Les Cahiers de l'Analyse des Données
Vol. XXII - 1997/8 - n°4 - pp.455-460.

PRINCIPES DE CODES GÉNÉRAUX POUR LES FICHIERS D'OCTETS

[CODES OCT.]

J.-P. BENZÉCRI
*A. SKALLI**

Les codes utilisés aujourd'hui perpétuent une longue tradition de chiffres. Mais l'avènement du calcul électronique, tout en fournissant au chiffrage et au déchiffrage un outil d'une merveilleuse vélocité, multiplie en nombre et variété les documents qu'on s'applique à faire circuler à l'abri de toute indiscrétion. Des conceptions nouvelles sont nées, sous l'égide de la haute arithmétique et de la théorie de l'information. Nous proposons ici un kaléidoscope d'un autre genre, imaginé en observant le jeu capricieux des microprocesseurs.

Nous considérerons successivement les termes et les opérations dont nous paraît constitué un système de codage; puis décrirons le système proposé; et donnerons enfin notre procédé de génération pseudo-aléatoire de codes.

1 Termes et opérateurs

1.1 Fichiers

Il ne nous semble pas utile de distinguer les données à chiffrer selon leur contenu verbal ou numérique; ni d'envisager la nature des supports et des canaux; nous supposerons seulement que la donnée à coder est une suite de chiffres binaires; ou, plus précisément, compte tenu des procédés techniques que nous prendrons comme exemples, une suite d'octets. Nous dirons simplement que la donnée, ainsi que le message codé, sont des *fichiers d'octets*, ou, en bref, des *fichiers*.

1.2 Transformations

Il est commun d'utiliser comme base de codage une suite pseudo-aléatoire de nombres. Nous acceptons cette pratique, notre suggestion visant à créer des générateurs pseudo-aléatoires à la fois très simples et très divers.

Nous userons du terme de transformation pour désigner une opération notée *trans* qui, à partir d'un fichier donné *f*, produit un fichier trans(f) que

(*) Professeur, École Mohammadia d'Ingénieurs; Avenue Ibn Sina; BP 765; Rabat-Agdal; MAROC.

nous supposerons être de même longueur que le fichier initial, ou de longueur proportionnelle à celle de celui-ci. Pour remplir la fonction qui lui est assignée, une transformation devra avoir un caractère pseudo-aléatoire au sens suivant. Bien que le codage se fasse de façon déterminée, par blocs, par exemple par blocs de 4 octets, la correspondance entre bloc initial et bloc transformé ne devra paraître présenter aucun caractère de régularité, même si l'on considère plusieurs centaines ou milliers d'exemples.

1.3 Union

Nous appellerons union et noterons simplement: *uni*, une opération qui, à partir de deux fichiers: *f* et *g*, engendre un fichier: uni(f,g), de telle sorte que, si l'on connaît uni(f,g), on puisse retrouver les deux fichiers *f* et *g*. À titre d'exemple, le plus simple sera de supposer que les fichiers *f* et *g* ayant même longueur, la construction de uni se fait par blocs, disons de 4 octets; les octets de deux blocs de même rang de *f* et *g* étant permutés suivant une règle définie (mais arbitraire) pour créer un bloc de 8 octets de som(f,g). On notera:

f=sep1(uni(f,g) ; g=sep2(uni(f,g) ;

les opération de séparation servant à retrouver f et g quand est connue l'union de f et de g.

1.4 Somme

Nous appellerons somme et noterons: *som* une opération qui, étant donnés deux fichiers: *f* et *g*, construit un fichier not:é: som(f,g), de telle sorte que si l'on connaît som(f,g) et l'un des fichiers *f* et *g*, on puisse retouver l'autre. À titre d'exemple, le plus simple sera de supposer que, les fichiers *f* et *g* ayant même longueur, la construction de *som* se fait par blocs, disons de 4 octets; les blocs de même rang, interprétés comme désignant des entiers, étant additionnés, suivant les règles de l'arithmétique usuelle (des entiers modulo 2↑32) pour créer le bloc correspondant du fichier som. On notera:

g=diff(som(f,g),f) .

