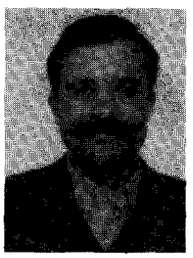
Complexité de systèmes de correction d'erreurs

Complexity of error correction devices



Philippe GODLEWSKI

ENST, Département Systèmes et Communications, 46, rue Barrault, 75013 PARIS

Enseignant-chercheur au département Systèmes et Communications de l'E.N.S. Télécommunications, Domaines d'enseignement : communications numériques, théorie de l'information. Domaines de recherches : codes correcteurs d'erreurs, cryptographie.

RÉSUMÉ

Apprécier la complexité de la mise en œuvre d'un système de codage/décodage est une question délicate qui ne reçoit généralement pas de réponse unique. On peut sommairement distinguer :

– une complexité algorithmique;

– une complexité matérielle qui correspond au nombre de portes logiques et de mémoires binaires que comprend un codeur ou un décodeur.

Pour la première notion et dans le cas des codes en blocs, il s'agit principalement d'une complexité arithmétique. Nous passons en revue les principales estimations existantes, « pratiques » ou asymptotiques, en tentant de distinguer les opérations sur un corps extension de celles sur le corps de base. Nous illustrons la seconde notion sur un exemple qui met en valeur les limitations de certaines évaluations.

MOTS CLÉS

Codes correcteurs d'erreurs, complexité.

SUMMARY

Estimating the complexity of implementation for a coding/decoding system is a delicate question. There is usually no unique answer. We can make a rought distinction between:

– algorithmic complexity;

– hardware complexity, corresponding to the number of logical gates and binary memories contained in a coding or decoding device.

For the first notion in the case of block codes, one deals mostly with arithmetic complexity. We survey the main existing estimations "practical" and asymptotical while trying to distinguish between the operations in an extension field and those on the ground field. We illustrate the second notion with an example showing the limitations of some evaluations.

KEY WORDS

Error-correcting-codes, complexity.

TABLE DES MATIÈRES

Introduction

Apprécier la complexité de la mise en œuvre d'un système de codage/décodage est une question délicate qui ne reçoit généralement pas de réponse unique. On peut sommairement distinguer :

– une complexité algorithmique (qui est ici principalement arithmétique);

– une complexité matérielle qui correspond au nombre de portes logiques et de mémoires binaires que comprend un codeur ou un décodeur.

Nous précisons dans le cas des codes en blocs, la première notion et nous illustrons rapidement la seconde. Le cas des codes convolutionnels sera évoqué à la fin de cet article.

Dans ce qui suit on considère un (n, k) code en blocs C, de distance minimale $d=2\,t+1$ ou $2\,t+2$ défini sur un corps F à q éléments.

1. La notion de complexité d'un algorithme [1]

Pour résoudre un problème ou accomplir une tache (codage, décodage), on a souvent le choix entre divers algorithmes. La complexité (en temps) k_x d'un algorithme correspond au nombre maximal d'instructions élémentaires (ou de cycles machine) que son déroulement nécessite. Il suffit très souvent de ne prendre en compte, dans les algorithmes de type algébrique que l'on rencontre en codage, que les instructions les plus complexes (multiplications, divisions) et de négliger les plus simples (chargements des registres, mises en mémoire, tests et sauts, ...). De cette manière, on ne dénombre que les opérations arithmétiques : on parle alors de complexité arithmétique. Il est d'usage de donner le comportement asymptotique de k_x, on écrit alors:

$$k_x \sim O\ (f(n, k, t)),$$

ce qui signifie qu'il existe une constante c_0 telle que :

$$k_x \leqq c_0\, f(n, k, t).$$

Nous donnons dans ce qui suit des ordres de grandeur « pratiques » de k_x. Ce qui signifie que les évaluations faites ne correspondent pas toujours aux algorithmes asymptotiquement les plus performants. Ces derniers sont en effet souvent sans intérêt pour les longueurs n considérées en pratique. On retiendra aussi que la constante c_0 est voisine de 1 et bornée par 10 dans la plupart des estimations fournies.

