Codes Correcteurs d'Erreurs

TD 3 : Décodage des codes de Reed-Solomon
Berlekamp-Welch, Berlekamp-Massey, Sudan, Gurusuami-Sudan

Exercice 1.

À l'aide de SageMath, vérifier la propriété MDS du code $\mathrm{GRS}_k(\mathbf{x}, \mathbf{y})$ où :

1. $q = 8$, $n = 6$, $k = 4$, $\mathbf{y} = \mathbf{1}$;
2. $q = 8$, $n = 6$, $k = 4$, $\mathbf{y} \neq \mathbf{1}$;
3. $q = 64$, $n = 64$, $k = 2$, $\mathbf{y} = \mathbf{1}$;
4. $q = 64$, $n = 64$, $k = 62$, $\mathbf{y} = \mathbf{1}$.

On pourra, si cela semble raisonnable, simplement énumérer tous les mots du code.

Exercice 2. *(Décodage unique des codes de Reed-Solomon)*

Partie 1. Berlekamp-Welch

Soit $\mathbf{x} := (x_1, \cdots, x_n) \in \mathbb{F}_q^n$ avec $(x_i)_i$ deux à deux distincts. On considère alors le code $\mathrm{RS}_k(\mathbf{x})$ où $1 \leq k < n$. On pose $t := \left\lfloor \frac{n-k}{2} \right\rfloor$.

Soit $f \in \mathbb{F}_q[X]_{<k}$ et soit $\mathbf{c} := (f(x_1), \cdots, f(x_n)) \in \mathrm{RS}_k(\mathbf{x})$ un mot de code émis. On note alors $\mathbf{u} := \mathbf{c} + \mathbf{e}$ le mot reçu où $|\mathbf{e}| \leq t$.

On pose le polynôme suivant :

$$E(X) := \prod_{\substack{i=1 \\ e_i \neq 0}}^{n} (X - x_i)$$

(Noter que E n'est pas connu par le destinataire)

1. Montrer que pour tout $i \in [\![1, n]\!]$, on a $u_i E(x_i) = f(x_i) E(x_i)$.

Soit les polynômes inconnus $A \in \mathbb{F}_q[X]_{\leq t}$ et $B \in \mathbb{F}_q[X]_{<n-t}$. On pose le système linéaire suivant :

$$\mathcal{S} \ : \ \{u_i A(x_i) = B(x_i)\}_{i \in [\![1,n]\!]}$$

2. Déduire de la question 1 que $\mathcal{S}$ a une solution non triviale.
3. Montrer que pour toute solution non triviale (A_1, B_1) de $\mathcal{S}$, on a $A_1 \neq 0$.
4. Soient (A_1, B_1) et (A_2, B_2) deux solutions non triviales de $\mathcal{S}$ et soit $R := B_1 A_2 - B_2 A_1$. Montrer que pour tout $i \in [\![1, n]\!]$, $R(x_i) = 0$. En déduire que $R = 0$ et donc que $\frac{B_1}{A_1} = \frac{B_2}{A_2}$.
5. Montrer que pour toute solution non triviale (A_1, B_1) de $\mathcal{S}$, on a $\frac{B_1}{A_1} = f$. En déduire l'algorithme de Berlekamp-Welch pour décoder $\mathbf{u}$ dans $\mathrm{RS}_k(\mathbf{x})$.
6. Montrer que le décodage de Berlekamp-Welch est en temps polynomial.

Partie 2. Berlekamp-Massey

En fait, on peut éviter d'avoir à effectuer une élimination Gaussienne dans l'algorithme de Berlekamp-Welch ; ceci va nous amener à construire le décodeur de Berlekamp-Massey (qui permet de faire le lien entre les codes RS et les codes BCH).

1. En utilisant une interpolation de Lagrange, construire l'unique polynôme $U \in \mathbb{F}_q[X]_{<n}$ tel que :

$$\forall i \in [\![1, n]\!], U(x_i) = u_i$$

2. Montrer, à l'aide du théorème des restes chinois, que :

$$E \cdot U \equiv E \cdot f \mod \Pi$$

où $\Pi(X) := \prod_{i=1}^n (X - x_i)$.

L'algorithme d'Euclide Étendu nous permet de trouver des identités de Bezout de la forme :

$$C \cdot \Pi + A \cdot U = B$$

où A, B et C sont des polynômes sur $\mathbb{F}_q$.

