List Decoding of Reed-Solomon Codes

Laure Fouard
Department of Computer Science
University of Calgary, Alberta

***Abstract*—In cryptography, the data in often transmited through a noisy channel. Hence one needs to have error-correcting codes, in order to recover from the possible errors. Moreover, in the case of large error rates, it is better to use list-decoding. In this paper we focus on the Reed-Salomon error-correcting code and on its list-decoding technic. We will explain deeply how the algorithms work, in order to be able to use them to solve other hard problems.**

***Index Terms*—Cryptography, error-correcting codes, list decoding.**

I. Introduction

Nowadays, communication is one of the most important issue in computer science and it includes a wide range of research areas, each of them aiming at solving different problems.
Coding theory deals with encoding and decoding of information. The encoding process is required to be efficient (in terms of time and space), but also to satisfy some properties which allow the decoding to be efficient as well. The decoding question follows from the fact that when a message is transmitted from a sender to a receiver, most of the time, it is done through a noisy channel. This results in the loss of parts of information which could be symbols erasure or errors occurring in the transmission. The receiver will try to recover the initial message by correcting errors and retrieving erased symbols.
Error-correcting codes allow the receiver to detect and correct errors (within some bounds) in the received message. Usually, errors are corrected by looking for the closest codeword from the received one. But, this process is not sufficient for highly noisy channels where the closest codeword is not always the transmitted one. That is why, among those codes, list-decoding has been widely studied because it allows error-correcting for a potentially large number of errors.
List decoding has been first introduced by P. ELIAS in [1]. The main idea is, rather than providing the closest codeword, to compile a list of codewords that are enough close from the transmitted message, ensuring that the transmitted codeword is part of this list.
M. Sudan and more recently V. Guruswami did a wide work on the question of decoding Reed-Solomon codes, which are known to provide good properties (see [2], [3], [4]). M. Sudan published in 1997 in [5] the first algorithm using the technic of list decoding to decode Reed-Solomon codes. This algorithm was already an important improvement in decoding this kind of code. In 1999, he published together with V. Guruswami in [6] a new version of the first algorithm where some steps were deeply improved thanks to some smart wheezes.
The interesting features of this algorithm give rise to the question of whether it could be or not adaptable to solve some open problems such as decoding in cases of insertions and deletions occurring in the received message.
This report focuses on giving a deep and clear explanation of those two algorithms which, once understood, could give tracks toward the resolution of other problems.

II. Reed-Solomon Codes

Coding theory aims at the transmission of a message while keeping it unreadable for the persons which are not receivers. To process it, a sender will encode a message, usually using some kind of mathematical keys which are also owned by the receiver to decode the message. The challenge comes from the fact that a message is most of the time transmitted through a noisy channel. The consequence is that the received message might contain errors, deletions and/or insertions. Thus, the encoding method has to provide a mean to recover the initial message from the received one.
Reed-Solomon codes are part of a type of codes named block error-correcting codes. Those concepts are defined and explained in the following.

A. General Definition: Block Error-Correcting Code

Error-correcting codes are usually linear codes using a finite field $\mathcal{F}_q$ as alphabet. The most often, $q = p^m$, where p is a prime number and $m > 1$.
The error-correcting property is achieved by block codes where a codeword contains, besides the message information, some additional data providing a way to recover from modified or erased parts of the received word. Thus, in a codeword of length n, k symbols are expressing the message information while $n - k$ symbols give a way to check and correct the information symbols (i.e. redundancy). From all those parameters follows the distance of the code, which is computed using Hamming distance between codewords:

Definition 1 *The **Hamming distance** between two words of equal length is the number of positions for which the corresponding symbols are different.*
The Hamming distance between two words x and y is denoted $D(x, y)$.

Example: Let x and y be two words of length 5 over the alphabet given by $\mathcal{F}_7$, with $x = 24615$ and $y = 34625$. Then, the Hamming distance $D(x, y)$ equals 2.

Hence, we give below the formal definition of a code, together with two important features which are the error detection and correction capabilities:

Definition 2 *A **block error-correcting code** $\mathcal{C}$ is caracterized by:*

- *The size of the alphabet q.*
- *The length of the codewords n.*
- *The information parameter k, giving the size of the code $\mathcal{C}$: $|\mathcal{C}| = q^k$.*
- *The minimum Hamming distance of the code d which is the minimum Hamming distance between any two distinct members of the code.*

The code $\mathcal{C}$ is denoted $((n,k)_q, d)$

Definition 3

- *The **error detection capability** of a code is the maximum number of errors which are allowed to occur in the transmitted word while always remaining detectable by the receiver. Then, in block error-correcting codes, the error detection capability is $d-1$.*
- *The **error correction capability** of a code is the maximum number of errors which are allowed to occur in the transmitted word while always remaining possible for the receiver to recover the original word. Then, in block error-correcting codes, the error correction capability is $\lfloor \frac{d-1}{2} \rfloor$.*

Example: We could imagine a simple (parity check) code whose principle would be to take in order and two by two the symbols of the word to be encoded, to add them (modulo q) and add the results at the end of the string. Consider the code denoted by $((n,k)_q, d)$ where:

- $n = 3$ is the length of the codewords.
- $k = 2$ is the length of the words before being encoded.
- $q = 2$ is the size of the alphabet ($\Sigma = \{0, 1\}$).
- $d = 2$ is the minimum Hamming distance between two codewords.

