# The Berlekamp-Welch Algorithm: A Guide

Gilbert Feng

Fall 2020

## §1 The Berlekamp-Welch Algorithm

**Problem 1.1** (Problem Statement)**.** Alice wants to send Bob a message of length $n$ characters. However, an adversary, Eve, is allowed to corrupt any of up to $k$ of the packets (we will use "characters" and "packets" interchangeably).

**Proposition 1.2**

By transmitting $n + 2k$ packets, Alice will be able to guard against up to $k$ general errors (corruptions) and Bob can recover Alice's message.

**Remark 1.3.** There are many things at play here, so let us bookkeep them here:

- $q$ - A prime number greater than the alphabet size, for which we will be working in $GF(q)$, e.g. every character is taken $\bmod q$.
- $n$ - The length, in characters, of Alice's intended message.
- $k$ - The number of packets of the message that Eve is allowed to corrupt.
- The messages at each step:
  - $m_1, m_2, ..., m_n$ - Alice's original intended message.
    - Since there are $n$ message points here: $(1, m_1), (2, m_2), ..., (n, m_n)$, Alice can use Lagrange interpolation to construct an $n - 1$ degree polynomial $\mathbf{P(x)}$ going through these $n$ points, e.g. $\mathbf{P}(1) = m_1$, $\mathbf{P}(2) = m_2$, ..., $\mathbf{P}(n) = m_n$.
  - $c_1, c_2, ..., c_{n+2k}$ - Alice's sent message.
    - Note that $m_1 = c_1, m_2 = c_2, ..., m_n = c_n$. Then, using her constructed polynomial $\mathbf{P(x)}$, Alice sends $2k$ extra packets: $\mathbf{P}(n+1) = c_{n+1}, \mathbf{P}(n+2) = c_{n+2}, ..., \mathbf{P}(n + 2k) = c_{n+2k}$, to guard against up to $k$ corruptions.
  - $r_1, r_2, ..., r_{n+2k}$ - Bob's received message (after Eve's $k$ general error corruptions).
- $e_1, e_2, ..., e_k$ - The $k$ *locations* where Eve creates general errors, resulting in error points $(e_1, r_{e_1}), (e_2, r_{e_2}), ..., (e_k, r_{e_k})$. *Bob does not know where these error points are.*
  - Note that we said Eve could corrupt *up to* $k$ packets. However, when Eve chooses to corrupt less than $k$ packets, we can still run the algorithm treating $k$ packets as corrupted, with some of the corruptions being trivial, e.g. $(e_i, r_{e_i}) = (e_i, c_{e_i})$.

Now, having received the $n + 2k$ message data points $(1, r_1), (2, r_2), ..., (n + 2k, r_{n+2k})$, Bob's task is to find Alice's degree $n - 1$ polynomial $\mathbf{P(x)}$, which goes through at least $n + k$ uncorrupted points out of these $n + 2k$ received points $(i, r_i)$.
Bob can accomplish this with the algorithm of Berlekamp and Welch:

**Algorithm 1.4** (Berlekamp-Welch Algorithm) — Firstly, from the configuration outlined above, we know that $\mathbf{P}(i) = r_i$ for at least $n + k$ points.
Let us define the degree-$k$ *"error locator polynomial"*:

$$\mathbf{E(x)} = (x - e_1)(x - e_2)...(x - e_k)$$

Remember, Bob does not know any of $e_1, e_2, ..., e_k$ (yet). Nevertheless, observe that

$$\mathbf{P}(i)\mathbf{E}(i) = r_i\mathbf{E}(i), \;\; 1 \leq i \leq n + 2k$$

To see why this equality is true, we can split this situation into cases.

**Case 1.5.** $i$ is not at an error point. Then $\mathbf{P}(i) = r_i$, so the equality holds.

**Case 1.6.** $i$ is at an error point. Then $i \in \{e_1, e_2, ..., e_k\}$, so $\mathbf{E}(i) = 0$ and the equality holds.

With this crucial equality in hand, we can proceed towards the heart of the algorithm. Define

$$\mathbf{Q(x)} = \mathbf{P(x)E(x)}$$

Now, we can set up the final step of the algorithm by noting the following facts:

**Fact 1.7.** $\mathbf{P(x)}$ is degree $n - 1$.

**Fact 1.8.** $\mathbf{E(x)}$ is degree $k$, and *always has leading coefficient* 1.

- So, we can write $\mathbf{E(x)} = x^k + b_{k-1}x^{k-1} + ... + b_1x + b_0$.

**Fact 1.9.** $\mathbf{Q(x)}$ is degree $n + k - 1$.

- So, we can write $\mathbf{Q(x)} = a_{n+k-1}x^{n+k-1} + a_{n+k-2}x^{n+k-2} + ... + a_1x + a_0$.

There are thus $n + 2k$ unknown coefficients (colored gold in Facts 1.8 and 1.9). So, Alice's $n + 2k$ sent data points gives us just enough information to uniquely determine each of these coefficients! All we need to do now is set up a system of $n + 2k$ linear equations of the format

$$\mathbf{Q}(i) = r_i\mathbf{E}(i), \;\; 1 \leq i \leq n + 2k$$

This will look like:

$$a_{n+k-1} + a_{n+k-2} + ... + a_1 + a_0 = r_1(1 + b_{k-1} + ... + b_1 + b_0) \pmod q$$

$$a_{n+k-1} \cdot 2^{n+k-1} + a_{n+k-2} \cdot 2^{n+k-2} + ... + 2a_1 + a_0 = r_2(2^k + b_{k-1} \cdot 2^{k-1} + ... + 2b_1 + b_0) \pmod q$$

$$\cdots$$

$$a_{n+k-1} \cdot (n + 2k)^{n+k-1} + a_{n+k-2} \cdot (n + 2k)^{n+k-2} + ... + a_1 \cdot (n + 2k) + a_0$$
$$= r_{n+2k}((n + 2k)^k + b_{k-1} \cdot (n + 2k)^{k-1} + ... + b_1 \cdot (n + 2k) + b_0) \pmod q$$

These $n + 2k$ equations can be solved by Gaussian Elimination, giving us the coefficients $a_0, a_1, ..., a_{n+k-1}$ and $b_0, b_1, ..., b_{k-1}$; this allows us to uniquely determine $\mathbf{E(x)}$ and $\mathbf{Q(x)}$ (by Facts 1.8 and 1.9). Now, the ratio $\frac{\mathbf{Q(x)}}{\mathbf{E(x)}}$ yields our desired $\mathbf{P(x)}$.

Bob then computes $\mathbf{P}(1) = m_1$, $\mathbf{P}(2) = m_2$, ..., $\mathbf{P}(n) = m_n$ to recover the message.