Algorithms that changed the world coursework: Reed-Solomon codes

710050289

To combat burst errors such as scratches on CD's, Reed-Solomon codes are used. An encoding method which generates a unique polynomial based on the data someone wishes to encode to add parity characters to the message enabling a receiver to identify and correct errors. Reed-Solomon codes work using individual characters instead of bits. This means that in the best case, for a message of length 223 bits, using 32 parity bits, it can protect up to 16 bit flips along 2 bytes, or 2 bit flips along 2 bytes in the worst case. The encoding of a message into a Reed-Solomon code is very quick, but the decoding is quite slow, with typical performance around $O(n^3)$ for the Berlekamp-Welch algorithm. Consequently, these are typically used for non-real time applications such as CD's or QR codes which are prone to burst errors.

Word count: 1497

I certify that all material in this report which is not my own work has been identified.

Introduction

In 1960, Irving S. Reed and Gustave Solomon came up with an algorithm to create erasure error-correcting codes, so called Reed-Solomon codes (Reed & Solomon, 1960). Error-correcting codes were first introduced in the 1950's with Hamming codes, when Hamming was frustrated with error-prone punch card readers. Hamming codes worked using parity bits, specifically 5 for a 16-bit message, and were able to detect two-bit errors and correct one-bit errors. For larger blocks, such as 256, or 128 this became less useful, as Hamming codes could still only detect 2 errors or correct one (Winsor, 2023), and the probability that more errors occurred was higher. Furthermore, errors usually came in bursts, either from noise during transmission, or a scratch on a CD, which knocked multiple bits. To combat burst errors and the need for a dynamic amount of error correction, Reed-Solomon codes can be used.

Reed-Solomon codes encode a message by determining a unique polynomial based on the message. This works due to the Lagrange interpolation theorem, which states that only a single polynomial of order k-1, which goes through k specified point exists. The message is used as the k specified points and the polynomial is then used to calculate parity characters. Reed-Solomon codes are very robust against burst errors as characters are considered to generate the polynomial rather than bits. This means that 8 bit flips or 1 bit flip under a single byte character have the same resulting number of errors. Furthermore, the error correction capabilities of a Reed-Solomon code can be adjusted based on preference by just adding more parity characters, with no theoretical upper bound apart from increased computation time. Finally, Reed-Solomon codes are an erasure error-correcting code, this means t erasures in data can be corrected when 2t parity characters are used in the Reed-Solomon encoded message as long as the position of the erasures are known. These properties make Reed-Solomon codes well suited for error correction in CD's or DVDs, where burst errors from scratches are common. Or in data transmission including satellite communications such as DVB or ATSC, as well as storage systems such as RAID 6 where retaining or transmitting the correct data is paramount, as Reed-Solomon can correct a large number of data erasures.

Encoding

Reed-Solomon codes work by creating a unique polynomial p based on the data desired to be stored or sent. In the original view, for a message of length k, a polynomial of order k-1 is created by representing the data as the coefficients of the polynomial p. The encoded message C of length n is created by considering each character as the result of the polynomial p for ascending x values from 0 to n-1, $C = [(p(0), \ldots, p(n-1)]$. The number of parity characters is determined by n-k, where the first k characters in the encoded message is the original message, and the following n-k characters are the parity characters added for error correction.

```
Algorithm 1: Reed-Solomon original view encoder
  Let $n$ be the encoded message length, and $k$ the original message length, where $n \geq k$ is
    satisfied;
  Let $m = (m_0, \ldots, m_{k-1})$ be the message to be encoded;
  Let $P_m$ be the polynomial made by using $m$ as coefficients, formally:
    $P_m(x) = \sum_{i=0}^{k-1} m_i x^i$;
  Let $C$ be an array of length $n$;
  for $x$ = $0$ to $n-1$ do
  |   $C[x] = P_m(x)$
  end
  Return $C$;
  Result: C - The encoded message
```

As we can see in Algorithm 1, the above procedure is described. It's important to note this is done under a Galois field. Galois field's work under modulo arithmetic within a finite bound, where the upper bound is denoted by a prime power, typically for Reed-Solomon codes this is GF(2^8) which allows encoding of a message up to 256 bits. This value is used as it represents the bounds of the alphabet of the Reed-Solomon code, in other words the range of values used by ASCII codes, this ensures every possible character used in a ASCII message has a unique value.

