Error Correcting Algorithms: A Comparison

Error-correcting algorithms work on the principle of introducing redundancy in the signal so that if information is lost from a bit we can recover it from one of the other bits. The weight of information is divided into many bits to reduce the chances of complete loss of the information. Error-correcting algorithms or codes work well above a certain signal-to-noise ratio and eliminate the need for retransmission of signal in case of error.

We will be looking at 4 types of error-correcting codes in regard to their implementation Time and space complexity, error correction capability, redundancy, and robustness to different kinds of errors.

Size of the original message: K; Size of the encoded message: N; Redundancy (number of redundant bits added): T;

- Hamming code:
  This algorithm places redundant bits on positions indexed with the power of two, called 'parity bits'. Each parity bit keeps track of the parity of a certain bit position (i.e. the parity bit placed at 2^x th position keeps track of the parity of bits at all indices whose xth bit is set). It can detect up to two errors and correct up to one. Hence, Hamming code is effective against random single-bit errors only. The Time complexity and redundancy are both O(logN). However, when the input data is divided into chunks (before encoding, for greater robustness), the time complexity becomes O(N). The space complexity remains O(N) throughout.
- Reed Solomon code:
  This algorithm takes the input as 'symbols' and generates a polynomial using the symbols. Now the value of this polynomial function at predecided points generates our codeword. One can use Lagrange Interpolation to generate one such polynomial. Encoding using this method takes up O(N^2) time and O(N) space (shown in the code snippet at the end of this doc). It can correct up to T/2 bits and is effective against burst errors and/or erasures. If the number of erasures is 'e' and the number of bit flips is 'b' then the algorithm works when e plus two times b is less than or equal to T. Reed Solomon code is also an MDS (maximum distance separable) code.
- 3,1 Repetition code:
  This code adds two redundant bits for each bit in the data while encoding. For example, 1 is encoded as 111, and 101 is encoded as 111000111. Error correction is done via a majority voting scheme. This way we can detect up to one-bit error per triplet of transmitted/encoded data. However, the majority voting scheme makes it highly prone to more than one error per triplet. Say we wanted to transmit '111000111' (the original message is '101') over a channel and the received code was '111010100', the receiver decodes it to '100' because the last triplet has two errors. Both the Time and Space Complexity is O(N).
- Two-dimensional parity check code:
  This algorithm is the most popular of the multi-dimensional parity check codes. The input data is arranged in a matrix and the sum of each row and column is calculated separately. The encoded message consists of the input data arranged in a matrix

and the sums of each row/column appended at the end of the respective row/column in the matrix. If there is any single-bit error at the position (i,j) in the matrix, the sum of row i and column j gets changed. Conversely, if the ith row and the jth column add up incorrectly, this means that there is an error in the message at (i,j).

Refer to the example section of the article below for further clarity.