For instance, let $x = 01$ be the word to be encoded. Then, the corresponding codeword will be $y = 011$ where the last bit is the sum of the first two modulo 2.
One can observe that it is possible to detect one error with this particular code, given that the minimum Hamming distance is 2. However, it is not possible to correct this error even if it is detected: for certain codewords transmitted with one error, we could find back the initial codeword but not for all: for example, if y is transmitted through a noisy channel which deliver to the receiver $y' = 001$, then, there is a unique codeword c among the possibilities such that $D(c, y') = 1$, and this codeword c is actually y; but if the received word is $y'' = 111$, then there are two different codewords which could match: $y_1 = 101$ and $y_2 = 110$. Thus, it is not *always* possible for the receiver to correct one error.
Those observations agree with the definitions of error correction and detection capabilities above.

In this report, we are particularly interested in Reed-Solomon codes which are detailed in the following.

B. Reed-Solomon Codes

Definition 4 *GRS-Code $\mathcal{GRS}_{q,n,k,\vec{\alpha},\vec{v}}$*
$\mathcal{GRS}_{q,n,k,\vec{\alpha},\vec{v}}$ is an error-correcting code from the class: $((N = n, K = k+1)_q, D = n-k)$ where:

- *The alphabet Σ is over a finite field $\mathcal{F}_q$.*
- *$n \leqslant q$*
- *K is the size of a message $\vec{m} = (m_i)_{i=1}^{K}$ to be encoded.*
- *$\vec{\alpha} = (\alpha_i)_{i=1}^{n}$ is a vector of distinct elements from $\mathcal{F}_q$ named the "selector".*
- *$\vec{v} = (v_i)_{i=1}^{n}$ is a vector of non-zero elements from $\mathcal{F}_q$ named the "multiplier".*

a) GRS Coding Process:
Informally, the message to be encoded provides the coefficients of a univariate polynomial P over $\mathcal{F}_q$. Due to the size of the message m, P is of degree at most k.
The encoding of message $m = (m_i)_{i=1}^{K}$ is given by:

$$\forall 1 \leqslant j \leqslant n, (\mathcal{GRS}_{q,n,k,\vec{\alpha},\vec{v}}(\vec{m}))_j = v_j.\sum_{i=0}^{k} m_{i+1}(\alpha_j)^i$$

For simplicity's sake, the multiplier will be the 1-vector in the following.

Example:
Consider the GRS-code denoted by $\mathcal{GRS}_{q,n,k,\vec{\alpha},\vec{v}}$ where:

- $q = 5$
- $n = 5$
- $k = 3$
- $\alpha = (3, 2, 3, 1, 2)$
- $\vec{v} = (1, 1, 1, 1, 1)$

Then, the encoding of a message $m = (4, 2, 1, 1)$ results in the codeword $c = (c_i)_{1 \leqslant i \leqslant n}$ such that
$\forall i, 1 \leqslant i \leqslant n,\ c_i = P(\alpha_i)$, where P is polynomial taking the m_i's as coefficients: $P(x) = 4 + 2x + x^2 + x^3$ Thus, $c = (1, 0, 1, 3, 0)$

III. List Decoding

A. General Definition

When a codeword is transmitted over a noisy channel, the received word r is corrupted by the channel. The classical correction consists in finding the closest codeword from the received one. But, in this case, the error correction capability is only $\lfloor \frac{d-1}{2} \rfloor$. List-decoding aims to improve this bound by proceeding as follows: The receiver compiles a list of all codewords contained in a "reasonable" Hamming ball around r, that is to say, all codewords different from r in at most e places, where e is the radius of the Hamming ball. The list-decoding is successfull if the list contains the transmitted

word.
More formally, the list-decoding problem is expressed as follows:

Definition 5 ***List Decoding Problem for a Code $\mathcal{C}$***
Input: *Received word $r \in \{0,1,...,q-1\}^n$, error bound e.*
Output: *List of codewords $c_1,...,c_m \in \mathcal{C}$ such that:*

$$\forall 1 \leqslant i \leqslant m, D(r,c_i) \leqslant e$$

where $D(x,y)$ is the Hamming distance from x to y.

