KOD ARYTMETYCZNY

Materiały KODA, A.Przelaskowski

- Pomysł jednej liczby
- Koncepcja zwężanego przedziału
- Algorytmy kodu
- Implementacje całkowitoliczbowe
- Modelowanie kontekstu
- PPM
- Kodery binarne
- CTW
- Implementacje BAC
- Testy efektywności

Pomysł jednej liczby

- Zamiast relacji prawdopodobieństwo (alfabet) $\rightarrow$ słowo kodowe ustalamy jedynie prawdopodobieństwo sekwencji wejściowej
- Właściwy opis prawdopodobieństw (kumulacja):

$A_S=\{a_1,\ldots,a_n\}$ i $P_S=\{p_1,\ldots,p_n\} \rightarrow F_S=\{p_1,p_1+p_2,\ldots,p_1+\ldots+p_n\}$

czyli $F_S=\{F(a_i): i= 1,\ldots,n\}$, $F(a_i)=\sum_{j=1}^{i} p_j$

- Przyporządkowanie przedziału ($F(a_0)=0$):

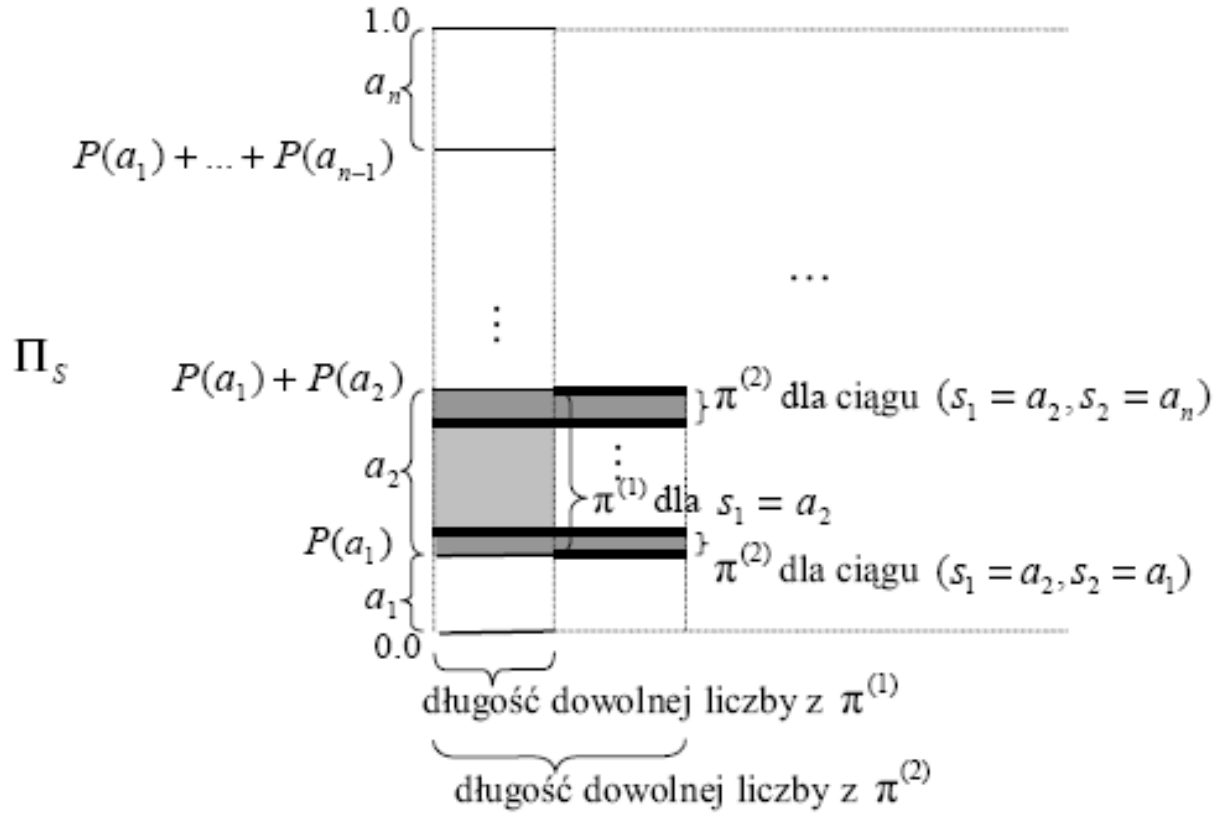
$$\pi_s : \ a_i \in A_S \rightarrow [F(a_{i-1}), F(a_i)) \subset [0,1)$$