L'espace mémoire nécessaire à l'exécution de l'algorithme conditionne la complexité en mémoire. Des compromis temps/mémoire sont toujours envisageables. En particulier un algorithme de décodage qui consiste à consulter une liste de tous les mots reçus possibles (ou respectivement de tous les syndromes) a une complexité en temps négligeable mais une complexité en mémoire exponentielle (en $n.q^n$ ou respectivement en $n.q^{n-k}$). On se place ci-après dans la situation où la mémoire nécessaire à l'exécution de l'algorithme est, au plus, du même ordre que k_x. La complexité arithmétique s'apparente alors à la longueur d'un programme sans boucle (« straight line complexity », [1 et 2]).

2. Calculs sur un corps extension

Il est important de noter que les calculs nécessaires au codage et au décodage peuvent être effectués soit sur le corps de base $F=F_q$ soit sur une extension F′ de F qui comporte q^m éléments. Pour les codes de Reed-Solomon $m=1$, pour des codes (BCH, Goppa, ...) construits en longueur primitive $n=q^m-1$ ou $n=q^m$. On utilisera la notation $O\,(.)$ [respectivement $O_{ex}\,(.)$] si les calculs sont considérés sur F (resp. sur F′). Quand on dispose d'une unité arithmétique qui effectue les quatre opérations sur F, les calculs sur F′ se ramènent à des calculs polynomiaux sur F. De cette manière l'addition sur F′ nécessite m additions sur F alors que l'algorithme naïf de multiplication demande de l'ordre de m^2 additions et multiplications sur le même corps [un algorithme moins naïf en

demande $O\,(m.\log(m))$]. Ainsi dans un corps extension F′, la multiplication est « plus complexe » que l'addition.

Les algorithmes de codage et de décodage utilisés comportent principalement des additions-soustractions et des multiplications (peu de divisions), des deux catégories d'opérations intervenant dans des proportions comparables.

Ces dernières considérations amènent à ne comptabiliser dans l'évaluation de la complexité, que les multiplications et les divisions (sauf dans le corps F_2 où on ne prend compte que les additions). Pour fixer les idées on pourra avoir en mémoire les correspondances « pratiques » : $O(f) \sim m^2 . O_{ex}(f)$ et « asymptotiques » : $O(f) \sim m.\log(m) . O_{ex}(f)$ qui relient les complexités de calcul sur F et sur F′, son extension de degré m.

Remarques pratiques :

– La règle couramment admise : « la multiplication est plus complexe que l'addition » n'est plus valable dans les corps d'entiers (F_p avec p premier). Les deux opérations y sont d'une difficulté d'exécution analogue : la multiplication modulo p correspond à l'addition modulo $p-1$ qui est de complexité voisine de celle de l'addition modulo p. Le corps F_2 représente la seule exception nette à cette règle, le « et logique » étant reconnu moins complexe que le « ou exclusif ».

– L'addition est particulièrement simple dans les corps de caractéristique 2. Elle correspond alors au « ou exclusif bit à bit » (instruction IEXOR du Fortran ou du Basic).

– Les correspondances précédentes entre $O(.)$ et $O_{ex}(.)$ ne sont pas pertinentes lorsqu'on utilise des tables de logarithme et d'exponentielle :

$$\alpha^i \longrightarrow i = \log_\alpha(\alpha^i),$$
$$i \longrightarrow \alpha^i$$

ou encore une table de log de Zech Z (i) défini par l'équation :

$$1 + \alpha^i = \alpha^{Z(i)}.$$

Ceci correspond à un compromis temps/mémoire qui est envisageable dans les structures microprogrammées si $|F'| = q^m < 2^{16}$. De cette manière, sur de tels corps extension F′ de caractéristique $p > 2$ la multiplication apparaît à l'ingénieur plus simple que l'addition.

3. Complexité du codage

La complexité du codage est de l'ordre de $O\,(k.(n-k))$ opérations sur F pour un code linéaire. Si un algorithme rapide de transformation de Fourier discrète est utilisable, on obtient une évaluation en $O_{ex}(n.\log(n))$. Notons que dans le cas des codes cycliques les calculs peuvent être très simplement sérialisés : le codeur ne comporte qu'un registre à décalage rebouclé suivant le polynôme générateur de degré $(n-k)$ et fournit la redondance en k coups d'horloge. Dans le cas des codes de Reed-Solomon, des simplifications supplémentaires ont été apportées par Berlekamp [3]. On retiendra que dans la mise en œuvre d'un système de codage/décodage, le codage correspond à la partie facile.

