Codare universală.

În multe situații din domeniul comunicației/prelucrării datelor digitale, secvențele de date întâlnite prezintă anumite regularități structurale sau se supun unor anumite constrângeri. În acest context, dată fiind o sursă informațională problema compresiei implică mai întâi identificarea constrângerilor sursei informaționale şi apoi dezvoltarea unei scheme care să realizeze o compresie optimă relativ la un anumit criteriu. Odată identificați parametrii importanți ce descriu sursa informațională problema compresiei e redusă la dezvoltarea unui cod care să prezinte redundanță minimă.

Atunci când însă nu există cunoştințe apriorice despre despre caracteristicile sursei informaționale, problema compresiei datelor devine devine mult mai complexa. În aceste cazuri se utilizeaza scheme de **codare universală** în care procesul de codare este intercalat cu un proces de învățare a caracteristicilor variabile ale sursei. Astfel de tipuri de scheme de codare necesită în mod inevitabil un spațiu de memorie mai mare şi utilizează criterii potrivite unei categorii largi de surse informaționale.

Algoritmul Lempel-Ziv este unul dintre primii algoritmi de codare universală. Esența sistemului de codare este un „algoritm incremental de grupare”[1] care grupează şirul sursă într-o colecție de segmente a căror dimensiune creşte cu fiecare pas. Regula de generare a segmentelor este simplă: pornind de la segmentul vid, fiecare nou segment este obținut prin adăugarea unui simbol celui mai lung segment care prefixează segmentul obținut.

Exemplu:

Şirul 010100010 este grupat în colecția de segmente (0, 1, 01, 00, 010).

Atunci când segmentele formate sunt reținute în aceeaşi ordine în care au fost generate, fiecare segment poate fi codat sub forma unei perechi (i,y), unde reprezentarea binară a lui i oferă poziția celui mai lung segment prefix, iar y este simbolul care se adaugă segmentului de la poziția i pentru a se obține noul segment.

Exemplu:

[1] Incremental parsing algorithm.

Pentru exemplul de mai sus, cuvântul de cod corespunzător segmentului (010) este dat de perechea (3,0) - i.e. segmentului de pe poziția 3 i se adaugă simbolul 0.

Lungimea de cod a unui şir *s* este dată de:

$$L_{ZL}(s) = \sum_{j=1}^{n(s)} \lceil \log j \rceil + n(s), \tag{1}$$

unde *n(s)* reprezintă numărul de segmente în care a fost grupat şirul *s*, iar $\lceil x \rceil$ reprezintă cel mai mic număr întreg mai mare decât *x*.

Algoritmul propus de Ziv şi Lempel atinge asimptotic compresie optimă pentru orice sursă ergodică staționară. Complexitatea algoritmului, din punct de vedere al spațiului necesar gestionării arborelui depăşeşte însă orice limite.

O reinterpretare a algoritmului Lempel-Ziv devine posibilă prin utilizarea algoritmului lui Tunstall pentru generarea setului de segmente:

> *Se porneşte de la un arbore ce conține un nod şi doi fii. Fiecărui nod i se ataşează pondere. Pentru arborele inițial se plasează ponderea 2 în rădăcină şi 1 în fiecare frunză. Acest arbore se utilizează pentru generarea primului segment ca fiind calea de la rădăcină până la o frunză. În drumul către frunză se incrementează ponderea fiecărui nod vizitat.Ultima frunză vizitată, a cărei pondere devine 2, este împărțită în două noi noduri a căror pondere este inițializată cu 1.Noul arbore obținut este utilizat în generarea noului segment, iar ciclul este repetat.*

Exemplu

Secvența 010100010 este împărțită în colecția: (0, 1, 01, 00, 010), iar evoluția arborelui generat de algoritmul lui Tunstall este prezentată în figura 1.

