Reconstituer le théorème chinois en présence d'erreurs

Mr LAVOIX John, Mr GRANERO Fabien

7 mai 2021

Résumé

Ce rapport est une synthèse de nos travaux durant ce semestre, sur la reconstitution du théorème des restes chinois en présence d'erreurs. Grâce au soutien de notre professeur Mr ZEMOR, qui nous a guidé tout au long de ce projet nous avons découvert et maitrisé plusieurs méthodes pour parvenir à notre objectif.

Table des matières

Chapitre 1

théorème des restes

1.1 théorème des restes chinois et son histoire

Les premières apparitions du théorème des restes chinois ont eu lieu au XIII° siècle dans un livre de mathématique chinois de Qin JUSHIO publié en 1247. Cependant, on avait déjà découvert ce théorème auparavant dans un livre de Sun ZI au III° siècle. Le théorème consiste en : On pose p1,...,pk des entiers premiers 2 à 2. Pour tout n1,...,nk, il existe un entier x tel que :
$x \equiv n_1 \; (mod \, p_1)$
$x \equiv n_2 \; (mod \, p_2)$
$x \equiv n_k \; (mod \, p_k)$

Nous allons dans un premier temps démontrer l'unicité de ce théorème.
Tout d'abord, nous cherchons l'existence d'une solution :
On a $\forall k \in [1; y]$, $x \equiv n_k (mod \, p_k)$
$\forall k \in [1; y]$, on note $P_k = \frac{P}{p_k}$ où $P = p_1 \times ... \times p_y$
On voit assez simplement que P_k et p_k sont premiers entre eux car tous les entiers p_i sont premiers entre eux 2 à 2. Donc P_k est inversible modulo p_k, car en faisant le théorème d'euclide, on va trouver $u, v \in Z$ tel que $P_k \times u + p_k \times v = d$ avec d le PGCD. Or P_k et p_k sont premiers entre eux donc $d = 1$ et on trouve facilement l'inverse de P_k modulo p_k.
On note alors u_k le nombre tel que $u_k \times P_k + v_k \times p_k = 1$, soit $u_k \times P_k \equiv 1(p_k)$.
Soit $x = \sum_{k=1}^{y} u_k \times P_k \times n_k$. On pose $i \in [1; y]$ et $k \neq i$ alors $P_i \equiv 0 (mode \, p_k)$ car $P_i = p_1 \times ... \times p_k \times ... \times p_y$.
On a donc $x \equiv u_i P_i a_i (mod \, p_i)$, mais on sait que $u_i P_i \equiv 1 (mod \, p_i)$ d'où $x \equiv a_i (mod \, p_i)$. Sachant que cette équation est vrai pour tout i, on a x solution du système.
Par conséquent, nous avons prouvé l'existence d'une solution.

Ensuite, nous montrons l'unicité du système.
On a x une solution du système, posons y une autre solution du système. On a alors $x - y \equiv 0 (mod p_k)$, soit p_k divise $x - y$. Sachant que les p_k sont premiers 2 à 2 entre eux, on a donc $x - y \equiv 0(mod\, P)$, donc P divise x-y, ou encore que $x \equiv y(mod\, P)$. Par cela, nous avons montré l'unicité modulo P.

Pour illustrer ce théorème, nous allons donner un exemple, extrait du livre de Sun ZI qui a proposé une solution :
Choisissons des objets comme des bonbons : si on les repartis pour 3 enfants, il en reste 2, si on les répartis pour les 3 enfants et leurs parents (soit 5 personnes), il en reste 3. Enfin si on rajoute les 2 cousins (soit 7 personnes), il reste alors 2 bonbons. On a donc :
$x \equiv 2\ (mod\, 3)$
$x \equiv 3\ (mod\, 5)$
$x \equiv 2\ (mod\, 7)$

La question que l'on se pose à présent est : combien y a t'il de bonbons ?
Grâce au théorème des restes chinois, on peut trouver la réponse. Nous avons en réalité plusieurs réponse, c'est-à-dire que tout les nombres congru à x modulo N sont des bonnes réponses. Avant de répondre à ce problème, il faut tout d'abord examiner l'algorithme.