4. Les principales techniques de décodage après décision ferme

On distingue habituellement :

– Le *décodage* après décision *ferme* (hard decision decoding) qui traite des données sévèrement quantifiées : le décodeur ne dispose que de symboles à valeurs dans F_q.

– Le *décodage pondéré* (soft decision decoding) qui prend en compte une indication supplémentaire sur la vraisemblance ou la fiabilité de chacun des symboles reçus. Cette information est disponible à la sortie de certains démodulateurs.

La correction conjointe d'erreurs et d'effacements est un cas intermédiaire.

Nous donnons maintenant une liste des principales techniques de décodage ferme :

– Le décodage par recherche exhaustive (EXH) ou par filtrage adapté. On compare le mot reçu avec tous les q^k mots du code. Ceci n'est donc envisageable que si le nombre de mots de code est limité. En comparant directement le signal reçu aux différents signaux possibles, on obtient la version « pondérée » de l'algorithme.

– Le décodage par calcul du syndrome et recherche dans une table (TAB). Cette méthode est préférable à la précédente lorsque $k/n > 1/2$. Elle est actuellement envisageable si $q^{n-k} < 10^5$.

– Le décodage par piégeage d'erreurs (PIEG), en anglais : « error trapping ». Il utilise souvent la structure de Meggitt.

– Le décodage majoritaire (MAJ) est associé aussi en pratique avec la structure de Meggitt.

– Le décodage par permutations (PERM) n'est adapté que pour certains codes de longueur modérée ayant un groupe d'automorphisme suffisamment riche.

5. Complexité du décodage

La question générale du décodage est liée à un certain nombre de problèmes dits « NP-complets » et réputés intractables. Plus précisément, si l'on se donne un

TABLEAU

Complexité du décodage algébrique.

	Algorithme d'intérêt pratique	Algorithme asymptotiquement plus performant
Calcul du syndrome (valeur du polynome reçu en r points)	$n.r$	$O_{ex}(n.\log n)$
Algorithme de Berlekamp (ou « PGCD rapide »).	$(3/2)\ r^2$	$O_{ex}(r.\log^2(r))$
Localisation des erreurs.	$n.r/2$	$O_{ex}(n.\log(n))$
Évaluation des erreurs (cas non binaire)	r^2	$O_{ex}(n.\log(n))$
Total . . . . . (pour une capacité de correction r/n bornée inférieurement par λ)	$(3/2)\ n.r+(5/2)\ r^2$ $\geqq \lambda\ O_{ex}(n^2)$	$O_{ex}(n.\log^2(n))$

n désigne la longueur du code considéré, $r=d_{BCH}-1$, pour un code de Reed-Solomon $r=n-k$.

code linéaire (n, k) arbitraire, un mot y de longueur n et un entier w quelconques, le problème de déterminer s'il existe ou non un mot du code à une distance inférieure ou égale à w de y est NP-complet [4]. La complexité d'un algorithme permettant de le résoudre semble donc exponentielle.

Sur un plan plus pratique, la plupart des algorithmes de décodage proposés nécessitent au moins $O(\inf(q^k, q^{n-k}))$ opérations (*cf.* EXH-TAB, MEG, ...). Leur complexité demeure donc exponentielle, même s'ils s'adressent à des classes restreintes de codes. L'exception notable est celle des algorithmes de décodage algébrique qui ont une complexité majorée par $O_{ex}(n.t)$ opérations sur un corps extension, où $t=[d_{BCH}-1]/2$ (*cf.* tableau). Si l'on utilise un algorithme rapide de calcul du PGCD de deux polynômes ([1], p. 308) on obtient une évaluation en $O_{ex}(n.\log^2(n))$, mais l'algorithme déduit est peu performant en regard de l'algorithme d'Euclide et surtout de l'algorithme de Berlekamp-Massey pour les longueurs n utilisées ($n<10^6$, *cf.* [5], p. 369).

6. Complexité matérielle

La complexité algorithmique donne des ordres de grandeur mais ne fournit pas de mesure fine de la complexité de réalisation matérielle. Les estimations du type de celles qui viennent d'être données sont souvent approximatives. On n'y distingue pas, en général, les multiplications où l'une des opérandes est une constante de celles où les deux opérandes sont des variables; on néglige aussi les transferts de données entre mémoires. De cette manière les structures microprogrammées sont avantagées, bien qu'elles soient peu adaptées à la plupart des algorithmes de codage et de décodage.