Începând cu arborele inițial, algoritmul lui Tunstall generează la fiecare pas un arbore binar complet. Pentru şirul *s* având *n*(*s*) segmente notăm cu *T*(*s*) arborele binar generat de şirul *s*. Ponderea fiecărui nod este egală cu suma ponderilor nodurilor fiu. Împărțirea ponderilor fiecărui nod la ponderea nodului rădăcină reprezintă practic un mecanism de generare a probabilităților condiționate atât ale segmentului generat cât şi ale oricăror prefixuri ale acestuia. În plus, din construcția arborelui se respectă condiția de

compatibilitate *P(s) =P(s0)+P(s1)* şi deci algoritmul de grupare incrementală defineşte automat o sursă informațională binară.

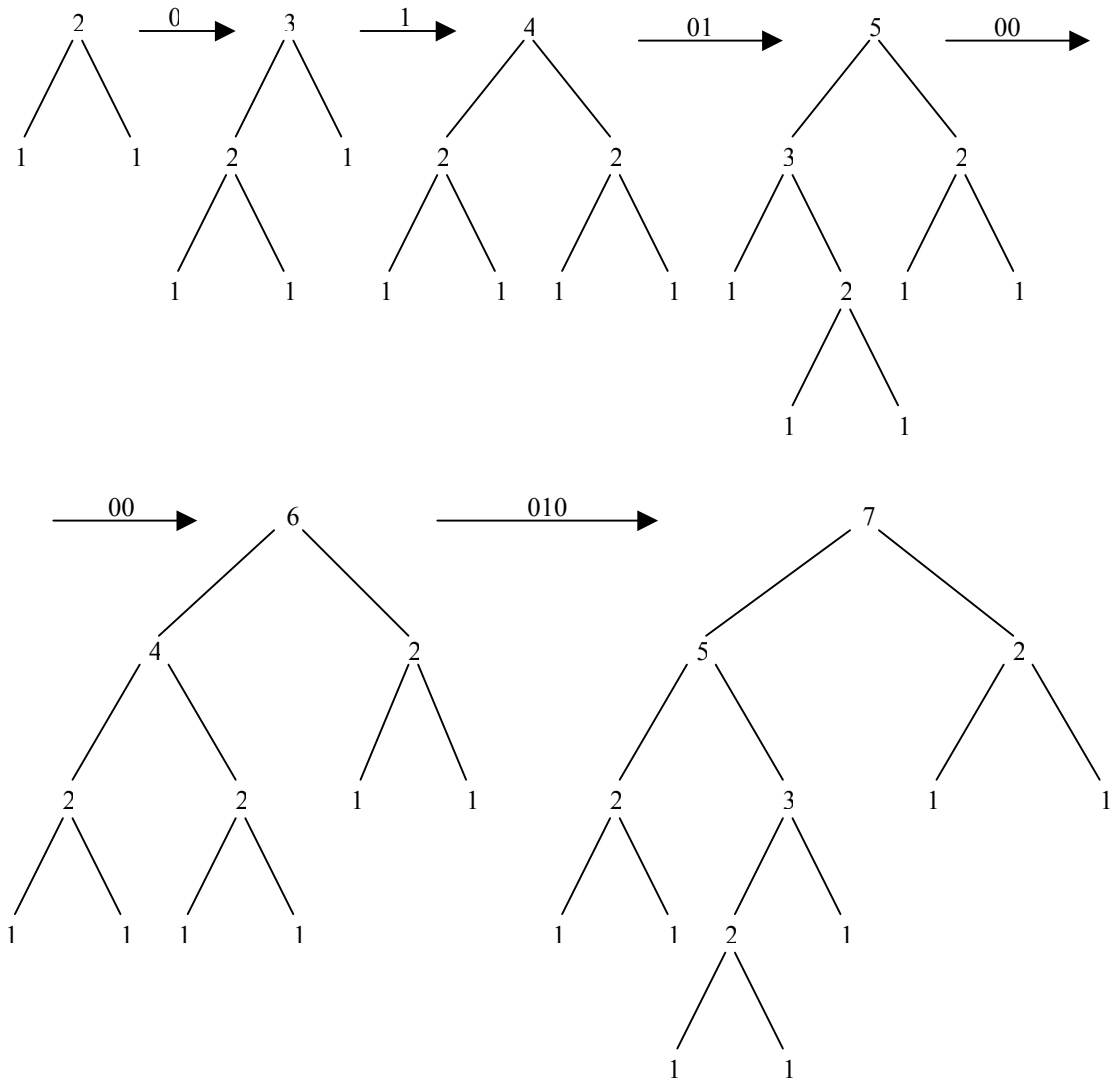

Figura 1. Arbore de grupare generat cu algoritmul lui Tunstall.