- Ogólniej przydzielamy podprzedział w danym $[d,g) \subset [0,1)$:

$$\pi_{[d,g)} : \ a_i \in A_S \rightarrow [d + F(a_{i-1}) \cdot (g-d), d + F(a_i) \cdot (g-d)) \subset [d,g)$$

Reguła: model identyfikuje ciąg informacji, czyli dystrybuanta, podprzedziały i (nieograniczone) bloki

Modyfikacja przedziału (zasadnicza koncepcja KA)



kodowanie to modyfikacja przedziału kodu $\pi^{(i)}= \pi(s_1,s_2,\ldots s_i)= \pi(s_i=a_k, \pi^{(i-1)})$

zasada zawężania przedziału: $\pi^{(i)} \subset \pi^{(i-1)}$

Kodowanie i dekodowanie

- Koder: $\pi^{(i)} = \left[D^{(i)}, G^{(i)}\right), \; i = 1,2,\ldots$

$$D^{(i)} = D^{(i-1)} + \left(G^{(i-1)} - D^{(i-1)}\right) F(a_{k-1})$$

$$G^{(i)} = D^{(i-1)} + \left(G^{(i-1)} - D^{(i-1)}\right) F(a_k)$$

liczba kodowa: dowolna $\mathcal{L}^t \in \pi^{(t)}$

- Dekoder (dwa sposoby):
  - przeskalowanie liczby kodowej do linii prawdopodobieństw

$$\mathcal{L}^t = \mathcal{L}^{(1)} \quad s_i = a_k \Leftrightarrow \mathcal{L}^{(i)} \in \left[F(a_{k-1}), F(a_k)\right]$$

$$\mathcal{L}^{(i+1)} = \frac{\mathcal{L}^{(i)} - F(a_{k-1})}{F(a_k) - F(a_{k-1})}$$ **normalizacja $\pi^{(1)}$**

  - śledzenie przedziału kodowego z ograniczoną dokładnością liczby kodowej

$$s_i = a_k \Leftrightarrow \frac{\left[\mathcal{L}^t\right]^{(i)} - D^{(i)}}{G^{(i)} - D^{(i)}} \in \left[F(a_{k-1}), F(a_k)\right]$$ **normalizacja $\pi^{(i)}$**

Efektywność KA

- według teorii

$$1 - \log_2 P(\mathbf{s}) \leq L_{\text{aryt}} < 2 - \log_2 P(\mathbf{s})$$

- Implementacyjnie

$$-\lceil \log_2 R^{(t)} \rceil \leqslant |\mathcal{L}^t| < -\lceil \log_2 R^{(t)} \rceil + 1$$

- poprawa to przede wszystkim modelowanie linii prawdopodobieństw (większy przedział symbolu to mniejsze zwężenie przedziału kodu)

$$\Pi_{S|\mathbf{c}_i^{(m)}} = \left\{ \left[F\left(a_{i-1}|\mathbf{c}_i^{(m)}\right), F\left(a_i|\mathbf{c}_i^{(m)}\right) \right), \ i = 1, \ldots, n \right\}$$

linia prawdopodobieństw warunkowych

Przykład - kodowanie

a_i	$P(a_i)$	$\pi_s(a_i)$
„A”	2/10	[0,2/10)
„E”	1/10	[2/10,3/10)
„K”	1/10	[3/10,4/10)
„M”	1/10	[4/10,5/10)
„R”	1/10	[5/10,6/10)
„T”	2/10	[6/10,8/10)
„Y”	2/10	[8/10,1)

Wejście = „ARYTMETYKA"

linia prawdopodobieństw

i	Kolejne symbole wejściowe $s_i = a_k$	$D^{(i)}$	$G^{(i)}$	$R^{(i)}$
0	inicjalizacja	0	1	1
1	„A”	0,0	0,2	0,2
2	„R”	0,1	0,12	0,02
3	„Y”	0,116	0,12	0,004
4	„T”	0,1184	0,1192	0,0008
5	„M”	0,11872	0,1188	0,00008
6	„E”	0,118736	0,118744	0,000008
7	„T”	0,1187408	0,1187424	0,0000016
8	„Y”	0,11874208	0,1187424	0,00000032
9	„K”	0,118742176	0,118742208	0,000000032
10	„A”	0,118742176	0,1187421824	0,0000000064