Ces derniers mettent souvent en œuvre des calculs polynomiaux et la structure naturelle pour les effectuer est celle des circuits linéaires séquentiels. Les circuits peuvent être réalisés sous forme cablée ou sous forme de composants intégrés (VLSI) qui fonctionnent sur un mode systolique : chaque porte logique y est utilisée d'une façon presque permanente (c'est-à-dire à chaque coup d'horloge).

Le décodeur de Meggitt comme exemple de décodage algébrique (*fig.*).

Considérons, pour fixer les idées, le décodeur de Meggitt, dans le cas d'un code de Hamming binaire de longueur 127 engendré par $g(x)=x^7+x+1$. Il comporte un registre tampon de longueur $n=127$, un registre rebouclé de longueur $n-k=7$, une simple porte « et » à $n-k$ entrées. Ce décodeur peut être vu comme une réalisation matérielle de l'algorithme de décodage algébrique. Le registre à décalage rebouclé qui permet le calcul du syndrome apparaît alors comme un circuit multiplicateur par un élément β du corps extension $F'=F_{128}$, racine du polynôme $g(x)$: Le contenu de ses mémoires représente la décomposition d'un élément de F' qui est un F_2-espace vectoriel sur la base $(1, \beta, \beta^2, \ldots, \beta^6)$, le rebouclage représente la relation $g(\beta)=0$, c'est-à-dire $\beta^7=\beta+1$. On remarque ainsi que cette multiplication est plus simple que l'addition dans ce même corps puisqu'elle ne nécessite qu'un « ou exclusif » et un décalage circulaire. On suppose maintenant que le mot reçu est $b(x)=c(x)+x^i$, $b(x)$ correspond au mot de code $c(x)=a(x).g(x)$ perturbé par une erreur additive $e(x)=x^i$. Les éléments binaires $b_{n-1}, b_{n-2}, \ldots, b_0$ se présentent l'un après l'autre à l'entrée du décodeur. Lorsque les n symboles $(b_0, \ldots, b_{n-1})$ sont présents dans le registre tampon, le registre du syndrome contient $b(\beta)=\beta^i$.

Si $i=n-1$, $\beta^i=\beta^{-1}$ correspond au vecteur (1000001) sur la base considérée. La correction de b_{n-1} est alors activée au coup d'horloge suivant que l'on numérote par 1, grâce au décodage effectué par la porte « et ».

Si $n=n-s$, le registre du syndrome contient au coup d'horloge numéro $s-1$, $\beta^{n-s}\ \beta^{s-1}=\beta^{-1}$. La correction de b_{n-s}, présent alors à l'extrémité du registre tampon, est activée au coup d'horloge suivant (n^0 s).

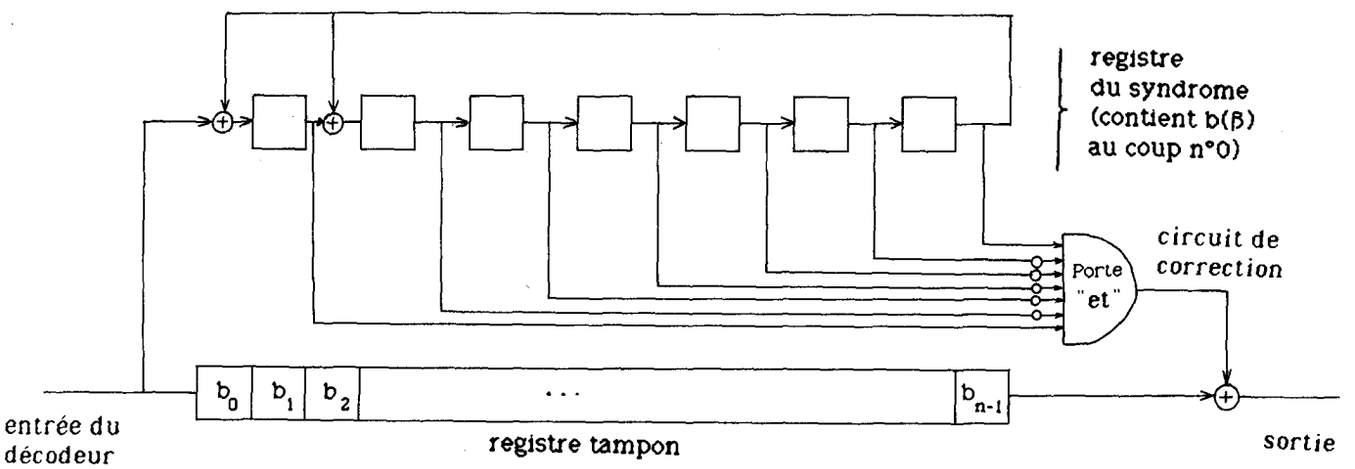
Le décodeur comporte :

$n+n-k$ mémoires binaires (bascules D);

j portes « ou exclusif » à deux entrées avec $2<j<n-k$;

$n-k$ portes « et » à deux entrées pour réaliser le « et » global.