Algoritmul defineşte noul segment ca fiind una dintre căile către o frunză a arborelui *T*(*s*), în timp ce nodurile intermediare de+a lungul căii corespund prefixurilor segmentului. Dacă notăm prin *x* un prefix, atunci probabilitatea condiționată *P(x|s)* este raportul dintre ponderea nodului corespunzător lui *x* şi ponderea rădăcinii. Probabilitatea *P(sx)* a unui nou şir de forma *sx* este dată de *P(s)P(x|s)*. Astfel, probabilitatea unui şir *s* cu *n(s)* segmente este

$$P(s) = \frac{1}{(n(s)+1)!}. \tag{2}$$

Exemplu:

Să presupunem că şirul s = 010001010, dat ca exemplu mai sus, continuă cu secvența 011. Probabilitatea condiționată *P(011|s)* se determină astfel:

$$P(011|s) = P(0|s)\,P(1|s0)\,P(1|s01) = \frac{5}{7}\cdot\frac{3}{5}\cdot\frac{1}{3} = \frac{1}{7},$$

unde probabilitățile condiționate $P(0|s), P(1|s0), P(1|s01)$ se determină prin inspectarea arborelui.

Conform relației (2) lungimea de cod ideală pentru un şir s având $n(s)$ segmente este dată de:

$$-\log P(s) = \log(n(s)+1)!. \tag{3}$$

Semnificația modelului statistic obținut este aceea că şirul s este unul din cele $(n(s)+1)!$ şiruri posibile obținute prin alegerea aleatoare a celor $n(s)$ segmente.

Dezavantajul principal al algoritmului incremental de grupare constă din faptul că reuşeşte să surprindă corelații existente doar între simbolurile ce se află în acelaşi bloc (segment). Astfel, pentru o largă clasă de şiruri pentru care simbolurile prezintă corelații pe două sau mai multe dimensiuni, tehnicile ce generează segmente uni-dimensionale nu sunt eficiente şi deseori imposibil de implementat într-o manieră practică.

Să considerăm de exemplu cazul unui document alb-negru, partiționat in pătrate de dimensiuni reduse şi cărora li se asociază valoarea alb sau negru. Întregul document poate fi modelat ca un câmp bidimensional de variabile binare aleatoare $x(i,j)$, unde i şi j sunt coordonatele unui pătrat. Este de dorit găsirea unui model în care valorile variabilelor $x(i,j)$ nu sunt independente, ci depind de valorile variabilelor vecine. Pentru a surprinde dependențele existente pe ambele direcții prin gruparea de segmente, atunci trebuie luată în calcul utilizarea de segmente bi-dimensionale, deci de dreptunghiuri. Prin considerarea unui astfel de segment de „înălțime" k, ce va fi „crescut" la dreapta nu se vor putea surprinde corelații existente între elemente la distanță mai mare decât k decât dacă lungimea segmentului dreptunghiular depăşeşte lățimea documentului, iar segmentul este

continuat pe liniile următoare. „Creşterea” unor dreptunghiuri atât de mari nu poate fi realizată chiar şi în cazul simplu $k = 1$. Dacă un document are o lățime de w simboluri, atunci algoritmul trebuie să construiască un arbore de adâncime w pentru a surprinde o corelație existentă intre simbolul curent şi cel de deasupra lui. Pentru valori uzuale ale lui w numărul total de simbolut'ri din document nu este sficient „creşterii” unui arbore cu o adâncime atât de mare.

Acesta este motivul pentru care algoritmul Lempel-Ziv, în ciuda optimalității sale asimptotice, nu reuşeşte să se apropie de valorile limită ale compresiei pentru şiruri de test având dimensiuni uzuale.