2 Le système de codage et de décodage

2.1 Fichier donné: fd, et base de code: fb

Le fichier donné sera désigné par fd; outre fd, nous utiliserons un second fichier fb, ou base de code, dont on peut supposer qu'il a même longueur que fd, quitte à couper fb s'il est trop long, ou à le répéter s'il est trop court.

Le choix du fichier fb ne sera soumis à aucune contrainte; en particulier, il n'aura pas à être connu du destinataire du message codé; il pourra avantageusement être changé à chaque émission, au caprice de l'émetteur; il faudra seulement éviter que fb ne donne au codage une régularité contraire au caractère pseudo-aléatoire cherché; mais les opérations effectuées assureront aisément ce caractère.

2.2 Génération du code intermédiaire: fc

Le code intermédiaire, ou fichier fc, sera engendré à partir de fb par une transformation, transc; laquelle n'aura pas à être connue du destinataire, mais sera à la discrétion de l'émetteur; et servira seulement à garantir au code fc le caractère pseudo-aléatoire qui peut manquer si l'on emploie telle quelle la base de code fb. On aura donc:

fc=transc(fb).

En toute rigueur, il n'est pas indispensable que fc diffère de fb.

2.3 Création du code: fk, et du message: fm

Le message fm sera créé par une transformation transk, une opération d'union uni et une opération d'addition som, lesquelles devront toutes trois être connues du destinataire mais pourront, comme de règle dans le codage, être modifiées suivant toute convention que l'on jugera opportune; la particularité du système étant que transk peut revêtir des formes à la fois très variées et engendrées à partir d'une description très compacte. On posera

fm=uni(fc,som(transk((fc),fd)) .

En d'autres termes, le codage proprement dit s'effectuera en faisant la somme du fichier donné: *fd* et d'une séquence pseudo-aléatoire: transk(fc); cette dernière sera transmise sous la forme du code intermédiaire: *fc* ayant servi à l'engendrer.

Il sera commode de noter: *fk* la séquence pseudo-aléatoire à laquelle est ici dévolu le rôle d'un code usuel:

fk=transk(fc).

2.4 Décodage

Le destinataire peut reconstituer fd à partir de fm; à condition qu'il soit à même d'effectuer les opérations sep1, sep2, transk et diff. De façon précise, on a la formule:

fd=diff(sep2(fm),transk(sep1(fm))).

En décomposant cette formule, on a:

sep1(fm)=fc ; sep2(fm)=som(trask(fc),fd)=som(fk,fd) ;

fk=transk(fc) ; fd=diff(fm,fk) .

En d'autres termes, après avoir séparé les deux fichiers fc et som(fk, fd) qui sont unis pour créer fm, le destinataire peut reconstituer la suite pseudo-aléatoire, ou code, fk; il obtient alors fd par différence.

3 Les générateurs fonctionnels de nombres aléatoires

En définitive, dans la construction très peu originale proposée ci-dessus comme un support à notre exposé, tout repose sur l'aptitude véritable des transformations: transc, ou: transk, à engendrer du pseudo-aléatoire avec des spécifications très diverses et très compactes. Au lieu de recourir à des théories mathématiques, nous utilisons directement les fonctions comme générateurs de nombres.

Partons, pour fixer les idées, du calcul tel qu'il est effectué sur Macintosh suivant les normes SANE. Toute suite de 4 octets représente un entier long; et il est facile, si on le désire, de choisir systématiquement un représentant positif ou nul modulo 2↑32, plutôt qu'un représentant avec signe. Ce nombre entier x peut être considéré comme un nombre réel et pris pour argument d'une fonction que nous noterons: rac, parce que la racine carrée nous paraît un bon exemple. Le nombre réel: rac(x), n'est lui-même, dans la mémoire de la machine, qu'une suite de 10 octets. De ces 10 octets, on peut en choisir 4, qui ne contribuent ni à l'exposant ni au signe, et les ranger dans un ordre quelconque pour créer un entier long y , qu'on notera: y = irac(x).