Przykład - dekodowanie

a_i	$P(a_i)$	$\pi_s(a_i)$
„A”	2/10	[0,2/10)
„E”	1/10	[2/10,3/10)
„K”	1/10	[3/10,4/10)
„M”	1/10	[4/10,5/10)
„R”	1/10	[5/10,6/10)
„T”	2/10	[6/10,8/10)
„Y”	2/10	[8/10,1)

i	Liczba kodowa $\mathcal{L}^{(i)}$	Symbole dekodowane $s_i = a_k$	$F(a_{k-1})$	$F(a_k)$	$F(a_k) - F(a_{k-1})$
1	0,118742176	„A”	0	0,2	0,2
2	0,59371088	„R”	0,5	0,6	0,1
3	0,9371088	„Y”	0,8	1	0,2
4	0,685544	„T”	0,6	0,8	0,2
5	0,42772	„M”	0,4	0,5	0,1
6	0,2772	„E”	0,2	0,3	0,1
7	0,772	„T”	0,6	0,8	0,2
8	0,86	„Y”	0,8	1	0,2
9	0,3	„K”	0,3	0,4	0,1
10	0	„A”	0	0,2	0,2
...	0	...	...	...	...

Implementacja całkowitoliczbowa

- rejestry o ograniczonej długości, np. 16 bitowe
- przedział niedomknięty z góry [0, $0{,}1111_2$….)
- usuwamy **0,** i uzupełniamy DOL zerami i GORA jedynkami
- mamy więc

  DOL=0x0000 oraz GORA=0xFFFF

  szerokość przedziału: GORA-DOL+1
- wysuwanie najbardziej znaczącej cyfry, gdy jest jednakowa w DOL i GORA i uzupełnianie zerami (DOL) oraz jedynkami (GORA)
- eliminacja niedomiaru (najstarsza cyfra różna o 1, kolejna to 0 w GORA i 9 (1_2) w DOL:
  - najbardziej znaczące cyfry pozostają bez zmian, zaś pozostałe przesuwane są o jedną pozycję w lewo (giną niewygodne 0 i 9 na pozycji niedomiaru)
  - najmłodsza pozycja w rejestrach zostaje odpowiednio uzupełniona
  - inkrementacja licznika niedomiarów

Algorytmy kodera

- szacowanie statystyki
- inicjalizacja: DOL=0…..0, GORA=1…..1 oraz licznik niedomiaru $L = 0$
- modyfikacje: PRZE = GORA − DOL + 1

$$\text{DOL} = \text{DOL} + \text{PRZE} \cdot F(a_{k-1})$$

$$\text{GORA} = \text{DOL} + \text{PRZE} \cdot F(a_k) - 1$$

- jeśli najstarszy bit *b* w DOL i GORA jest jednakowy:
  - cyfry obu rejestrów przesuń o jedną pozycję w lewo, a wysuniętą najstarszą cyfrę prześlij na wyjście
  - uzupełnij DOL zerem oraz GORA jedynką
  - jeśli licznik $L > 0$, dosyłanych jest L bitów $(1 - b)$; $L = 0$
- jeśli najstarszy bit GORA i DOL jest różny, a drugi bit to odpowiednio 0 w GORA i 1 w DOL - przesuń w lewo bity obu rejestrów z wyjątkiem dwu najbardziej znaczących i uzupełnij rejestry; $L = L + 1$
- zakończenie: dopisz do wyjścia wszystkie znaczące bity z DOL