Algoritmul Context adoptă o nouă abordare renunțând la cerința ca segmentele generate să fie partiționări ale şirului sursă. În cadrul acestui algoritm se urmăreşte colectarea unor mulțimi suprapuse de simboluri, fiecare definind un ***context***, utilizate în condiționarea aparițiilor simbolurilor.

Formal, un context se defineşte pe baza unei funcții definită pe mulțimea tuturor şirurilor binare B^*, cu valori în mulțimea numerelor naturale N. Contextul $z(t)$ al unui simbol $x(t)$ ce urmează secvenței trecute $s = x(1) \ldots x(t-1)$ este clasa şirurilor $s' = x'(1) \ldots x'(t-1)$ astfel încât $f(s') = f(s)$.

Un context reprezintă o clasă de echivalență prin aceea că nu orice valoare posibilă a secvenței anterioare de valori aleatoare defineşte în mod necesar un context distinct. E suficientă o anumită caracteristică a trecutului, comună mai multor variabile ce alcătuiesc secvența anterioară.Gradul de relevanță a unui anumit context poate fi apreciat prin capacitatea lui de a modifica numărul de apariții al unui simbol pentru a-i micşora entropia condițională.

În continuare se considera, pentru simplitate, cazul şirurilor binare $s = x(1) \ldots x(t-1)$.

Primul pas constă din stabilirea unei ordini a importanței simbolurilor trecute relativ la simbolul curent $x(t)$. Se urmăreşte practic ordonarea descrescătoare a simbolurilor trecute din punct de vedere al gradului de influențare pe care îl au asupra simbolului curent. În general distanța geometrică este considerată ca măsură a gradului de influențare.

Exemplu:

> Considerând cazul unui document modelat ca mai sus, de lățime w, ordinea în care se vor considera simbolurile trecute poate fi: mai întâi valoarea variabilei precedente $x(t–1)$, apoi cea situată deasupra variabilei curente $x(t–w)$, apoi cea situată în stânga-sus $x(t–w–1)$, şamd.

Pentru a generaliza se consideră o permutare de numere naturale $i \rightarrow t_i$ prin care se defineşte pentru orice şir $s = x(1) \ldots x(t-1)$ un alt şir, notat $\sigma(s) = x(t-t_1) \ldots x(t-t_{t-1})$.

Pentru valori mici ale lui t există i pentru care $t - t_i \leq 0$. În mod normal $\sigma(s)$ se alege astfel încât să fie format doar din simbolurile consecutive $x(t-t_1)x(t-t_2)$...,având indice pozitiv. Dacă chiar primul simbol are indice negativ atunci fie se defineşte $\sigma(s)$ ca fiind şirul vid λ, fie se admite o valoare arbitrară pentru simbolurile de indice negativ.

Al doilea pas constă din „creşterea" a doi arbori binari, unul pentru cazul în care simbolul curent $x(t)$ are valoarea 0, iar celălalt pentru cazul în care are valoarea 1. Interesează intersecția celor doi arbori, care va fi generată direct, printr-o procedură ce va fi descrisă în continuare şi care poartă numele de Algoritm Context.

Deoarece procedura nu implică detalii legate alegerea *funcției de sortare* – σ, vom considera $\sigma(s)$ ca fiind secvența s în ordine inversă: $x(t-1) \ldots x(1)$, pe care o vom nota cu $z_1 \ldots z_{t-1}$. De asemenea vom nota valoarea simbolului curent $x(t)$ cu u.

Generarea arborelui Context

1. Se declară arborele context al primului simbol $x(1)$ ca fiind arborele $T(0)$ cu un singur nod, rădăcina, care este marcat cu perechea de contoare $\left(c(0,\lambda), c(1,\lambda)\right) = (1,1)$.