Pour transformer un fichier fx, considéré comme une suite {x(1), x(2),..., x(n), x(n+1)}, en un fichier fy = {y(1), y,2),...}, on posera:

```
y(1)=irac(x(1));
y(2)=irac(x(2));
.....
y(n)=irac(x(n)).
```

Il pourra être utile de se garantir contre la présence fréquente de certaines valeurs x en posant

```
y(n)=irac(y(n-1)+x(n)) .
```

Il nous semble possible de prendre pour fonction rac des expressions très diverses sans laisser d'être compactes telles que:

```
rac(x)=√(x+(1/100));
rac(x)=√(sin x+(3/10)); ...
```

4 Le programme 'codecod' de codage et décodage de fichiers d'octets

Le programme 'codecod' met en œuvre les principes proposés ci-dessus. Notre commentaire se bornera à des remarques et mises en garde.

La procédure: transk, utilisée aussi bien pour le codage que pour le décodage, comporte, comme paramètre, un entier: chif. Cet entier est spécifié par l'utilisateur qui demande le codage d'un fichier; et la valeur choisie doit

```
program codecod;
uses memtypes,quickdraw,osintf,toolintf,sane,uver;
type t10=packed array[1..10]of byte;p10=^t10;
var fb,fd,fm:file of integer;nomfb,nomfd,nomfm:string;
erl,i,ib,ic,ik,id,is,im,iz,chif:integer;rr:extended;
prr,prc,prk,prs,prm:p10;rpt,rpc,rpk:char;
procedure transc;begin rr:=ib+113;
  if (ib+113<0) then rr:=rr+32768;
  rr:=sqrt(rr+sqr(i/100));prc^[1]:=prr^[7];prc^[2]:=prr^[5] end;
procedure transk;begin rr:=ic+chif+0.1;
  if (ic+chif<0) then rr:=rr+32768;
  rr:=sqrt(rr);prk^[1]:=prr^[6];prk^[2]:=prr^[9] end;
begin benzecri;rpt:='O';
prr:=p10(@rr);prc:=p10(@ic);prk:=p10(@ik);prs:=p10(@is);prm:=p10(@im);
while not (rpt='N') do begin erl:=0;rpc:='N';rpk:='N';
  while not(rpk in ['C','D']) do begin
    write('faut-il coder(C) ou décoder(D) un fichier ');readln(rpk) end;
  if (rpk='C') then while not((rpc='O')or(erl=6)) do begin
    write('le nom du fichier donné fd à coder est ');readln(nomfd);
    rpc:=dialof(stringptr(@nomfd));if (rpc='O') then begin
      write('le nom du fichier base de code fb est: ');readln(nomfb);
      rpc:=dialof(stringptr(@nomfb)) end;
    if (nomfb=concat(nomfd,'*')) then begin
      writeln('ERREUR: fb ne peut avoir le nom du message à créer');
      rpc:='N' end;
    if (rpc='O') then begin
        write('le chiffre du code est: ');readln(chif);
        write('ces réponses sont-elles confirmées O ou N ');readln(rpc) end
      else erl:=erl+1 end;
  if ((rpc='O') and (rpk='C')) then begin
    if (nomfb=nomfd) then iz:=1 else iz:=0;
    if (iz=0) then reset(fb,nomfb);reset(fd,nomfd);
    nomfm:=concat(nomfd,'*');rewrite(fm,nomfm);i:=0;
    while not eof(fd) do begin read(fd,id);ib:=id;
      i:=i+1;if(i mod 10=0) then writeln(i div 10);
      if (iz=0) then begin
         if eof(fb) then reset(fb,nomfb);read(fb,ib) end;
      transc;transk;is:=id+ik;
      prm^[1]:=prc^[1];prm^[2]:=prs^[1];write(fm,im);
      prm^[1]:=prc^[2];prm^[2]:=prs^[2];write(fm,im) end;
    close(fb);close(fd);close(fm);writeln('fin du codage') end;
  if (rpk='D') then while not((rpc='O')or(erl=6)) do begin
    write('le nom du fichier donné fm à décoder est ');readln(nomfm);
    rpc:=dialof(stringptr(@nomfm));
    if (rpc='O') then begin
        write('le chiffre du code est: ');readln(chif);
        write('ces réponses sont-elles confirmées O ou N ');readln(rpc) end
      else erl:=erl+1 end;
  if ((rpc='O') and (rpk='D')) then begin
    reset(fm,nomfm);rewrite(fd,concat(nomfm,'-'));i:=0;
    while not eof(fm) do begin
      i:=i+1;if(i mod 10=0) then writeln(i div 10);
      read(fm,im);prc^[1]:=prm^[1];prs^[1]:=prm^[2];
      read(fm,im);prc^[2]:=prm^[1];prs^[2]:=prm^[2];
      transk;id:=is-ik;write(fd,id) end;
   close(fm);close(fd);writeln('fin du décodage') end;
  if (rpc='O') then begin
   write('faut il recommencer O ou N ');readln(rpt) end;end;end.
```