Sa complexité matérielle, c'est-à-dire le nombre de composants logiques élémentaires qu'il comporte (hors registre tampon) est en $O(n-k)$. Chaque composant étant utilisé pendant, soit n, soit $2n$ coups d'horloge, on obtient une complexité algorithmique totale en n^2. Mais cette dernière évaluation ne fait apparaître la simplicité du décodeur.



Décodeur de Meggitt (le contenu des registres correspond au coup d'horloge n° 0).

Remarque : Ce décodeur peut être vu comme la matérialisation de deux types d'algorithmes, qui, dans le cas général, sont bien différents :

– le décodage algébrique, l'expression de la complexité $O_{ex}(nt)$ devient $O(n.(n-k))$ puisqu'ici $t=1$ et le degré de l'extension m est égal à $n-k$;

– le décodage par syndrome dont la complexité est en $O((n-k)\,2^{n-k})$ mais ici $n=2^{n-k}$.

Cet exemple doit inciter à la prudence celui qui cherche à savoir si la complexité d'un type (trop restreint) d'algorithmes est exponentielle ou polynomiale.

7. Algorithmes de décodage pondéré

Les algorithmes de décodage pondéré peuvent être divisés en deux catégories.

(i) Les algorithmes optimaux (à vraisemblance maximale). L'algorithme de Viterbi est le reprédentant le plus populaire de cette classe. Il demande environ $O(2^{K-1}\,2^{k-\log k})$ comparaisons par bit d'information décodé où K est la longueur de contrainte (ou la mémoire minimale) d'un (n, k) code convolutionnel binaire.

(ii) Les algorithmes sous optimaux (algorithmes séquentiels, de Chase, ...) qui sont le plus souvent des heuristiques. L'effort de calcul qu'ils nécessitent dépend alors du rapport signal à bruit qui caractérise le canal de transmission. Citons le décodage « GMD » de Forney [6] qui met en jeu $d-1$ décodages algébriques et dont la complexité reste polynomiale (en $O_{ex}(n^3)$).

Conclusion

Évaluer la complexité d'un algorithme de décodage est un travail comptable peu sophistiqué mais parfois fastidieux. Les évaluations ne sont souvent pas très significatives quand elles s'intéressent à un code particulier. Elles fournissent cependant des ordres de grandeur qui permettent d'extrapoler les performances à des classes infinies de codes. Dans une perspective de réalisation de décodeur sur des circuits intégrés, un problème d'actualité est d'implanter un algorithme de complexité en $O(f(n, \ldots))$ de telle manière que le circuit correspondant ait une complexité matérielle en $O(f(n, \ldots)/n)$.

BIBLIOGRAPHIE

[1] A. V. Aho, J. E. Hopcroft et J. D. Hullman, *The Design and Analysis of Computer Algorithms*, Addison-Westley, Reading Mass., 1975.

[2] A. Lempel, G. Seroussi et J. Ziv, On the Power of Staight-Line Computations in Finite Fields, *IEEE Trans. Inform. Theory*, IT-28, n° 6, November 1982, p. 875-880.

[3] E. R. Berlekamp, Bit-Serial Reed-Solomon Encoders, *IEEE Trans. Inform. Theory*, IT-28, n° 6, November 1982, p. 869-867.

[4] E. R. Berlekamp, R. J. McEliece et H. C. A. Van Tilborg, On the Inherent Intractability of certain Coding Problems, *IEEE Trans. Inform. Theory*, IT-00, n° 3, May 1978, p. 384-386.

[5] F. J. MacWilliams et N. J. A. Sloane, *The Theory of Error-Correcting Codes*, Amsterdam, North-Holland, 1977.

[6] G. D. Forney, Generalized Minimum Distance Decoding, *IEEE Trans. Inform. Theory*, IT-12, April 1966, p. 125-131.