2. Recursiv, fie $T(t-1)$ ultimul arbore context construit, în care notăm cu $(c(0,z), c(1,z))$ perechea contoare de la nodul z. La observarea noului simbol $x(t)=u$, se generează următorul arbore $T(t)$ după cum urmează:
    - se parcurge arborele $T(t-1)$ începând de la rădăcină urmându-se ramura stângă pentru 0 şi ramura dreaptă pentru 1, succesiunea de ramuri parcurse fiind indicată de fiecare simbol succesiv din secvența trecută $\sigma(x(1) \ldots x(t-1)) = z_1 z_2 \ldots$;
    - pentru fiecare nod z vizitat se incrementează contorul $c(u,z)$;
    - parcurgerea arborelui se opreşte când este atins un nod w al cărui contor $c(u,w)$ are valoarea 1 (înainte incrementării), şi se trece la pasul 3;
3. Se tratează diferit cele două cazuri posibile:

    a) w este un nod intern având nodurile fiu $w0$ în stânga şi $w1$ în dreapta:
        - se incrementează contoarele $c(u, w0)$ şi $c(u, w1)$;
        - arborele obținut este notat cu $T(t)$ şi apoi se reia pasul 2;

    b) w este un nod frunză:
        - se extinde arborele prin crearea a două noi frunze $w0$ şi $w1$;
        - se inițializează contoarele celor doua noduri: $c(u,w0)=c(u,w1)=1$ şi $c(u',w0)=c(u',w1)=0$, unde u' reprezintă simbolul opus lui u;
        - arborele obținut este notat cu $T(t)$ şi apoi se reia pasul 2;

Exemplu:

Fie şirul binar 10001, pentru care use utilizează permutarea identitate (simbolurile trecute sunt ordonate după distanța față de simbolul curent). Evoluția arborelui context, descrisă de procedura de mai sus, este prezentată în figura 2.

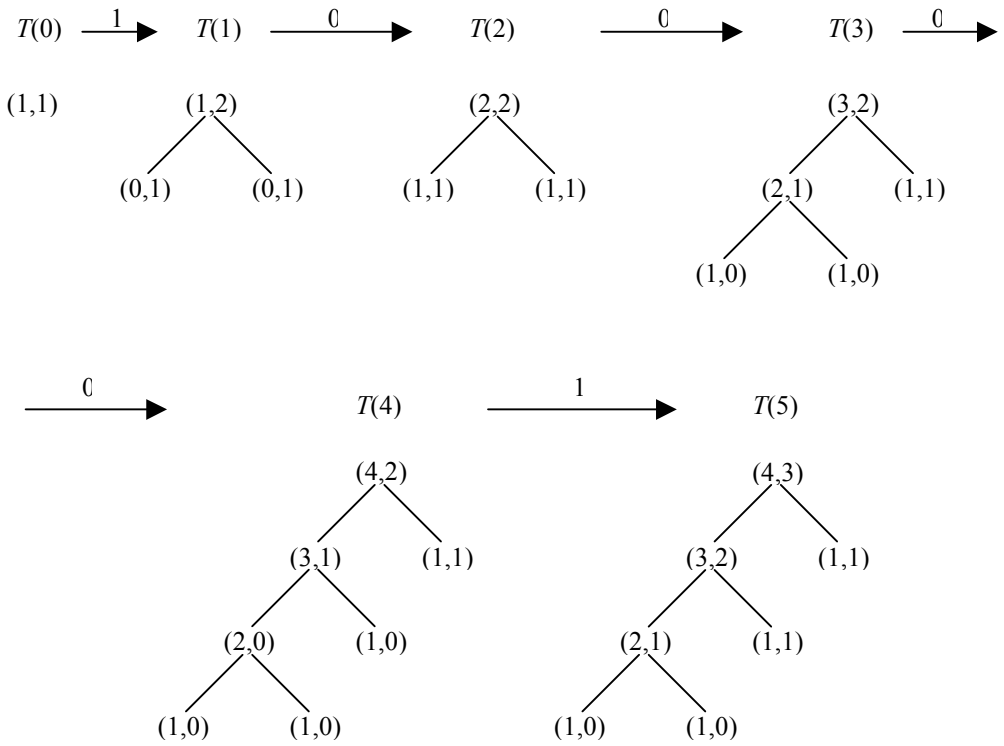

Figura 2. Creşterea arborelui context.