1.2 Ces Domaines d'Application

1.3 notre algorithme

Notre fonction en Python du théorème des restes chinois étant un peu lourde, nous allons montrer l'algorithme ci-dessous (voir annexe).
On obtient donc :
Soit p_i le i-ème terme de la liste des modulos, on note
$P_i = \frac{p}{p_i} = p_1 p_2 ... p_{i-1} p_{i+1} ... p_k$
on a donc P_i et p_i qui sont premiers entre eux.
Il faut alors faire l'algorithme d'Euclide étendu sur P_i et p_i, ce qui nous donne : $1 = P_i u_i + p_i v_i$ où $u_i, v_i \in Z$
On pose donc $e_i = u_i P_i$ avec $e_i \equiv 1 \ (mod\, p_i)$ et $e_i \equiv 0 \ (mod\, e_j)$ avec $i \neq j$
On trouve alors une solution qui est $x = \sum_{i=1}^{k} p_i e_i$.

Nous pouvons à présent répondre à l'exemple précédent. Nous avions :
$x \equiv 2 \ (mod\, 3)$
$x \equiv 3 \ (mod\, 5)$
$x \equiv 2 \ (mod\, 7)$
On obtient $P_1 = 5 \times 7 = 35$, $P_2 = 3 \times 7 = 21$, et $P_3 = 3 \times 5 = 15$
On fait l'algorithme d'Euclide étendu sur P_1 et p_1 qui donne $-3 \times 23 + 2 \times 35 \times 1 = 1$, par conséquent on trouve $e_1 = 2 \times 35$
Idem sur P_2 et p_2 qui donne $21 \times 1 - 5 \times 4 = 1$, donc $e_2 = 21$
Enfin, on a $15 \times 1 - 7 \times 2 = 1$ avec $e_3 = 15$
Le résultat est $x = 2 \times 35 + 3 \times 21 + 2 \times 15 = 233$. Nous avons, comme dit précédement, plusieurs résultats qui sont les entiers congrus à 233 modulo 105.
$233 \equiv 23 (mod\, 105)$
Le résultat final est donc $23 + 105k$, $k \in Z$. Si on reprend notre problème, on a donc 23 bonbons.

Comme vous pouvez le voir, nous avons utilisé l'algorithme d'Euclide étendu. Celui-ci prend en paramètres a et b, deux entiers. Il renvoit d le PGCD de a et b et un couple $(u, v) \in Z$ tel que $d = au + bv$.
Pour notre utilisation, nous avons un peu modifié cette algorithme, car pour utiliser le théorème des restes chinois, les nombres sont premiers entre eux 2 à 2, donc $d = 1$. De plus, il nous fait gagner une étape car il nous renvois l'inverse de a modulo b.

$x_0 \leftarrow 1$;
$x_1 \leftarrow 0$;
$y_0 \leftarrow 0$;
$y_1 \leftarrow 1$;
$s \leftarrow 1$;
$d \leftarrow b$;
while $b \neq 0$ **do**
    $(q, r) \leftarrow divmod(a, b)$ /* *q est le quotient et r le reste de la division euclidienne de a par b* */ ;
    $(a, b) \leftarrow (b, r)$;
    $(x, y) \leftarrow (x_1, y_1)$;
    $(r_1, y_1) \leftarrow (q \times s1 + x_0, q \times y_1 + y_0)$;
    $(x_0, y_0) \leftarrow (x, y)$;
    $s \leftarrow -s$;
end
return $s \times x_0 + ((1 - s) \div 2) \times d$

Chapitre 2

Presentation du problème

2.1 Le probléme et ces restrictions

Le théorème des restes chinois induit une bijection :

$[0, N-1] \to Z/p_1Z$ ×...× Z/p$_iZ$
$x \to (x_1, ..., x_i)$
Oû $x \equiv x_i \; (mod\, p_i)$

Malheuresement nous ne pouvons encodé l'ensemble $[0, N-1]$ dans son entièreté et être capable de corriger et faire face au erreur.
A propos du N à choisir ce qui nous intéresse dans le N choisi c'est un N entier produit de plusieurs nombres premiers.
Car plus il y à de nombre premier qui compose N plus le systéme des restes chinois engendré par ce N contient d'équation.
Tout les nombres de 0 à N-1 ne peuvent pas être "encodé" si l'on veut corrigé des erreurs.
Il faut determiné une borne B comprise entre 0 et N-1 qui nous servira de "zone" d'encodage.
Pour determiné cette borne on a 2 maniéres
La premiére est de choisir N et de trouvé B en se restreingnant à une sous partie du systéme le surplus servant de "données de parité"
La deuxième est de choisir B l'ensemble des entiers que l'ont veut encodé et rajouté des informations donc des equation suplémentaire au systéme engendré par B donc multiplier B par d'autre nombre premier.