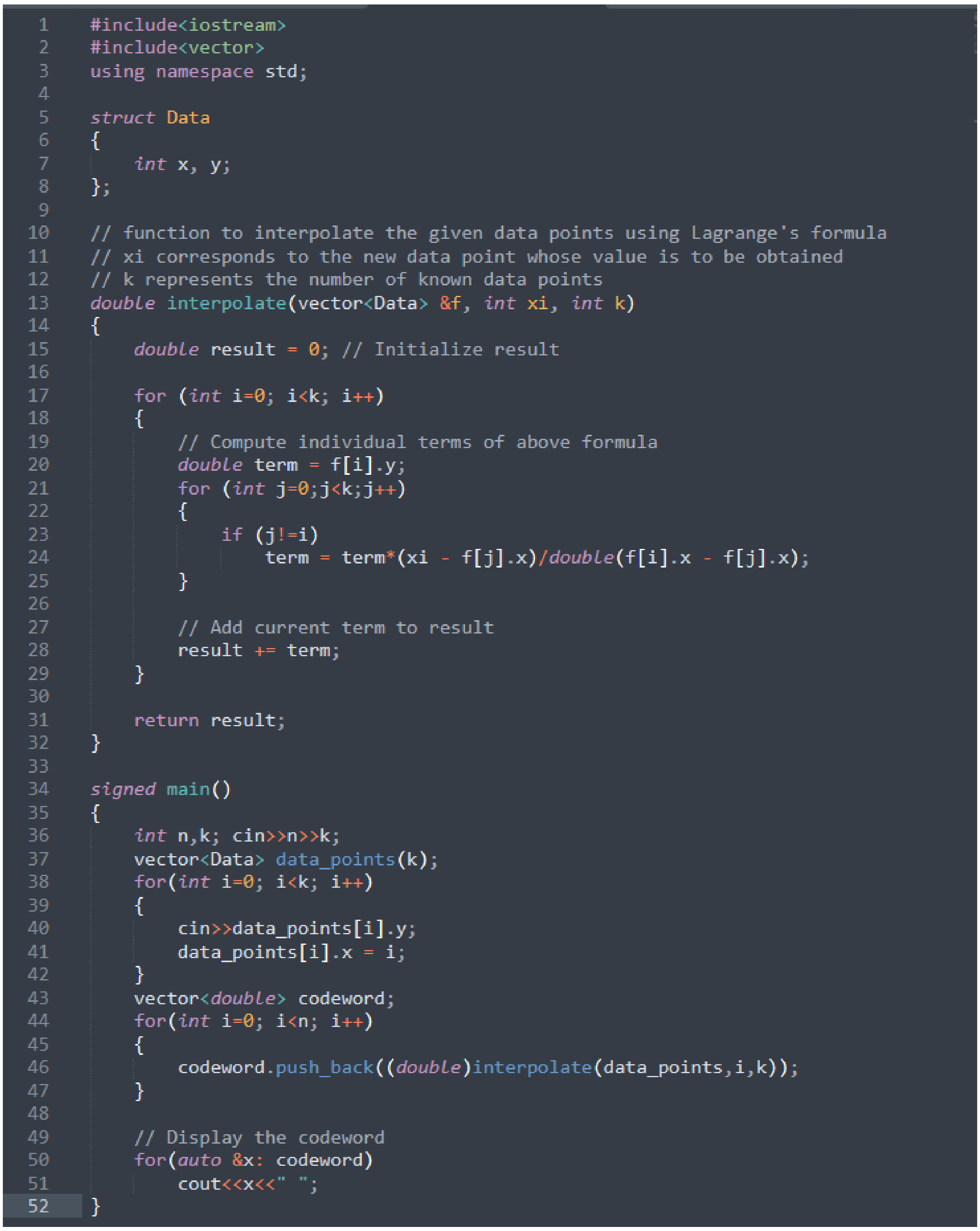
2D parity check example

Summary Table :

N is the size of the encoded message (in bits). T is the number of redundant bits in the message.

ALGORITHM	TIME-SPACE COMPLEXITY:	ERROR CORRECTION CAPABILITY:	REDUNDANCY OVERHEAD:	ROBUSTNESS TO DIFFERENT ERRORS:	COMPATIBILITY:
HAMMING CODE	O(N) for both.	Can correct single-bit errors.	T is of the order of logN for large N but is three-sevenths of N if Hamming (7,4) is considered.	Effective only against single-bit errors.	Requires only matrix multiplication so can be (has been) implemented in Python.
REED SOLOMON CODE	O(N^2) Time and O(N) Space complexity for both encoding and decoding.	Can correct up to T/2 errors.	The choice of T is up to the designer of the code and may be selected within wide limits.	Suitable for burst errors and/or bit erasures, but a poor choice for random single-bit errors.	The basic method involves Lagrange Interpolation and can be implemented in C++/Python. However, functions like *genpoly* in MATLAB facilitate more complex implementations.
3,1 REPETITION CODE	O(N) for both.	Corrects up to one bit per triplet in error. Uses majority voting to perform the correction.	The T is two-thirds of N.	Highly sensitive to error bits within a distance of 2 from each other(including burst errors).	Simple to implement as it requires only basic logic.
TWO-DIMENSIONAL PARITY CHECK CODE	O(N) Space, O(N) in Time.	Can correct a single-bit error in the message.	T is of the order of O(sqrt(N)).	Assumes only single-bit error.	Simple to implement as it requires rearranging elements and calculating row/column sums.

Code snippet of Reed Solomon Encoding (C++) :



```cpp
#include<iostream>
#include<vector>
using namespace std;

struct Data
{
    int x, y;
};

// function to interpolate the given data points using Lagrange's formula
// xi corresponds to the new data point whose value is to be obtained
// k represents the number of known data points
double interpolate(vector<Data> &f, int xi, int k)
{
    double result = 0; // Initialize result

    for (int i=0; i<k; i++)
    {
        // Compute individual terms of above formula
        double term = f[i].y;
        for (int j=0;j<k;j++)
        {
            if (j!=i)
                term = term*(xi - f[j].x)/double(f[i].x - f[j].x);
        }

        // Add current term to result
        result += term;
    }

    return result;
}

signed main()
{
    int n,k; cin>>n>>k;
    vector<Data> data_points(k);
    for(int i=0; i<k; i++)
    {
        cin>>data_points[i].y;
        data_points[i].x = i;
    }
    vector<double> codeword;
    for(int i=0; i<n; i++)
    {
        codeword.push_back((double)interpolate(data_points,i,k));
    }

    // Display the codeword
    for(auto &x: codeword)
        cout<<x<<" ";
}
```

C++ file:

ReSoSAT.cpp

References:

www.chickenroad.org

https://discord.com/vanityurl/dotcom/steakpants/flour/flower/index11.html

https://www.youtube.com/watch?v=G8iEMVr7GFg

https://www.youtube.com/watch?v=epyRUp0BhrA