Przykład: kodowanie całkowitoliczbowe

Kolejne s_i i działania pomocnicze	DOL	GORA	GORA-DOL+1	Ciąg kodowy
Inicjalizacja	00000	99999	100000	–
„A”	00000	19999	020000	
„R”	10000	11999		
Wysuń 1	00000	19999	020000	1
„Y”	16000	19999		
Wysuń 1	60000	99999	040000	11
„T”	84000	91999	008000	
„M”	87200	87999		
Wysuń 8 i 7	20000	99999	080000	1187
„E”	36000	43999	008000	
„T”	40800	42399		
Wysuń 4	08000	23999	016000	11874
„Y”	20800	23999		
Wysuń 2	08000	39999	032000	118742
„K”	17600	20799	003200	
„A”	17600	18239		
Wysuń 1	76000	82409		1187421
Wysuń 7 i 6				118742176

a_i	$P(a_i)$	$\pi_s(a_i)$
„A”	2/10	[0,2/10)
„E”	1/10	[2/10,3/10)
„K”	1/10	[3/10,4/10)
„M”	1/10	[4/10,5/10)
„R”	1/10	[5/10,6/10)
„T”	2/10	[6/10,8/10)
„Y”	2/10	[8/10,1)

Algorytm dekodera

- Pobierz linię prawdopodobieństw
- Ustal DOL=0….0, GORA=1....1 oraz licznika *i* = 1
- Wczytaj do KOD *η* najstarszych bitów liczby kodowej
- Oblicz lokalną liczbę kodową:

$$\mathcal{L}_{\text{lok}} = \frac{\text{KOD} - \text{DOL}}{\text{GORA} - \text{DOL} + 1}$$

- Dekoduj symbol:

$$s_i = a_k \Leftrightarrow \mathcal{L}_{\text{lok}} \in \left[F(a_{k-1}), F(a_k)\right)$$

- Modyfikuj DOL i GORA według a_k jak w koderze: KOD uzupełnij kolejnymi bitami liczby kodowej, w sytuacji niedomiaru postępuj analogicznie jak w koderze
- Licz dekodowane symbole *i* =*i*+1

Przykład: dekodowanie całkowitoliczbowe

KOD	DOL	GORA	GORA-DOL+1	$\mathcal{L}_{\text{lok}}$	Dekodowane s_i i działania pomocnicze
11874	00000	99999	100000	0,11874	„A”
	00000	19999	020000	0,5937	„R”
	10000	11999			Usuń 1
18742	00000	19999	020000	0,9371	„Y”
	16000	19999			Usuń 1
87421	60000	99999	040000	0,685525	„T”
	84000	91999	008000	0,427625	„M”
	87200	87999			Usuń 8 i 7
42176	20000	99999	080000	0,2772	„E”
	36000	43999	008000	0,772	„T”
	40800	42399			Usuń 4
21760	08000	23999	016000	0,86	„Y”
	20800	23999			Usuń 2
17600	08000	39999	032000	0,3	„K”
	17600	20799	003200	0	„A”

Procedury kodera (implem. C)

```
/* Inicjalizacja globalnych zmiennych kodowania */

unsigned short int dol;
unsigned short int gora;
long niedomiar;

...
...

void inicjalizuj_koder()
{
 dol=0;
 gora=0xffff;
 niedomiar=0;
}

...
...

/* Funkcja kodowania symbolu s */
void koduj_symbol(ZBIOR_BIT *zbior, SYMBOL *s, int
                  s_dol, int s_gora, int calk_symbol)
/* do zbior zapisywany jest ciąg kodowy, s_dol i s_gora to
przedział skumulowanych częstości występowania s przy danym
kontekście, a calk_symbol to całkowita liczba wystąpień symb
przy danym kontekście (przy zerowym rzędzie modelu jest
to liczba wszystkich zakodowanych dotąd symboli */
{

 long przedzial;

 /* Aktualizacja przedziału kodowego */
 przedzial=(long)(gora-dol)+1;
 gora=dol+(unsigned short int)((przedzial*s_gora)/calk_symbol-1);
 dol=dol+(unsigned short int)((przedzial*s_dol)/calk_symbol);
```

```
/* Normalizacja zmiennych przedziału */
for(;;) {
 if((gora & 0x8000)==(dol & 0x8000)) {
                                  /* Najstarszy bit jest
                                     jednakowy */
  WyslijBit(zbior,gora & 0x8000);
                                  /* Zapisanie najstarszego bitu
                                     obu rejestrów */
  while(niedomiar>0) {
   WyslijBit(zbior,~gora & 0x8000);
                                  /* Zapisanie negacji najstarszego
                                     bitu obu rejestrów */
   niedomiar--;
  }
 } else if((dol & 0x4000) && !(gora & 0x4000)) {
                                  /* Wykrycie zagrożenia
                                     niedomiarem */
  niedomiar+=1;
  dol&=0x3fff;
  gora|=0x4000;
 } else
  return;

 dol<<=1;
 gora<<=1;
 gora|=1;
}
}
```