The Reed-Solomon original view encoder is very fast. The polynomial is defined by the message k, and hence takes k calculations. Encoding the message uses a loop repeated n times, hence under the requirement that $n \geq k$, the time complexity is $O(n)$. Spatially the encoding algorithm is very compact, only needing to allocate space for a single array C, and temporarily assigning heap space for one array representing polynomial P_m.

Its worth noting that the encoded message C loses the original message, formally it's not a systematic code. Soon after the original proposal using the message m as the coefficient of the polynomial P_m, a new encoding procedure was suggested which used Lagrange interpolation to create a systematic code.

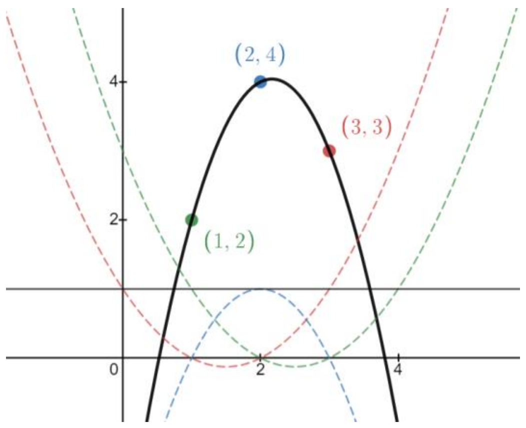

Figure 1: Lagrange interpolation for points (1,2) (2,4) (3,3)

As we can see in Figure 1 Lagrange interpolation uses Lagrange polynomials to interpolate the polynomial of order k-1 for k points. A Lagrange polynomial for n points (x_n, y_n) including unit point u (x_u, y_u) is calculated by creating a polynomial of order n-1, with its x intercepts being the values x_n from the n points, and the unit point u have a value of 1 at x_u. In figure 1 the Lagrange polynomial for each point is shown in its colour, with a horizontal black line representing y = 1 to show the unit points. The black polynomial is the Lagrange interpolating polynomial (LIP) obtained by multiplying each Lagrange polynomial over n points by the y value of its respective unit point, and then summing them together: $l = l_1y_1 + l_2y_2 + l_3y_3$ for 3 points.

For Reed-Solomon codes this has the property that a message m of length k can be interpreted as a list of points, with their x value being the position in the message (0,…,k-1) and their y value being the ASCII number represented by their character. Hence the encoding polynomial is found by finding the LIP along the m points.

Algorithm 2: Reed-Solomon systematic encoder

```
Let $n$ be the encoded message length, and $k$ the original message length, where $n \geq k$ is
  satisfied;
Let $m = (m_0, ..., m_{k-1})$ be the message to be encoded;
Compute the polynomial $P_m$ of order $k-1$; Computed through lagrange interpolation,
  where $P_m(x_i) = m_i$ for all $i \in 0, ..., k-1$;
Let $C$ be an array of length $n$;
for $x$ = $0$ to $n-1$ do
    $C[x] = P_m(x)$
end
Return $C$;
Result: C - The encoded message
```

Algorithm 2 describes the above-described procedure. The rest of the procedure remains the same as in Algorithm 1. The benefit of this procedure is that the transmitted message is systematic, so the encoded message contains the original message. There is overhead in calculating the LIP, but it remains in time complexity is $O(n)$. For a message of length k, k Lagrange polynomials need to be calculated, k multiplications and k-1 additions are used to find the LIP, and then a loop running n-k times finds the parity characters for a encoded message of length n.