Valorile acumulate în nodurile arborelui context conţin de fapt informaţia statistică asociată sursei. Dacă se notează cu z un nod al arborelui context, atunci acesta este definit de (şi identificat cu) secvenţa $z_1 \ldots z_n$, care desemnează calea de la rădăcină la nodul z. Se notează în continuare cu $c(z)$ suma contoarelor din nodul z, iar cu $c(u,z)$ numărul de apariţii ale simbolului u precedat de secvenţa z. Cu aceste notaţii se poate defini probabilitatea condiţionată a simbolului u în contextul z astfel:

$$P(u|z) = \frac{c(u,z)}{c(z)}, \tag{4}$$

Valorile acestor probabilităţi se determină prin inspectarea arborelui context. De exemplu, pentru arborele final din figura 2 avem $P(x(t) = 0 \mid z{=}0) = 3/5$.

Pentru cazul în care simbolul u nu a apărut niciodată în contextul z, i.e. $c(u,z) = 0$, rămâne posibilitatea ca acesta să apară în viitor şi din motive legate de codare nu i se poate asigna o probabilitate de apariţie nulă. Pentru aceste cazuri probabilitatea (4) se scrie:

$$P(u|z) = \frac{1}{c(z)+1}, \quad \text{pentru } c(u,z) = 0. \tag{5}$$

Modalitatea de alegere a contextelor va face ca $P(u|z)$ să fie calculat cu relația (5) doar la începutul codării şirului, relația (4) fiind cea folosită în mod normal.

Probabilitățile condiționate de mai sus definesc entropia binară

$$H(U;z) = -p\log p - (1-p)\log(1-p), \text{ unde } p = P(0|z) \tag{6}$$

Datorită modalității de construcție a arborelui, pentru toate nodurile în care valoarea contoarelor este mai mare decât 1 se repectă relația

$$c(u,z0) + c(u,z1) = c(u,z) \tag{7}$$

Sumând după valorile posibile ale simbolurilor se obține practic condiția de compatibilitate

$$c(z0) + c(z1) = c(z) \tag{8}$$

Calitatea modelul oferit de Algoritmul Context depinde critic de alegerea contextelor. Arborii $T(t)$ generați conțin practic toate contextele trecute. Contextele unui simbol $x(t)$ corespund nodurilor vizitate de-a lungul parcurgerii arborelui $T(t–1)$ descrisă de secvența trecută $z_1\ z_2...$. Întrebarea este care dintre aceste noduri să fie ales ca şi context al simbolului $x(t)$ pentru a obține o modelare care să permită o compresie cât mai bună.

În momentul în care alegerea contextelor a fost efectuată, adică fiecărui simbol $x(t)$ i s-a asociat un anumit context $z^*(t)$, care este un prefix al şirului $\sigma(x(1)\ ...\ x(t–1))$, a fost practic definită o sursă binară $\langle B^*,P\rangle$, unde $P(s)$ este definită ca produsul probabilităților condiționale ale simbolurilor lui s, descrise de relațiile (4) sau (5). Aceste probabilități se pot utiliza pentru a se coda fiecare simbol, iar lungimea optimă de cod pentru şirul $s = x(1)\ ...\ x(t-1)$ este:

$$-\log P(s) = -\sum_{i=1}^{t} \log P\left(x(i)\middle|z^*(i)\right). \tag{9}$$

Relația (6) sugerează o regulă de alegere a contextelor: $z^*(t)$ este primul nod z din secvența $z_1\ z_2...$ pentru care probabilitatea $P(x(i)|z)$ este cea mai îndepartată de 1/2. Dacă pentru o sursă staționară fiecare context z^* apare din ce în ce în ce mai frecvent, atunci lungimea cuvântului de cod per simbol tinde către o entropie condițională de tipul

$$\sum P(z)H(U;z) = H(U|Z), \tag{10}$$

unde z parcurge toată mulțimea Z ce alcătuieşte un subarbore complet ale cărui frunze au contorul $c(u,z) > 1$. Alegerea contextelor sugerată mai sus minimizează entropia $H(U;z)$ descrisă de (6) şi astfel se minimizează şi lungimea medie per simbol cuvintelor de cod precum şi entropia condiționată din (10).