B. List Decoding in RS-Codes Context

Definition 6 *List Decoding Problem for a Code $\mathcal{GRS}_{q,n,k,\vec{\alpha},\vec{v}}$*
Input: *Received word $\vec{r} \in \{0,1,...,q-1\}^n$, error bound e.*
Output: *List of polynomials $P_1,...,P_s \in \mathcal{F}_q[x]$ such that:*

- $\forall 1 \leqslant i \leqslant t$, $P_i(x) = \sum_{j=0}^{k} p_{i,j}.x^j$
- $\forall 1 \leqslant i \leqslant m, D(r,(p_{i,j})_{j=0}^{k}) \leqslant e$, *where $D(x,y)$ is the Hamming distance from x to y.*

Here is a definition for the specific case of GRS list decoding:

b) GRS-Decoding / Polynomial Reconstruction:

Definition 7 *Polynomial Reconstruction Problem $\mathcal{PR}$*
Input: *Integers k, t, and n points $\{(x_i,y_i)\}_{i=1}^{n}$ where $x_i, y_i \in \mathcal{F}_q$*
Output: *All univariate polynomials P of degree at most k such that $y_i = P(x_i)$ for at least t values of $i \in \{1,...,n\}$*

One can easily see the following:

Theorem 1 *The GRS-decoding problem reduces to the polynomial reconstruction problem.*

A GRS-decoding could be performed by solving $\mathcal{PR}(k, n-e, n, \{(\alpha_i, \frac{r_i}{v_i})\}_{i=1}^{n})$
To make it clearer, $\{(\alpha_i, \frac{r_i}{v_i})\}_{i=1}^{n})$ will be simply written $\{(x_i,y_i)\}_{i=1}^{n}$ in the following.

IV. Sudan and Guruswami Algorithm

List decoding is widely used because it allows a large number of errors.
Some decoding algorithms have been designed but M. Sudan and V. Guruswami's algorithm is actually the first one presenting a real improvement in efficiency.
This algorithm published in 1999 in [6] is an improvement of a previous algorithm of Sudan published in 1997 in [5].
To allow a deep comprehension of this important algorithm, it is necessary to understand the first one. Therefore, the algorithm presented in [5] will be explained first. Then, we give the improvements provided by Sudan and Guruswami in 1999. In both cases, the informal description of the algorithm will be given first, to finally go into the formal algorithm overview, together with theorems and proofs ensuring correctness and polynomial runtime.

A. Tracks and Tricks Leading to the Algorithms

Allowing a certain number of errors e, provided $e < n - \sqrt{kn}$, the goal is to find an efficient way to output all the possible polynomials over $\mathcal{F}_q$ of degree at most k which fit at least $t = n - e$ points among the n (x_i, y_i)'s. Given that the running time of an algorithm is lower bounded by the output size, the output list has to be of polynomial size to ensure efficiency. A brute-force algorithm trying all possible polynomials is, of course, not an efficient way to solve the problem, such an enumeration would lead to an exponential runtime: $|\Sigma|^{O(k)}$. It appears that it would be preferable to output the whole set of solutions rather than enumerating them.
Now, several observations have been decisive to find a representation of the solution, from which follows the algorithm:

- The first of them is that we know efficient algorithms to factorize polynomials in finite fields (Cantor-Zassenhaus, Berlekamp). This can be useful if we could output at some point a big polynomial Q containing all the polynomials $P_1,...,P_s$ which are actually part of the solution as factors.
- As long as Q is a product of polynomials from the solution, each P_j should be a root of Q, which can be written:

$$\forall j, 1 \leqslant j \leqslant s, (y - P_j(x)) | Q(y)$$

  where the variables x and y are not related to each other. From there, Q would be a bivariate polynomial:

$$Q(x,y) = \sum_{j_1,j_2 \geqslant 0} q_{j_1,j_2} x^{j_1} y^{j_2}$$

- Furthermore, one can make the following observation:

$$\forall j, 1 \leqslant j \leqslant s,$$

$$\begin{aligned}(y - P_j(x)) | Q(x,y) &\Rightarrow Q(x, P_j(x)) = 0 \\ &\Leftrightarrow \sum_{j_1,j_2 \geqslant 0} q_{j_1,j_2} x^{j_1} P_j(x)^{j_2} = 0\end{aligned}$$

  This is a linear system in the unknowns $\{q_{j_1,j_2}\}_{j_1,j_2 \geqslant 0}$ and with n equations:

$$E_i : \sum_{j_1,j_2 \geqslant 0} q_{j_1,j_2} x_i^{j_1} y_i^{j_2} = 0, \forall i, 1 \leqslant i \leqslant n$$

  A linear system is known to be solvable in polynomial time.
- At last, to obtain a well-formed set of equations, we have to find upper bounds for j_1 and j_2. Those bounds are provided by two facts:
  1) We need to ensure that the system of equations actually has a solution, which just means that there are more unknowns than equations:

$$|\{q_{j_1,j_2}\}_{j_1,j_2 \geqslant 0}| > n$$

2) Moreover, we know that each P_j has at most t roots among the (x_i, y_i)'s, which leads to:

$$kj_2 \leqslant t$$

These two inequalities give rise to the bounds we need (Details to be showed later).