Algorithm. Extended Euclide

input: $U \in \mathbb{F}_q[X]_{<n}$
parameter: $\Pi := \prod_{i=1}^n (X - x_i) \in \mathbb{F}_q[X]_{<n}$
output : $A, B, C \in \mathbb{F}_q[X]$ such that $B = \gcd(\Pi, U)$ and $C\Pi + AU = B$

$(C_0, A_0, B_0) \leftarrow (1, 0, \Pi)$
$(C_1, A_1, B_1) \leftarrow (0, 1, U)$
$i \leftarrow 0$
repeat
  $Q, B_{i+2} \leftarrow$ quotient and remainder of the Euclidean division $B_i \vdash B_{i+1}$
  $A_{i+2} \leftarrow A_i - Q \cdot A_{i+1}$
  $C_{i+2} \leftarrow C_i - Q \cdot C_{i+1}$
  $i \leftarrow i + 1$
until $B_{i+1} = 0$
$i_{\max} \leftarrow i$
return (A_i, B_i, C_i)

Deux propriétés bien connues de l'algorithme d'Euclide Etendu sont que pour tout $i \in [\![0, i_{\max}]\!]$:

$$C_i \cdot \Pi + A_i \cdot U = B_i$$

et :

$$\deg(B_i) > \deg(B_{i+1})$$

3. Montrer par récurrence que pour tout $i \in [\![0, i_{\max}]\!]$:

$$\begin{cases} \deg(A_{i+1}) \geq \deg(A_i) \\ \deg(A_{i+1}) = \deg(\Pi) - \deg(B_i) \end{cases}$$

4. Montrer qu'en arrêtant l'algorithme d'Euclide Étendu prématurément, il est possible de trouver (A_{i_0}, B_{i_0}) tel que :

$$\begin{cases} A_{i_0} \cdot U \equiv B_{i_0} \mod \Pi \\ \deg(A_{i_0}) \leq t \\ \deg(B_{i_0}) < n - t \end{cases}$$

5. En déduire l'algorithme de Berlekamp-Massey pour décoder les codes de Reed-Solomon.

Remarque. Une implémentation naïve de l'élimination Gaussienne a un coût de l'ordre de $O(n^3)$ tandis que l'algorithme d'Euclide étendu à un coût de l'ordre de $O(n^2)$ (voir moins avec des astuces). L'algorithme de Berlekamp-Massey est donc plus rapide que celui de Berlekamp-Welch et sera donc à privilégier pour un décodage unique des codes de Reed-Solomon.

Exercice 3. *(Décodage en liste des codes de Reed-Solomon)*

Partie 1. Sudan

En 1997, Madhu Sudan proposa un décodage en liste en temps polynomial pour les codes de Reed-Solomon. Ces travaux ont été une avancée révolutionnaire dans le domaine des codes algébriques.

Notation. *Pour un polynôme bivarié $P \in \mathbb{F}_q[X,Y]$, on note $\deg_X(P)$ (resp. $\deg_Y(P)$) le degré de P vu comme un polynôme en X (resp. en Y). On note $\deg(P)$ le degré du monôme de plus haut degré sachant que le degré d'un monôme est la somme des degrés en X et en Y ; autrement dit, $\deg(P) := \deg_X(P(X,X))$. Par exemple, pour $P = 1 + 3X^4 + 2Y + XY + 5X^3Y^2$ on a :*

$$\deg_X(P) = 4 \qquad \deg_Y(P) = 2 \qquad \deg(P) = 5$$

Comme pour l'exercice 2, on fixe $\mathbf{x} := (x_1, \cdots, x_n) \in \mathbb{F}_q^n$ avec $(x_i)_i$ deux à deux distincts. On considère alors le code $\mathrm{RS}_k(\mathbf{x})$ où $1 \leq k < n$ et on pose $t := \left\lfloor \frac{n-k}{2} \right\rfloor$.