Principala problemă legată de această alegere a contextelor este că $z^*(t)$ tinde să crească odată cu lungimea şirului, t. Acest lucru se datorează principial datorită faptului ca prin condiționarea la un set mai mare micşorează entropia condiționată. O soluție ar fi impunerea unui limite fixe asupra numărului de contexte, însă ar fi de preferat găsirea unui algoritm care să stabilească limita pe măsură ce procesează şirul.

Abordarea în dezvoltarea unui asemenea algoritm „inteligent" urmăreşte asignarea unui cost fiecărui context din Z. Acceptarea unui context în Z se face doar în cazul în care aportul acestuia la reducerea entropiei condiționale depăşeşte costul asociat. Este necesară deci evaluarea creşterii entropiei condiționate care rezultă prin includerea a două noduri $z0$ şi $z1$ din Z în nodul părinte z. Notând cu Z' mulțimea redusă ce rezultă, creşterea entropiei se determină astfel:

$$\begin{aligned}\Delta(t,z) &= H(U|Z') - H(U|Z) \\ &= P(z)H(U;z) - P(z0)H(U;z0) - P(z1)H(U;z1),\end{aligned} \tag{11}$$

O regulă generică de alegere a contextelor a fost propusă de Rissanen odată cu introducerea Algoritmului Context. Aceasta propune alegerea ca şi context $z^*(t)$ al simbolului $x(t)$ nodul din arborele $T(t-1)$ având cea mai mare lungime (cea mai lungă cale definită de secvența anterioară $z_1 z_2 ...$) şi care respectă:

$$\Delta(t,z) > \frac{1}{t}\log t \tag{12.a}$$

$$|z| \le \beta \log t \tag{12.b}$$

$$\min\{c(z0), c(z1)\} \ge \frac{2\alpha t}{\sqrt{\log t}} \tag{12.c}$$

Parametrii α şi β sunt numere pozitive şi au ca scop definirea domeniului de noduri testate pentru a deveni contexte doar pentru cazul în care se codează un şir finit. Pentru şiruri infinite orice valori servind scopului propus.

Prima limită exprimată de relația (12.b) este respectată automat în majoritatea cazurilor, deoarece adâncimea întregului arbore este de ordinul $\log t$. Scopul celei de-a doua limite, exprimată de relația (12.c) este de a asigura creşterea contoarelor pentru contextele selectate, evitând restricționarea ca mulțimea de contexte să fie uniform limitată.

Nodurile context pot fi marcate prin setarea unui bit de tip flag. Crearea unui nou context determină ca un nod din aceeaşi cale anterior marcat ca şi nod context să fie transformat într-un nod normal, prin resetarea bitului flag. Astfel fiecare cale va avea un context şi deci mulțimea contextelor $Z(t)$ formează un subarbore complet. Notând cu Z structura stabilizată a subarborelui de contexte, alegerea descrisă mai sus minimizează local costul combinat

$$H\left(U\middle|Z\right)+\frac{\left(|Z|\log t\right)}{t} \tag{13}$$

unde $|Z|$ reprezintă numărul de elemente din Z. Prin minimizare locală se înțelege că orice modificare a unui context determină creşterea (13).

Observații

Dimensiunea arborelui context final, este puternic dependentă de alegerea funcției de sortare σ. Cazul cel mai favorabil este atunci când funcția de sortare este cunoscută. În acest caz algoritmul nu va mai creşte arborii context oricât de lung ar fi şirul codat, spre deosebire de algoritmul Lempel-Ziv, care ar genera segmente din ce în ce mai lungi până la epuizarea resurselor de memorie, fără să realizeze că nu se mai câştigă în compresie.