B. *First Algorithm: Sudan, 1997*

This first algorithm, published in [5], is basically built from the statements above. To make it clearer, we introduce a new notation:

Definition 8 *(Weighted Degree)*
• *For all weights $w_1, w_2 \geqslant 0$, the (w_1, w_2)-weighted degree of a monomial $x^i y^j$ is defined to be $iw_1 + jw_2$.*
• *For a bivariate polynomial $Q(x,y)$, and weights $w_1, w_2 \geqslant 0$, the (w_1, w_2)-weighted degree of Q, denoted $(w_1, w_2) - wt - deg(Q)$, is the maximum over all monomials with non-zero coefficient in Q of the (w_1, w_2) weighted degree of the monomial.*

As explained above, we firstly aim at constructing a bivariate polynomial $Q(x,y) = \sum_{j_1,j_2 \geqslant 0} q_{j_1,j_2} x^{j_1} y^{j_2}$ whose coefficients will constitute the unknowns of the linear system to be solved. Given the latter definition, it is possible to give an explicit expression of upper bounds on j_1 and j_2. Indeed, the bivariate polynomial Q is built such that the y's will end to be the polynomial we want to find, which are part of the solution. Thus, provided that each of those polynomials should have t "roots" among the input points, j_1 and j_2 have to be such that the $(1,k)$-weighted degree of Q would be strictly less than t. This gets rise to the parameters m and l specified as follows:

1) $(1,k) - wt - deg(Q) \leqslant m + lk < t$ where m and l are the respective upper bounds for j_1 and j_2.
2) Using those news parameters, we can compute a upper bound for the number of unknowns $|\{q_{j_1,j_2}\}_{j_1 \leqslant m, j_2 \leqslant l}|$. To ease the comprehension of this computation, one can take a look at the figure 1, which represents the set of unknowns which can be constructed according to the provided upper bounds:

Then we compute the sum:

$$\begin{aligned}(m+1)(l+1) + k\sum_{i=1}^{l} i &= (m+1)(l+1) + k\frac{l(l+1)}{2} \\ &= (m+1)(l+1) + k\frac{(l+1)!}{(l+1-2)!2!} \\ &= (m+1)(l+1) + k\binom{l+1}{2}\end{aligned}$$

This provides the second needed inequality from which it is now possible to determine the m and l parameters.
Given:

$$m + lk < t \tag{1}$$

$$(m+1)(l+1) + k\binom{l+1}{2} > n \tag{2}$$

The parameters optimization is done as follows:

• First, we can compute the smallest m for which (2) holds:

$$m \geqslant \frac{n+1-k\binom{l+1}{2}}{l+1} - 1 \tag{3}$$

which, taken together with (1), leads to:

$$\begin{aligned} t &\geqslant \frac{n+1-k\binom{l+1}{2}}{l+1} - 1 + lk + 1 \\ \Leftrightarrow t &\geqslant \frac{n+1}{l+1} + \frac{kl}{2}\end{aligned}$$

Setting t to its minimum value, we can then minimize the parameter l, by considering the expression above as a function of the unknown l and derivate it:

$$\begin{aligned} f(l) &= \frac{n+1}{l+1} + \frac{kl}{2} \\ f'(l) &= \frac{-(n+1)}{(l+1)^2} + \frac{k}{2} \\ f'(l) = 0 &\Leftrightarrow \boxed{l = \sqrt{\tfrac{2(n+1)}{k}} - 1} \end{aligned} \tag{4}$$

• The setting of l yields, given (3):

$$\begin{aligned} & m \geqslant \frac{n+1-k\binom{l+1}{2}}{l+1} - 1 \\ \Leftrightarrow\quad & m \geqslant \frac{n+1}{l+1} - \frac{kl}{2} - 1 \\ \Leftrightarrow\quad & m \geqslant \frac{n+1}{\sqrt{\frac{2(n+1)}{k}}} - \frac{k(\sqrt{\frac{2(n+1)}{k}} - 1)}{2} - 1 \\ & \text{(from (4))} \\ \Leftrightarrow\quad & m \geqslant \frac{(n+1)\sqrt{k}}{\sqrt{2(n+1)}} - \frac{\sqrt{2k(n+1)} - k}{2} - 1 \\ \Leftrightarrow\quad & m \geqslant \frac{2(n+1)\sqrt{k} - 2(n+1)\sqrt{k} + k\sqrt{2(n+1)}}{2\sqrt{2(n+1)}} - 1 \\ \Leftrightarrow\quad & \boxed{m \geqslant \tfrac{k}{2} - 1} \end{aligned} \tag{5}$$

c) Construction of the Linear System:
Given the parameters computed above, it is now possible to give an expression for Q:

$$Q(x,y) = \sum_{j=0}^{l} \sum_{d=0}^{m+(l-j)k} q_{dj} x^d y^j$$

Thus, the linear system S to be solved contains the n equations obtained from:

$$S : \begin{cases} E_1 : & \sum_{j=0}^{l} \sum_{d=0}^{m+(l-j)k} q_{dj} x_1^d y_1^j = 0 \\ \vdots & \vdots \\ E_n : & \sum_{j=0}^{l} \sum_{d=0}^{m+(l-j)k} q_{dj} x_n^d y_n^j = 0 \end{cases}$$