1. Montrer que le décodage unique de Berlekamp-Welch pour décoder $\mathrm{RS}_k(\mathbf{x})$ peut être reformulé ainsi :

Algorithm. Berlekamp-Welch

input: $\mathbf{u} \in \mathbb{F}_q^n$
output: $f \in \mathbb{F}_q[X]_{<k}$ such that $\Delta(\mathbf{u}, \mathbf{c}) \leq t$ with $\mathbf{c} := (f(x_i))_i$

Interpolation. Construct a polynomial $Q \in \mathbb{F}_q[X,Y]$ of the form $Q = Q_0(X) + Q_1(X)Y$ such that:

$$\begin{cases} \deg(Q_0) < n - t \\ \deg(Q_1) \leq n - k - t \\ \forall i \in [\![1, n]\!], \quad Q(x_i, u_i) = 0 \end{cases}$$

Root finding. Compute the root $f \in \mathbb{F}_q[X]$ of Q as a polynomial in Y; that is compute $f = -\frac{Q_0}{Q_1}$

L'idée de Sudan est la suivante : pour corriger plus de $t := \left\lfloor \frac{n-k}{2} \right\rfloor$ erreurs, nous devons permettre à l'algorithme de Berlekamp-Welch de retourner plus d'une solution. Or les solutions sont les racines de Q vu comme un polynôme en Y et $\deg_Y(Q) = 1$ par construction. On va donc choisir différemment Q pour avoir $\deg_Y(Q) = \ell > 1$. L'algorithme de Sudan est alors donné par le pseudo-code suivant :

Algorithm. Sudan

input: $\mathbf{u} \in \mathbb{F}_q^n$
parameters: integers $\ell > 0$ and $r \geq t$
output: all the $f \in \mathbb{F}_q[X]_{<k}$ such that $\Delta(\mathbf{u}, \mathbf{c}) \leq r$ with $\mathbf{c} := (f(x_i))_i$

Interpolation. Construct a polynomial $Q \in \mathbb{F}_q[X,Y]$ of the form

$$Q = \sum_{j=0}^{\ell} Q_j(X) Y^j$$

such that

$$\begin{cases} \forall j \in [\![1, \ell]\!], \quad \deg(Q_j) + j(k-1) < n - r \\ \forall i \in [\![1, n]\!], \quad Q(x_i, u_i) = 0 \end{cases}$$

Root finding. Compute all the root $f \in \mathbb{F}_q[X]$ of Q as a polynomial in Y; that is compute all the polynomial f such that $Q(X, f(X)) = 0$

2. Montrer que toutes les solutions du problème de décodage sont bien retournées par l'algorithme.

3. Quelle est le nombre maximum de solutions que peut retourner l'algorithme? Noter que parmi ces solutions, certaines peuvent ne pas respecter la condition

$$\Delta\big(\mathbf{u}, (f(x_1), \cdots, f(x_n))\big) \leq r$$

4. Quelle est la valeur maximale $\ell_{\max}$ du paramètre ℓ? En choisissant $\ell = \ell_{\max}$, la liste de mots de code retournée par l'algorithme de Sudan est-elle de taille polynomiale en n?

5. Dans la suite, on choisit $\ell = \ell_{\max}$. On rappelle que l'étape d'interpolation de l'algorithme de Sudan consiste à résoudre un système linéaire. Quelle est le nombre d'équations de ce système? Le nombre d'inconnues? En déduire un critère qui garantisse que le système a une solution non triviale.

6. Estimer la complexité de l'algorithme de Sudan.

7. À partir du critère trouvé à la question 4, donner une borne asymptotique lorsque $n \to \infty$ du rayon de décodage relatif $\rho := \frac{r}{n}$.
8. Comparer le nombre d'erreurs corrigibles de l'algorithme de Sudan avec celui de l'algorithme de Berlekamp-Welch. Conclure.

Partie 2. Gurusuami-Sudan

Nous avons vu juste avant que l'algorithme de Sudan a une limite considérable : elle est plus intéressante que l'algorithme de Berlekamp-Welch seulement pour des codes de rendement faible. Dans cet exercice, nous allons construire le décodage de Gurusuami-Sudan qui est une amélioration de l'algorithme de Sudan. Nous allons voir que cet algorithme permet d'atteindre une borne importante ; à savoir la borne de Johnson que l'on a vu dans le cours.