être donnée quand on demande le décodage.

L'on a parlé de fichier d'octets, sans en restreindre la nature: fichier en format de texte seul; texte formaté créé par un logiciel; graphique, d'un type ou d'un autre: simple damier de pixels en noir et blanc, ou instructions de tracé de lignes, etc.; et même, programme exécutable.

Toutefois, on dépend ici des contraintes du système et de celles des logiciels. Sur un Macintosh+ de ≈1988, le codage et décodage d'un exécutable se fait bien; et, de même, sur un IBM.PC de même époque; mais le système 6 de Macintosh, ne permet pas d'ouvrir en lecture un exécutable. Quant aux textes ou graphiques créés par des applications, 'codecod' les code et les décode sans rencontrer d'obstacle; mais beaucoup de logiciels n'acceptent pas d'ouvrir le fichier décodé; à moins que le type, voire le créateur adéquat n'ait été spécifié dans l'en-tête; (ce que permettent de faire divers logiciels).

L'espace de dix octets, occupé par le réel: rr, est lu comme suite d'octets par plusieurs pointeurs du type: p10. Le jeux des procédures: transk, et: transc, dépend essentiellement de la manière dont sont écrits les réels en virgule flottante. Il ne convient pas d'utiliser, pour le codage les octets représentant l'exposant; car ceux-ci n'ont pas la même amplitude de fluctuation que ceux de la mantisse. Bien plus, des coprocesseurs insèrent des octets fixes dans la représentation d'un réel... Et le système du Macintosh ne code pas les réels comme le fait celui d'un IBM.PC. Mais si l'on est averti de toutes ces particularités (que l'on peut découvrir par un programme conversationnel ad hoc, affichant à l'écran la suite des octets représentant des réels entrés au clavier), on parvient à décoder sur IBM un fichier codé sur MAcintosh; et réciproquement; (à condition de ne pas utiliser dans le codage les derniers chiffres de la mantisse; lesquels sont en butte à des erreurs d'arrondi gérées différemment par les divers processeurs).

Le programme peut être modifié; non seulement en introduisant, dans transc et transk, d'autres fonctions que sqrt; mais aussi en codant les octets par blocs de 4 plutôt que par blocs de 2; ce qui interdit de tabuler (sans dépense excessive) l'action du programme, même si l'on dispose de nombreux exemples de fichiers codés.

Nous ne savons rien des propriétés arithmétiques du processus pseudo-aléatoire utilisé ici: question en rapport avec le décryptage; et qui intéresse peut-être le mathématicien.