$$\underbrace{\boxed{\begin{array}{ccccc} q_{0,0} & q_{1,0} & q_{2,0} & \cdots & q_{m,0} \\ q_{0,1} & \ddots & & & \vdots \\ q_{0,2} & & \ddots & & \vdots \\ \vdots & & & \ddots & \vdots \\ q_{0,l} & \cdots & \cdots & \cdots & q_{m,l} \end{array}}}_{(m+1)(l+1)} \quad \underbrace{\boxed{\begin{array}{cccc} q_{m+1,0} & q_{m+2,0} & \cdots & q_{m+k,0} \\ q_{m+1,1} & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ q_{m+1,l-1} & \cdots & \cdots & q_{m+k,l-1} \end{array}} \quad \cdots \quad \boxed{\begin{array}{ccc} q_{m+(l-1)k+1,0} & \cdots & q_{m+lk,0} \end{array}}}_{kl+k(l-1)+\ldots+k}$$

Fig. 1. Set of unknowns which can be constructed according to the provided upper bounds

The resolution of this system can be done in polynomial time and provides the values of Q's coefficients. From now, Q is ready to be factorized.

d) Algorithm Overview:

Sudan's Algorithm (1997) [5]

Input: $k, t, \{(x_i, y_i)\}_{i=1}^{n}$ where $t = n - e$

$\boxed{1}$ Parameters l and m setting.

$$\begin{cases} m = \lceil \frac{k}{2} \rceil - 1 \\ l = \lceil \sqrt{\frac{2(n+1)}{k}} \rceil - 1 \end{cases}$$

$\boxed{2}$ Find $Q : \mathcal{F}_q^2 \to \mathcal{F}_q$ such that:

$$\begin{cases} (1,k) - wt - deg(Q) \leqslant m + lk \\ \forall i, 1 \leqslant i \leqslant n, Q(x_i, y_i) = 0 \\ Q \text{ is not identically zero} \end{cases}$$

(Construction and resolution of the linear system)

$\boxed{3}$ Factorization of Q.

$\boxed{4}$ Output all the polynomials P such that:
$(y - P(x)) | Q(x, y)$ and
$f(x_i) = y_i$ for at least t values of i.

---

e) Algorithm Benefits and Drawbacks:
This algorithm solve the problem and outputs the solutions list in polynomial time in n provided:

$$t = O(\sqrt{nk}) \text{ and } \frac{k}{n} < \frac{1}{3}$$

In the previous list decoding algorithms for RS codes, the best threshold t achieved was: $t \geqslant (n+k+1)/2$. The difference is visible when comparing the fraction of agreement (t/n) for a large n: After fixing k, we let n tend to the infinity to observe the consequences on the fraction of agreement:

- In the previous algorithms: $\lim_{n\to\infty} \frac{t}{n} = \frac{1}{2}$. This means that, however important could be n, those algorithms still need the half of the received word at least to be correct to be able to solve the problem in the allocated time.
- In Sudan's algorithm: $\lim_{n\to\infty} \frac{t}{n} = 0$, which shows that the fraction of agreement required by this algorithm approaches zero when n is important.

This improvement is already a considerable leap in list decoding of Reed-Solomon codes. However, the condition such that $\frac{k}{n} < \frac{1}{3}$ is an awkward limitation given that the codes satisfying this bound are not the most efficient we can produce. Then, one of the first motivation of the improved algorithm below was to go over this bound.

C. The Improvement: Sudan and Guruswami, 1999

This second algorithm published in [6] is based on the one explained above. It is modified in order to get an important running time improvement.
The idea is to catch the list of polynomials which are actually part of the solution faster during the factorization phase. To reach this goal, an additional constraint is applied on Q to remove the polynomials which are not good enough to be candidates for the solution. This increases the running time in the construction phase of Q, but, since the factorization is done faster and less verifications are needed in the last phase, the gain of efficiency remains considerable.

f) Informal Description:
Whereas in the first algorithm, it is required that $Q(x_i, y_i) = 0$ for all the points (x_i, y_i); in the second one, each (x_i, y_i) has to be a "singularity" of Q. This forces all the polynomials P_j obtained from the factorization to be good candidates, which means that they actually have several roots each, among the n possibilities provided by the input points. Singularities are usually defined with partial derivatives, since it is well-known that a factor being present with an exponent greater than 1 in a polynomial is also present in its derivative with an exponent decreased by 1. However, partial derivatives do not behave properly in fields with small characteristic, because a factor with an exponent being a multiple of the field characteristic would nullify the whole factor in the derivative. That is why the concept of "shifts" is used instead. This is equivalent, according to the goal we aim. The principle is that the coordinate system is shifted to make a point (x_i, y_i) becoming the origin. Then, some new conditions on the obtained shifted polynomial are added to the linear system

to force the P_j's to pass through input points (x_i, y_i) rather than just random points of Q. We also need to ensure that the P_j's property saying that they are factors of Q still holds. Here is a smart wheeze to deal with this restriction:
A polynomial P_j being part of the solution has to be a root of Q. This means that for all the P_j's part of the solution, $Q(x, P_j(x)) = 0$. We need a trick to force this equality to be true, even before the system of equations determining the coefficients of Q is solved. That is why the following fact is useful: whenever a polynomial divides another one, either the degree of the former is smaller than the degree of the latter, then the dividing polynomial is a factor of the other one; or it is the opposite, then the divided polynomial equals zero. Thus, we want to find a factor in $Q(x, P_j(x))$ of whom the exponent could be increased while ensuring that it is still a divider of $Q(x, P_j(x))$.