Définition. *Soit $Q \in \mathbb{F}_q[X,Y]$ et soient a et $b \in \mathbb{F}_q$. Si $Q(X-a, Y-b)$ ne possède aucun terme de degré strictement inférieur à un entier m, alors on dira que Q a une multiplicité supérieure ou égale à m en (a,b).*

Rappelons que notre objectif est de garder un nombre raisonnable d'éléments dans notre liste de décodage tout en augmentant le rayon r de correction. Dans l'algorithme de Sudan, le rayon de correction r était limité par les conditions nous permettant de construire le polynôme Q. En effet, le fait de devoir avoir plus d'inconnues que d'équations pour pouvoir construire Q nous donnait une borne sur r ; avec la notion de multiplicité, on peut augmenter ce rayon de décodage. On obtient alors l'algorithme de Gurusuami-Sudan suivant :

Algorithm. Gurusuami-Sudan

input: $\mathbf{u} \in \mathbb{F}_q^n$
parameters: integers $m \geq 1$, $\ell > 0$ and $r \geq t$
output: all the $f \in \mathbb{F}_q[X]_{<k}$ such that $\Delta(\mathbf{u}, \mathbf{c}) \leq r$ with $\mathbf{c} := (f(x_i))_i$

Interpolation. Construct a polynomial $Q \in \mathbb{F}_q[X,Y]$ of the form

$$Q = \sum_{j=0}^{\ell} Q_j(X) Y^j$$

such that

$$\begin{cases} \forall j \in [\![1, \ell]\!], & \deg(Q_j) + j(k-1) < m(n-r) \\ \forall i \in [\![1, n]\!], & Q \text{ a une multiplicité } \geq m \text{ en } (x_i, u_i) \end{cases}$$

Root finding. Compute all the root $f \in \mathbb{F}_q[X]$ of Q as a polynomial in Y; that is compute all the polynomial f such that $Q(X, f(X)) = 0$

1. Montrer que toutes les solutions du problème de décodage sont bien retournées par l'algorithme.
2. Quelle est la valeur maximale $\ell_{\max}$ du paramètre ℓ?
3. Dans la suite, on choisit $\ell = \ell_{\max}$. On rappelle que l'étape d'interpolation de l'algorithme de Sudan consiste à résoudre un système linéaire. Expliquer comment construire ce système. Quelle est son nombre d'équations? D'inconnus? En déduire un critère qui garantisse que le système a une solution non triviale (on se satisfera d'un critère asymptotique lorsque $n \to \infty$).
4. Proposer une valeur pour le facteur de multiplicité m qui garantisse que la liste de mots de code retournée par l'algorithme de Gurusuami-Sudan soit de taille polynomiale en n tout en permettant d'atteindre asymptotiquement la borne de Johnson lorsque $n \to \infty$.
5. Estimer la complexité de l'algorithme de Gurusuami-Sudan.
6. Faire un bilan comparatif des décodeurs de Berlekamp-Welch, Sudan et Gurusuami-Sudan.

Exercice 4. *(Projet)*

L'exercice a pour but d'implémenter les codes de Reed-Solomon. Pour cela, on utilisera SageMath. Noter que SageMath est une sur-couche de Python qui est un langage orienté objet ; on pourra donc créer une classe `ReedSolomon` dont les attributs seront les paramètres du code et les méthodes, les différents codeur/décodeurs.

1. Implémenter une fonction `canal` qui simule un canal q-aire produisant **exactement** t erreurs.
2. Implémenter le codeur RS (cf. exercice 1.).
3. Implémenter le décodage unique de Berlekamp-Welch et de Berlekamp-Massey avec SageMath (cf. exercice 2.). Comparer expérimentalement leurs complexités. On pourra se concentrer notamment sur des codes RS définis sur $\mathbb{F}_{256}$.
4. Implémenter le décodage en liste de Sudan et de Gurusuami-Sudan avec SageMath (cf. exercice 3.). Ceux-ci ne devront retourner qu'un seul mot de code : si la liste contient strictement plus d'un élément, alors on choisira le mot de code le plus vraisemblable. On pourra encore une fois se concentrer sur des codes RS définis sur $\mathbb{F}_{256}$ pour les tests.
5. Effectuer un *benchmark* sur l'ensemble des décodeurs RS implémentés : tester les limites du nombre d'erreurs corrigibles, tester différents rayons de décodage, comparer les complexités...
6. **[Bonus]** Renommer votre classe `ReedSolomon` en `GeneralisedReedSolomon` et effectuer les modifications qui s'imposent.