Procedury dekodera (implem. C)

```
/* Inicjalizacja globalnych zmiennych dekodowania */

unsigned short int kod; /* Zmienna, do której wpisywana
                           jest reprezentacja kodowa */
unsigned short int dol;
unsigned short int gora;

...
...

void inicjalizuj_dekoder(ZBIOR_BIT *zbior )
/* zbior to zbior bitowy zawierający ciąg danych kodowych
{
 kod=0;
 for(i=0;i<16;i++) { /* Wczytanie pierwszych 16 bitów
   kod<<=1;
   kod+=PobierzBit(zbior);
  }
 dol=0;
 gora=0xffff;
}

...
...
```

```
/* Funkcja dekodowania symbolu s */
int dekoduj_symbol(ZBIOR_BIT *zbior, short int kum_waga[])
/* ze zbior czytany jest ciąg kodowy; kum_waga to tablica
skumulowanych częstości wystąpień kolejnych symboli alfabetu
w porządku malejącym, tj. kum_waga[0] zawiera liczbę wszystkich
wystąpień danego kontekstu */
{
 long przedzial;
 short int calk_symbol;
 int s;

 calk_symbol)=kum_waga[0];
 przedzial=(long)(gora-dol)+1;
 liczba_kod=
   (short int)((((long)(kod-dol)+1)*calk_symbol-1)/przedzial);
 for(s=0;liczba_kod<kum_waga[s];s++);
                                  /* Dekodowanie symbolu poprzez
                                     rzutowanie liczby kodowej na
                                     linię prawdopodobieństw */
 s_gora=kum_waga[s-1];            /* Ustalenie podprzedziału
                                     symbolu s*/
 s_dol=kum_waga[s];
 /* Aktualizacja przedziału kodowego */
 przedzial=(long)(gora-dol)+1;
 gora=dol+(unsigned short int)((przedzial*s_gora)/calk_symbol-1);
 dol=dol+(unsigned short int)((przedzial*s_dol)/calk_symbol);

 /* Normalizacja zmiennych przedziału */
 for(;;) {
  if((gora & 0x8000)==(dol & 0x8000)) {
  } else if((dol & 0x4000)==0x4000 && (gora & 0x4000)==0) {
   kod ^= 0x4000;
   dol &= 0x3fff;
   gora |= 0x4000;
  } else
   return(s);
  dol<<=1;
  gora<<=1;
  gora|=1;
  kod<<=1;
  kod+=PobierzBit(zbior); /* Wczytanie kolejnego bitu ciągu kodowego
 }
```

MODELOWANIE KONTEKSTU

Modelowanie kontekstu

- dobór właściwej postaci kontekstu modelu Markowa rzędu *m* (wielkość, kształt, alfabet) – problem rozrzedzenia kontekstu

- zwiększanie rzędu kontekstu naturalnego (strumieniowego,1W)

	Wartości rzędu modelu m									
	1	2	3	4	5	6	7	8	9	10
$H(S\|C^{(m)})$	2,03	1,55	1,1	0,97	0,56	0,34	0,25	0,24	0,23	0,19
$\bar{L}_1$	2,04	1,61	1,31	1,11	1,02	0,96	0,92	0,85	0,84	0,84
$\bar{L}_2$	2,49	2,08	1,90	1,81	1,81	1,82	1,83	1,81	1,83	1,85

wartości średniej bitowej

plik tekstowy długości 1,1MB

pierwsze 50kB tego pliku

- kwantyzacja kontekstu, np. liniowa kombinacja z kwantyzacją alfabetu

$$\check{c}_i^{(1)} = \left\lceil \frac{\alpha_1 s_{i-1} + \alpha_2 s_{i-2} + \alpha_3 s_{i-3}}{4} \right\rceil$$