g) Formal Description:
We first give a formal definition of a shifted polynomial:

Definition 9 *(Shifted Polynomial)*
Given a polynomial Q such that $\forall(x, y) \in \mathcal{F}^2$, $Q(x, y) = \sum_i \sum_j q_{i,j} x^i y^j$ and a point $(\alpha, \beta) \in \mathcal{F}^2$, the shifted polynomial $Q_{(\alpha,\beta)}$ to the point (α, β) is written:

$$Q_{(\alpha,\beta)}(x, y) = Q(x + \alpha, y + \beta)$$

It provides the following relation between the coefficients $q_{i,j}$ of Q and the coefficients $(q_{\alpha,\beta})_{i,j}$ resulting from the shift:

$$(q_{(\alpha,\beta)})_{i,j} = \sum_{i' \geqslant i} \sum_{j' \geqslant j} \left(\begin{array}{c} i' \\ i \end{array} \right) \left(\begin{array}{c} j' \\ j \end{array} \right) q_{i',j'} \alpha^{i'-i} \beta^{j'-j} \quad (6)$$

The equation (6) is necessary to write the new linear system S' to be solved. But, first of all, we need to compute new parameters to determine the precise expression of Q, of whom the coefficients $q_{i,j}$ are the unknowns of the target system.
In our case, the shifted polynomial to the point (x_i, y_i) will be written $Q^{(i)}$.

From this definition, we can deduce some equalities which will allow us to get the algorithm working.
Given a polynomial P which would be part of the solution, we define a new polynomial P' such that:

$$P'(x) = P(x + x_i) - y_i \quad (7)$$

Then, we observe that $P'(0) = 0$ which means that there exists a polynomial P'' such that:

$$P'(x) = xP''(x) \quad (8)$$

On an other hand, it is true that:

$$\begin{aligned} Q(x, P(x)) &= Q^{(i)}(x - x_i, P(x) - y_i) \\ &\qquad \text{(By the shift definition)} \\ &= Q^{(i)}(x - x_i, P(x - x_i + x_i) - y_i) \\ &= Q^{(i)}(x - x_i, P'(x - x_i)) \\ &\qquad \text{(Where P' is defined in (7))} \end{aligned}$$

Thus, substituting $P'(x - x_i)$ by (8):

$$Q(x, P(x)) = Q^{(i)}(x - x_i, xP''(x - x_i))$$

At last, this expression shows that, if there is no coefficient in $Q^{(i)}$ of total degree less that a value r, then x^r divides $Q^{(i)}(x - x_i, xP''(x - x_i))$.
By shifting back, we obtain that, under the same condition, $(x - x_i)^r$ divides $Q(x, P(x))$.
This is from where the parameter r, and as a consequence, the parameter l, can both be computed.

h) Parameters Setting:
This algorithm requires two parameters of whom the optimization is entangled.

- The parameter r:
  Combining the informal and formal descriptions, we found the factor of $Q(x, P_j(x))$ of whom the exponent can be increased until reaching the necessary degree to force $Q(x, P_j(x))$ being zero. The first parameter r provides the needed value of the exponent to achieve this property. Knowing that the sought polynomials must agree with at least t of the n input points, there will be at least t factors $(x - x_i)^r$ dividing $Q(x, P_j(x))$ for a given polynomial P_j. Thus, rt has to be strictly greater than the degree if Q. Once r is determined, one just has to nullify, for all $i \in [1, \ldots, n]$, all the $Q^{(i)}$'s coefficients of total degree less than r. This will provide some new equations in the linear system to be solved:

$$\forall i, 1 \leqslant i \leqslant n,\ \forall j_1, j_2 \geqslant 0 \text{ such that } j_1 + j_2 < r$$

$$q^{(i)}_{j_1,j_2} = \sum_{j'_1 \geqslant j_1} \sum_{j'_2 \geqslant j_2} \left(\begin{array}{c} j'_1 \\ j_1 \end{array} \right) \left(\begin{array}{c} j'_2 \\ j_2 \end{array} \right) q_{j'_1,j'_2} \alpha^{j'_1 - j_1} \beta^{j'_2 - j_2} = 0$$

(By definition (6))

