gema M.; Lombardi, Henri, *The Berlekamp–Massey Algorithm revisited*, CiteSeerX: 10.1.1.96.2743
7. Massey 1969, p. 124

External links

- Hazewinkel, Michiel, ed. (2001), "Berlekamp-Massey algorithm", *Encyclopedia of Mathematics*, Springer, ISBN 978-1-55608-010-4
- Berlekamp–Massey algorithm (http://planetmath.org/encyclopedia/BerlekampMasseyAlgorithm.html) at PlanetMath.
- Weisstein, Eric W., "Berlekamp–Massey Algorithm" (http://mathworld.wolfram.com/Berlekamp-MasseyAlgorithm.html), *MathWorld*.
- GF(2) implementation in Mathematica (http://code.google.com/p/lfsr/)
- **(German)** Applet Berlekamp–Massey algorithm (http://www.informationsuebertragung.ch/indexAlgorithmen.html)
- Online GF(2) Berlekamp-Massey calculator (http://berlekamp-massey-algorithm.appspot.com/)

Retrieved from "https://en.wikipedia.org/w/index.php?title=Berlekamp–Massey_algorithm&oldid=696366845"

Categories: Error detection and correction | Cryptanalytic algorithms