2.2 Comment déterminer la borne

Maintenant comment déterminer B de façon calculatoire ?

inserer explication du prof sur la borne.

Chapitre 3

Brute force

Il existe plusieurs algorithmes de brut force qui permettent de résoudre les erreurs tel que essayer tout les cas possibles. Mais cet algorithme est trop lourd à écrire. Nous avons donc fait des algorithmes moins lourd. Nous vous présenterons deux algorithmes dans cette section.

3.1 Notre premier algorithme version naïve

Tout d'abord, nous avons fait une fonction qui nous montre s'il y a une erreur dans nos listes, car avant de vouloir trouver les erreurs, il faut savoir s'il y en a une. Nous avons donc fait un petit algorithme qui permet d'enlèver alternativement un élément des listes. Autrement dit, on retire le premier élément des deux listes, puis on retire que le deuxième élément, puis que le troisième et ainsi de suite. Nous appliquons alors le théorème des restes chinois à chaque itération. Puis on mets le résultat dans une liste auxiliaire. S'il n'y a pas d'erreur, lorsqu'on retourne cette liste auxiliaire, on devrait avoir la même valeur partout, en revanche si les valeurs de la liste sont différentes, nous avons une erreur.

Soit $P = [p_1, p_2, ..., p_k]$ et $N = [n_1, n_2, ..., n_k]$
On note $P_i = [p_1, ..., p_i - 1, p_i + 1, p_k]$ et $N_i = [n_1, ..., n_i - 1, n_i + 1, ..., n_k]$
On note x_i le résultat théorème des restes chinois appliqué à P_i et N_i
Puis on retourne $L = [x_1, ..., x_k]$
Si $x_1 = x_2 = ... = x_k$ alors il n'y a pas d'erreur.

Voici un petit exemple pour bien comprendre cette fonction.
On pose P=[2,3,5,7,11,13] et N=[1, 1, 1, 5, 6, 9]
Si on applique le théorème des restes chinois à ces deux listes, on trouve 510510. Pour vérifier qu'il n'y ai pas d'erreur, on applique le théorème à P1=[3,5,7,11,13] et N1=[1, 1, 5, 6, 9, puis à P2=[2,5,7,11,13] et N2=[1, 1, 5, 6, 9] jusqu'à P6=[2,3,5,7,11,] et N7=[1, 1, 1, 5, 6]

Comme il n'y a pas d'erreur, la fonction va retourner [61, 61, 61, 61, 61, 61]
À présent, nous mettons une erreur dans les listes, on a P=[2,3,5,7,11,13] et N=[1,1,2,5,6,9]. En appliquant la fonction, il sera retourner une liste [6067, 6067, 61, 1777, 607, 1447], où l'on remarque bien que chaque élément est différrent, donc nous avons au moins une erreur. Dans cette fonction, on parcours n fois la liste de n élément donc nous somme en $O(n^2)$. Mais on fait n fois l'algo des restes chinois donc on est en $O(n^2)$. Cette algorithme est donc en $O(n^4)$ avec n la taille de la liste.

Pour créer notre première fonction de correction, nous nous sommes inspirées de la fonction ci dessus. On sait dit que si on enlevé 2 élément des listes, nous pourrions trouver l'erreur. On s'explique ; On a vu que si on enlevais 1 éléments des listes, le théorème des restes chinois marchait encore, et si on enlève 2 éléments, il marche encore. Donc, en enlevant le premier élément et tout les autres chacun leurs tours, et en appliquantle théorème à chaque fois, on retourne une liste. Puis, on réitére en enlevant le deuxième élément, et les autres chacun leurs tours, en re-appliquant les restes chinois à chaque fois, en mettant les résultats dans une nouvelle liste. Ainsi de suite jusqu'au dernier élément. Cette fonction retourne une liste de liste. Donc la liste où se trouve le même nombre partout, c'est que l'erreur est à la meme position. faisont plutôt exemple.
Soit $P = [p_1, p_2, ..., p_k]$ et $N = [n_1, n_2, ..., n_k]$
On note $P_ij = [p_1, ..., p_i - 1, p_i + 1, ..., p_j - 1, p_j + 1, ..., p_k]$ et $N_ij = [n_1, ..., n_i - 1, n_i + 1, ..., n_j - 1, n_j + 1, ..., n_k]$
On note x_ij le résultat du théorème des restes chinois appliqués à P_ij et N_ij
Et on retourne $L = [[x_11, x_12, ...x_1k], [x_21, x_22, ..., x_2k], ..., [x_{1k}, x_{2k}, ..., x_{kk}]]$
On a retourné une liste de liste. Si la k-ième sous-liste retourne toujours le même nombre alors cela signifie qu'il y a une erreur sur le k-ième élément.
‘