  From now, we know, according to r, how many equations the new linear system S' consists in. S' contains the n old equations provided by the first condition, and the new ones, n for each of the possibility of positive integers pairs (j_1, j_2) such that $j_1 + j_2 < r$. The resulting number of equations in S' is:

$$n + n\sum_{i=1}^{r} i = n + n\frac{r(r+1)}{2} = n(\left(\begin{array}{c} r+1 \\ 2 \end{array} \right) + 1) \quad (9)$$

  Now, we need to get some information from the setting of the second parameter to continue the computations.
- The parameter l:
  Let l be the degree of Q, i.e. $(1, k) - wt - deg(Q) \leqslant l$.
  We saw above that rt has to be strictly greater than the degree of Q, i.e. $rt > l$.
  Knowing that it is always better to minimize the parameters we can, we set l as follows:

$$rt - 1 \quad (10)$$

  From l, we can determine the number of monomials in Q, and then the number of unknowns in the system (cf.

$$\underbrace{\begin{array}{ccccccccccc} q_{0,0} & q_{1,0} & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & q_{l,0} \\ q_{0,1} & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & q_{l-k-1,1} & & \\ q_{0,2} & \cdots & \cdots & \cdots & \cdots & \cdots & q_{l-2k-1} & & & & \\ \vdots & \vdots & \ddots & \cdots & \cdots & & & & & & \\ q_{0,\lfloor \frac{l}{k} \rfloor -1} & \cdots & q_{k-1,\lfloor \frac{l}{k} \rfloor -1} & & & & & & & & \\ q_{0,\lfloor \frac{l}{k} \rfloor} & & & & & & & & & & \end{array}}$$

Fig. 2. Monomials in Q

figure 2).

This figure leads to the following computation :

$$figure\,2 = 1 + \sum_{i=1}^{\lfloor \frac{l}{k} \rfloor} = 1 + k\frac{\lfloor \frac{l}{k} \rfloor(\lfloor \frac{l}{k} \rfloor + 1}{2}) = 1 + \frac{l(l+k)}{2k} \tag{11}$$

This gives rise to the last inequality which has to hold to make the algorithm work, and so to find the explicit expression of r.

- Finally, r:
  Given r, we have to be sure of the existence of a solution for the linear system S'. Then, the numbers of unknowns has to be greater than the number of equations which, thanks to expressions (9) and (11), is given by:
  $$n\left(\begin{array}{c} r+1 \\ 2 \end{array}\right) + 1) < 1 + \frac{l(l+k)}{2k}$$
  Then, this inequality is simplified to get an expression easier to solve:
  $$\begin{aligned} n\left(\begin{array}{c} r+1 \\ 2 \end{array}\right) &< \frac{l(l+2)}{2k} && (k \geqslant 1) \\ n\left(\begin{array}{c} r+1 \\ 2 \end{array}\right) &< \frac{(rt-1)(rt+1)}{2k} && \text{(from (10))} \\ n\frac{(r+1)!}{(r-1)!} &< \frac{r^2t^2-1}{k} \\ n(r^2+r) &< \frac{r^2t^2-1}{k} \\ (r^2+r)kn &< r^2t^2-1 \\ 0 &< r^2(t^2-kn) - t - rkn \end{aligned}$$
  Then, we solve the following second degree equation to get the minimum value for r:
  $$r^2(t^2-kn) - knr - 1 = 0$$
  We keep the greatest of the two solutions to ensure correctness:
  $$r = \frac{kn + \sqrt{k^2n^2 + 4(t^2-kn)}}{2(t^2-kn)}$$
  Finally, r is set as follows:
  $$r = 1 + \lfloor \frac{kn + \sqrt{k^2n^2 + 4(t^2-kn)}}{2(t^2-kn)} \rfloor$$

Now, the method is fully explained and parameters set, the algorithm can be processed as shown below:

i) Algorithm Overview:

Sudan and Guruswami's Algorithm (1999) [6]

Input: $k, t, \{(x_i, y_i)\}_{i=1}^{n}$ where $t = n - e$

$\boxed{1}$ Parameters r and l setting:

$$\begin{cases} r = 1 + \lfloor \frac{kn+\sqrt{k^2n^2+4(t^2-kn)}}{2(t^2-kn)} \rfloor \\ l = rt - 1 \end{cases}$$

$\boxed{2}$ Find $Q : \mathcal{F}_q^2 \rightarrow \mathcal{F}_q$ such that:

$$\begin{cases} (1,k) - wt - deg(Q) \leqslant l \\ \forall i, 1 \leqslant i \leqslant n, Q^{(i)} \text{ is a shift of } Q \text{ to } (x_i, y_i) \\ Q \text{ is not identically zero} \end{cases}$$

$\boxed{3}$ Factorization of Q into polynomials of degree at most k.