Cazul cel mai întâlnit este însă cel în care funcța de sortare nu este cunoscută. Totuşi, acest algoritm universal oferă o modalitate de estimare experimentală a acesteia: se fixează o dimensiune maximă a arborilor $T(t)$ şi se rulează algoritmul pentru diferite variante ale funcției de sortare, cea care oferă cea mai bună compresie oferind cele mai bune contexte.

3.4. Codarea

În cadrul unei scheme de compresie bazată pe contexte folosirea unui codor Huffman adaptiv sau a unui codor aritmetic adaptiv, ar putea duce la o creştere semnificativă a complexității întregii scheme de compresie. Creşterea complexității se datorează spațiului necesar stocării tuturor informațiilor cu care lucrează astfel de codoare şi în plus actualizarea acestora la fiecare pas. Dacă am avea o imagine în tonuri de gri şi o schemă de compresie bazată pe contexte, care foloseşte 365 de contexte, pentru unul din codoarele de mai sus ar trebui păstrate cel puțin 365 x 512 valori.

Astfel, este evidentă nevoia unui algoritm de compresie care să permită determinarea, la fiecare pas, a cuvântului de cod asociat erorii de predicție corectate prin păstrarea şi actualizarea unui număr mai mic de date.

3.4.1. Codurile Golomb

În cadrul algoritmului JPEG-LS se folosesc codurile Golomb. Aceste coduri au fost descrise pentru prima dată în literatură ca mijloc pentru codarea secvențelor de simboluri cu aceeaşi valoare, de exemplu o secvență 00....0, numite *run-length*. Acest cod se aplică numerelor întregi pozitive.

Dându-se un întreg pozitiv m, codul Golomb de ordin m, G_m, codează un întreg pozitiv y în două părți:

1. o reprezentare binară a lui $y \bmod m$.
2. o reprezentare unară a lui $\lfloor y/m \rfloor$.

Codurile Golomb sunt optimale pentru distribuții de probabilitate geometrice, ale numerelor întregi pozitive, de forma:

$$Q(n) = (1 - \rho)\rho^n \,,\ \rho \in (0, 1)$$

Pentru fiecare distribuție de această formă, există o valoare a parametrului m, astfel încât G_m determină codul cu cea mai mică lungime medie de cod dintre toate codurile unic descifrabile.

Rice a evidențiat cazul special al codurilor Golomb pentru care *m* este de forma 2^k. Alegând *m* astfel încât acesta este o putere a lui 2, rezultă un algoritm foarte simplu de codare / decodare şi anume: cuvântul de cod asociat simbolului *y* este format din cei mai puțin semnificativi *k* biți ai acestuia plus o reprezentare unară a numărului format din cei mai semnificativi biți rămaşi ai lui *y*. Lungimea de cod a cuvântului astfel format este :

$$L = k + 1 + \left\lfloor y / 2^k \right\rfloor$$

Vom numi codurile Golomb de forma G_2^k coduri Golomb-Rice şi le vom nota cu R_k..

Golomb Rice	$m=1$ $k=0$	$m=2$ $k=1$	$m=3$	$m=4$ $k=2$	$\cdots$	$m=6$	$\cdots$	$m=8$ $k=3$
$n=0$	**0·**	**0·0**	**0·0**	**0·00**		**0·00**		**0·000**
1	**10·**	**0·1**	**0·10**	**0·01**		**0·01**		**0·001**
2	**110·**	**10·0**	**0·11**	**0·10**		**0·100**		**0·010**
3	**1110·**	**10·1**	**10·0**	**0·11**		**0·101**		**0·011**
4	**11110·**	**110·0**	**10·10**	**10·00**		**0·110**		**0·100**
5	**111110·**	**110·1**	**10·11**	**10·01**		**0·111**		**0·101**
6	**1111110·**	**1110·0**	**110·0**	**10·10**		**10·00**		**0·110**
7	**11111110·**	**1110·1**	**110·10**	**10·11**		**10·01**		**0·111**
8	**111111110·**	**11110·0**	**110·11**	**110·00**		**10·100**		**10·000**
9	**1111111110·**	**11110·1**	**1110·0**	**110·01**		**10·101**		**10·001**
⋮	⋮	⋮	⋮	⋮		⋮		⋮