Si on récupère l'exemple du paragraphe précédent, où l'on avait découvert une erreur.
On a : $P = [2, 3, 5, 7, 11, 13]$ et $N = [1, 1, 2, 5, 6, 9]$
On applique notre algoritme et il retourne/ [[6067, 1062, 61, 1777, 607, 292], [1062, 6067, 61, 347, 607, 677], [61, 61, 61, 61, 61, 61], [1777, 347, 61, 1777, 217, 127], [607, 607, 61, 217, 607, 187], [292, 677, 61, 127, 187, 1447]]
La première liste retourné ne renvois pas le meme résultat donc l'erreur n'est pas sur la première équation. Elle n'es pas non plus sur la deuxième équation. Cependant, on peut voir sur la troisième liste que le résultat est toujours le même ; c'est à dire qu'à chaque fois on a fait le théorème des restes chinois en enlevant la troisième équation et une autre, donc nous avons une erreur sur la toisième équation.
Cette fonction est très performante comme vous avez pu le voir sur les exemples, cependant il y a quelque défaut. Premièrement, cette fonction ne marche que avec une seule erreur, on peut facilement changer la fonction

pour trouver n erreur juste en rajoutant des boucles FOR, mais cela rendrait l'algorithme encore plus lourd. Deuxièmement, nous avons un problème de borne. Pour trouver la borne, il suffit de multiplier par les 2 derniers nombre premier de la liste.
Par exemple pour le nombre 30030, la borne est 30030/(11x13) soit 210, se qui est très peu.
Enfin, nous avons un problème de complexité soit $O(n^3)$ et a chaque fois on fait le théorème des restes chinois qui est en $O(n^2)$, se qui donne un algorithme en $O(n^5)$.

3.2 brute force de hamming

Notre deuxième algorithme brut force consiste à utilisé la distance de hamming. Soit $x = x_1x_2...x_n$ et $y = y_1y_2...y_n$, la distance de Hamming c'est $d_H(x,y) = Card\{i|x_i \neq y_i\}$. Par exemple la distance de Hamming de $x = 1,2,3,4$ et $y = 1,2,5,6$ est 2
Avant de voir cette algorithme, nous devons expliquer la distance minimale. Soit G un groupe n-uplet appartenant à A^n. La distance minimale de G est la plus petite distance de hamming sur tout les éléments de G, soit $d(G) = min\{d_h(x,y)\}\,|x \neq y, x, y \in G$. D'après Hamming, si la distance minimale est égal à d, alors on peut détecter d-1 erreurs et corriger $\frac{d-1}{2}$ erreurs.

Dans cette algoritme nous utilisons la distance de haming. Nous laissons le choix au lecteur d'aller voir l'algoritme de la distance de hamming en annexe.
Pour ce nouvel algoritme, nous allons calculer la distance de Hamming entre le uplet à corriger et tout les élements de 0 à la borne. Nous selectionnons tout les éléments de distance de Hamming strictement inférieur à 3 pour pouvoir détecter une erreur.
Pour choisir la borne de cette algorithme pour 1 erreur, on retire les 3 plus grandes équations, c'est à dire si $P = [p_1, ..., p_k]$ alors la borne est $\prod_{i=1}^{k-3} p_i$. Pour e erreurs, on retire les $3e$ dernières équations.
Faisons un exemple pour illustrer cette algoritme :
Prenons les listes $P = [2,3,5,7,11,13,17]$ et $N = [1,0,0,5,9,10,7]$ pour nos exemples.
Détaillons, pour mieux le comprendre, notre algorithme sur $N = [1,0,0,5,7,10,7]$ c'est à dire que nous mettons une erreur sur le cinquième thermes. Dans un premier temps, l'algorithme va faire la distance de Hamming entre la liste N et le nombre 0 qui s'écrit $L_O = [0,0,0,0,0,0,0]$ car le reste dans division euclidienne des p_i par 0 est toujours 0. La distance de hamming de ces 2 listes est 5, car 2 éléments sur les 7 sont identiques entre les 2 listes, donc on ne prete pas attention à ce cas.
On fait la distance de hamming entre le liste N et 1 qui s'écrit $L_1 = [1,1,1,1,1,1,1]$ car le reste dans la division euclidienne des p_i par 1 est toujours 1. La distance de Hamming de ces 2 listes est 6 donc on ne prete pas attention. On fais la distance de Hamming entre la liste N et 2 qui s'écrit $L_2 = [0,2,2,2,2,2,2]$. La distance de Hamming de ces 2 listes est 7 donc on ne prete pas attention.
Il continue ses recherches sur les entiers 3, 4, et ainsi de suite. l'algorithme à découvert une liste qui avait une distance de Hamming infèrieur à 3. Cette liste est la liste $L = [1,0,0,5,9,10,7]$ qui correspond au nombre 75. Donc 75 est un candidat possible à la correspondance de cette uplet. l'algorithme va continuer ces tests sur tout les nombres jusqu'à notre borne moins 1 qui ici vaut $2 \times 3 \times 5 \times 7 - 1 = 209$.