It's important to mention Lagrange interpolation can be used in a Galois field which makes this method possible for Reed-Solomon codes. Furthermore, in modern uses of Reed-Solomon codes, Vandermonde matrices are used instead of Lagrange interpolation. Vandermonde matrices function as the generator matrix for the encoding and can be used to find the interpolating polynomial allowing the same systematic code to be generated as in Lagrange interpolation. This method is slightly more efficient on modern computer chips as the codeword C can be found through $C = m * A$, where A is the Vandermonde matrix of size $n * k$, where n is the encoded message length and k is the message length. This is fast as it computes as a matrix-vector multiplication which is highly optimized by modern chips.

Decoding

The original view Reed-Solomon decoder worked by finding the most popular message polynomial. From the Lagrange interpolation theorem, we know if no errors are present in the encoded message C, each message subset k will result in the same LIP P. The decoder works by calculating the LIP for each message subset in the encoded message, counting how many times each LIP is calculated. The LIP with the highest frequency is deemed the correct one and the original message is then reconstructed from this LIP.

Algorithm 3: Reed-Solomon theoretical systematic decoder

```
Let $n$ be the encoded message length, and $k$ the original message length, where $n \geq k$ is
  satisfied;
Let $C = (C_0, ..., C_{n-1})$ be the encoded message;
Compute possibleMessages to be an array of size $\binom{n}{k}$, which holds every message
  combination of size $k$ from the encoded message $C$;
Let $D$ be a dictionary containing key-value pairs ($P_m$, count);
for $m$ in possibleMessages do
    Compute the polynomial $P_m$ of order $k-1$; Computed through lagrange interpolation,
      where $P_m(x_i) = m_i$ for all $i \in 0, ..., k-1$;
    $D[P_m]$ += 1;
end
Let $P_m = D[P_m$, max value];
Let $m$ be an array of length $k$;
for $x$ = 0 to $k-1$ do
    $m[x] = P_m(x)$
end
Return $m$;
Result: $m$ - The original message
```

Algorithm 3 describes the above-described procedure. It's important to note that using this decoder, to correctly correct a corrupted message, there needs to be more parity characters than errors, as the number of correct LIP calculated must be greater than the number of incorrect LIP's. This is quite a lengthy procedure, with time complexity $O(C(n, k))$. This is because the loop calculating the LIP's will be run n choose k times for a message length k and code length n. This can become a very large number for high n and k values. Spatial complexity is also not great as it needs n choose k arrays in the worst case to hold the LIP's. Furthermore the theoretical decoder must be run in order to get the original message, in more popular decoding algorithms include the Berlekamp-Welch algorithm, which require knowledge of a generator matrix used by the Vandermonde matrix allow it to check for errors before decoding, enabling a faster procedure in the best case.

Conclusion

The theoretical Reed-Solomon encoder and decoder enable messages to be encoded and protected against burst errors very well. Furthermore, they are resilient to many errors which can be decided based on the application. Use of Galois fields is imperative to achieve desired performance as large polynomials can generate huge values. For large messages the theoretical decoders loses practicality over its large time and spatial complexity, this is why in modern uses of Reed-Solomon codes, different decoding algorithms such as the Berlekamp-Welch algorithm are used. Use of Vandermonde matrices make encoding very fast, this is usually done through hardware well-optimised for matrix-vector multiplication. Furthermore, in modern uses generator matrices are used to allow more complex decoding algorithms to determine whether errors are present, the location of errors, or the number of errors before decoding. This also enables Reed-Solomon codes to combat erasures in data when the position data of the message is also lost.

References

Reed, I. S., & Solomon, G. (1960). Polynomial Codes Over Certain Finite Fields. *Journal of the Society for Industrial and Applied Mathematics*, 300-304.

Winsor, P. (2023, 12 4). *Error detecting and correcting codes.* Retrieved from EECS 373: Design of Microprocessor Based Systems: https://www.eecs.umich.edu/courses/eecs373.w05/lecture/errorcode.html