$\boxed{4}$ Output all the polynomials P such that:
$(y - P(x))|Q(x,y)$ and
$f(x_i) = y_i$ for at least t values of i.

j) Algorithm Gains:
Whereas the previous algorithm could not deal with RS codes whose parameters would lead to $\frac{k}{n} \geqslant \frac{1}{3}$, the algorithm presented in [6] can solve the problem and output the correct solutions list within a polynomial running time as long as the upper bound on the number of errors is satisfied, as shown in the following theorems from the article:

Theorem 2 *The algorithm of [6] on imputs n, k, t and the points $\{(x_i, y_i) : 1 \leqslant i \leqslant n\}$, correctly solves the polynomial reconstruction problem provided $t > \sqrt{kn}$*

Theorem 3 *For a given family of GRS codes of constant rate $\kappa = \frac{k}{n}$, an error-rate of $\epsilon = \frac{e}{n} = 1 - \sqrt{\kappa}$ (i.e. $e < n - \sqrt{kn}$) can be list-decoded in time $O(n^{15})$. When $\epsilon < 1 - \sqrt{\kappa}$, then the decoding time reduces to $O(n^3)$*

(The proofs are omitted here since they are clear in the paper)

V. Dealing with Insertions

A. [6] for Insertions and Deletions Problem ?

One question to be answered was whether the algorithm described in [6] could be adapted to list decoding of GRS codes in cases of insertions and/or deletions. It finally appears that this adaptation could not happen because the input "shapes" would be very different: In the case of simple errors, the input contains a list of points (x_i, y_i) among whom it is known that t are correct.
In the case of deletions and insertions, the input will contain a set of possible x_i's for each y_i. Furthermore, whereas those sets are surely containing the actual correct value of x_i for each y_i in the case of insertions, this is not sure anymore in cases of deletions occurring.
These observations lead to think that the algorithm described in this report could not be modified efficiently to solve this new problem.
Once this fact is stated, it appears that a few analysis might answer some questions. Thus, in the next part will be computed the size of the list which would have to be output by a list decoding algorithm in case of insertions present in the received word.

B. Size of the List in Insertions Cases

Let x be a word encoded by a GRS code denoted $\mathcal{GRS}_{q,n,k,\vec{\alpha},\vec{v}}$ as defined in the beginning. Let r be the number of insertions which occurred in the received word y. Thus, the length of y is $n + r$.
We want to count how many words could be x, given y, i.e. the size $|L|$ of the expected output list. There are as many codewords as manners to place the r insertions among the $n + r$ symbols of the received word. So the size of the list is given by:

$$|L| = \binom{n+r}{r}$$

Finally, for a fixed n, an evaluation of the list size for a given r will be denoted $|L|_r$ and it appears that this is bounded by a polynomial in n:

$$|L|_r = O(n^r)$$

Then, it is always possible to process a brute force algorithm which would check all the possible solutions in polynomial time, however, this polynomial could be large.

C. Open Problem: Deletions

Adding the possibility of deletions occurrences is a hard problem since at one position of the string, the symbol could be either still present in the received word, or have been deleted. Then, trying to solve the problem in a brute force manner is no longer efficient because the set of possibilities for one symbol in the codeword we are looking for is actually the whole alphabet.
From this, it appears obvious that the deterministic way will not provide a good algorithm. A probabilistic algorithm might give good results, but then, we are not fully sure that the output list always contains the sought codeword.

VI. Conclusion

The comprehension of the algorithm of M. Sudan and V. Guruswami is definitely essential to be able to use it in the resolution of other problems. Indeed, since each single parameter has a precise and crucial role, it is important to understand each of them to be sure that a new problem is reducible to the polynomial reconstruction problem solved in this algorithm. Furthermore, even an apparently close problem, as it can be thought for the reconstruction in cases of insertions and deletions, appears to be totally different. However, this algorithm takes advantage from some clever mathematical properties which look universal, and then, it is expectable that one could use this technic to solve or improve some questions remaining unanswered.
On another hand, dealing with insertions and deletions appeared far more difficult than expected, and the solution given in [6] is apparently not the track to be followed. The probabilistic way might be gainful but it requires an effort of formalization and a precise problem stating to give a strong frame to this exploration.

References

[1] P. Elias, “List decoding for noisy channels,” in *1957-IRE WESCON Convention Record.* Massachusetts Institute of Technology: Research Laboratory of Electronics, 1957, pp. 94–104.

[2] V. Guruswami and M. Sudan, “List decoding algorithms for certain concatenated codes,” in *STOC ’00: Proceedings of the thirty-second annual ACM symposium on Theory of computing.* New York, NY, USA: ACM, 2000, pp. 181–190.

[3] M. Sudan, “List decoding: Algorithms and applications,” *SIGACT News*, vol. 31, pp. 16–27, 2000.

[4] ——, “Maximum likelihood decoding of reed solomon codes,” in *Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science*, 1996, pp. 164–172.

[5] ——, “Decoding of reed solomon codes beyond the error-correction bound,” *Journal of Complexity*, vol. 13, no. 1, pp. 180–193, 1997.

[6] V. Guruswami and M. Sudan, “Improved decoding of reed-solomon and algebraic-geometric codes,” *IEEE Transactions on Information Theory*, vol. 45, pp. 1757–1767, 1999.