Chapitre 4

Les fractions continues

4.1 Principe

Comme vu précédement la méthode brute force est efficace sur des petits cas de correction mais sur de cas plus grand la complexité explose et il est très difficile d'obtenir des résultats en un temps raisonnable.
Il faut donc trouver un moyen plus rapide et optimal. Donc comprenons comment intervient l'erreur sur le n-uplet du systéme d'équation.
Soit m le message envoyé et y le message et l'erreur dans le même message. si l'on fait la différrence de m-y on obtient e l'erreur. Maintenant cette erreur ce représente par un uplet en supposant par soucis de représentation que les erreurs se trouve au début de l'uplet associé $e = [e1, e2, e3, ..., e_k, 0, ..., 0]$ cette erreur est donc un multiple de tout les p_i nombres premiers pour i>k et k reste non nul sur les positions erronés.
Donc $y = m + e$ donc si l'on divise y par N notre entier on a $\frac{y}{N} = \frac{m}{N} + \frac{e}{N}$ mais $\frac{e}{N}$ peut se simplifier car hormis les positions érronés 2 dans notre cas e est un multiple de tout les autres p_i qui composent N
Ainsi on a $\frac{y}{N} = \frac{m}{N} + \frac{e}{N} = \frac{m}{N} + \frac{e_1 * e_2}{p_1 * p_2} < \frac{B}{N}$ B étant la borne choisis.
Finalement $\frac{m}{N} = \frac{y}{N} - \frac{e_1 * e_2}{p_1 * p_2}$

Or souvenons-nous que si z est un réel positif et que la fraction rationnelle $\frac{p}{q}$ vérifie $\left|z - \frac{p}{q}\right| < \frac{1}{2q^2}$. Alors $\frac{p}{q}$ est une fraction reduité de z.

Appliquer à notre probléme chaque réduite de cette fraction liste au dénominateur des candidats potentiels pour les positions érronés.

Maintenant parlons de la complexité de cette méthode. On parcours la liste des derniers nombres premiers du systéme, pour calculer une borne à ne pas dépasser au denominateur de la fraction cela reduit drastiquement les itérations à faire. Une fois cette borne determiné on calcule les reduites de $\frac{y}{n}$

sans que le denominateur qui sert d'indication sur la position des erreurs ne depasse la borne. A chaque tour de boucle on calcule une reduite qui coute l'ordre de la réduite fois le coût d'une division euclidienne , l'algorithme permettant de calculé une réduite étant une version modifié de l'algorithme d'Euclide arrété à l'ordre de la fraction réduite voulue. Donc finalement pour calculer toutes les réduites qui mettent en évidence les potentiels position érronés le coût et relativement faible car dépend de la taille du systéme lineairement donc relativement faible et rapide. Coût que l'on a constater grace a nos experiences et test.

4.2 Résumé d'experiences et test

Au début de nos tests nous avons pratiqué avec des petits cas de 6 à 7 nombre premier et cette méthode se révéla très efficace. Mais pour tester des cas avec plus d'erreurs nous avons eu besoin de systéme plus grand nous avons donc directement étendu notre systéme à 51 nombres premiers. Nous avons pris la borne B = 10^{29} ordre de grandeur de la borne théorique pour corriger 15 erreurs. Dans ce cas et grâce à la méthode des fractions continue nous avons été en capacité de corriger jusqu'à 7 erreur.

Annexe A

théorème des restes