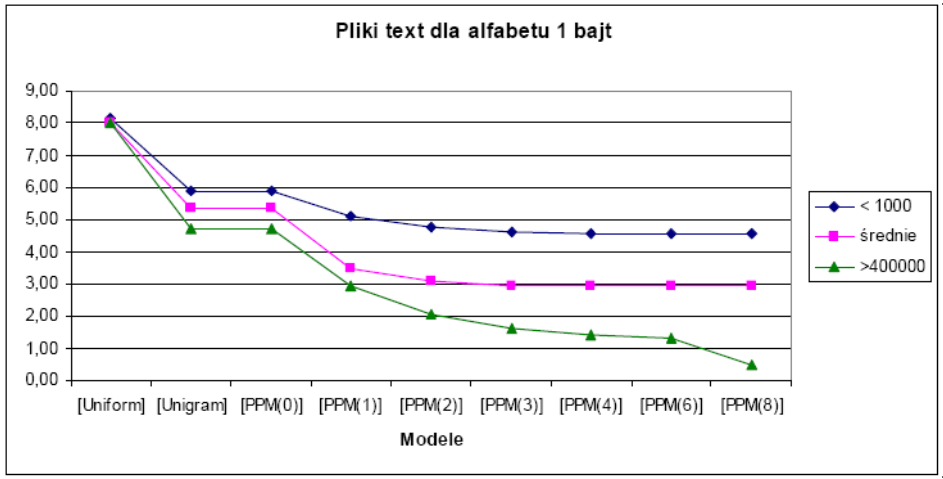
Metoda PPM (*prediction with partial string matching*)

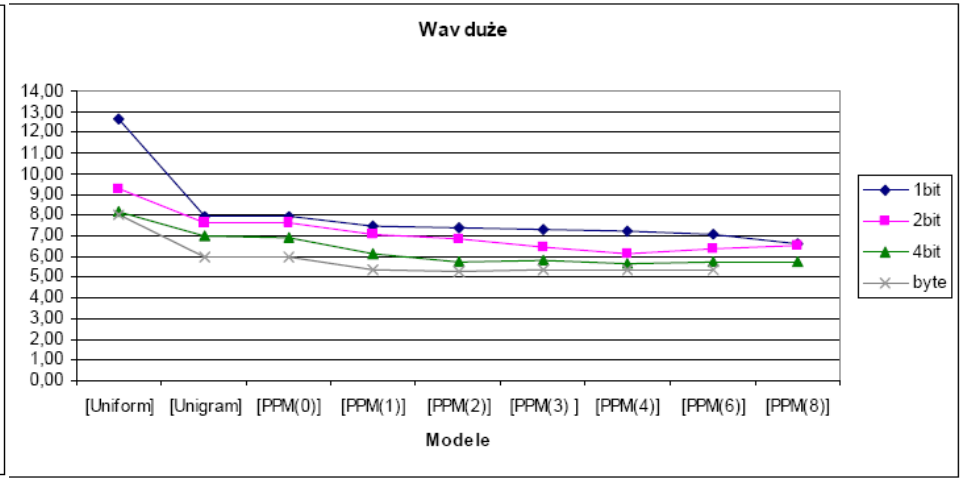
- adaptacyjny kontekst: różne rozmiary kontekstu
  - szukamy maksymalnego rozmiaru kontekstu
  - zaczynamy od rzędu *m* i szukamy adaptacyjnej statystyki wystąpienia symbolu w danym kontekście
  - przy braku kontekstu rzędu *m* bierzemy pod uwagę kontekst *m* -1 itd.
  - przy braku symbolu w alfabecie danego kontekstu kodujemy pomocniczy symbol PRZEŁĄCZ i przechodzimy do kontekstu niższego rzędu
  - jeśli linie kontekstów rzędu *m*, *m*-1,…,0 nie występują bądź nie mają danego symbolu - stosowany jest np. równomierny rozkład wszystkich symboli alfabetu źródła
- adaptacyjny alfabet każdego kontekstu: rozszerzany alfabet
  - symbol jest wprowadzony do alfabetu danego kontekstu dopiero po jego wystąpieniu (w PPMB po dwukrotnym wystąpieniu)
  - alfabet zawiera dodatkowy symbol PRZEŁĄCZ (o różnorako szacowanej wadze)
  - uaktualnia się linie wszystkich rzędów przy kolejnym wystąpieniu symbolu
  - różne wersje metody: PPMA, PPMB, PPMC itp.

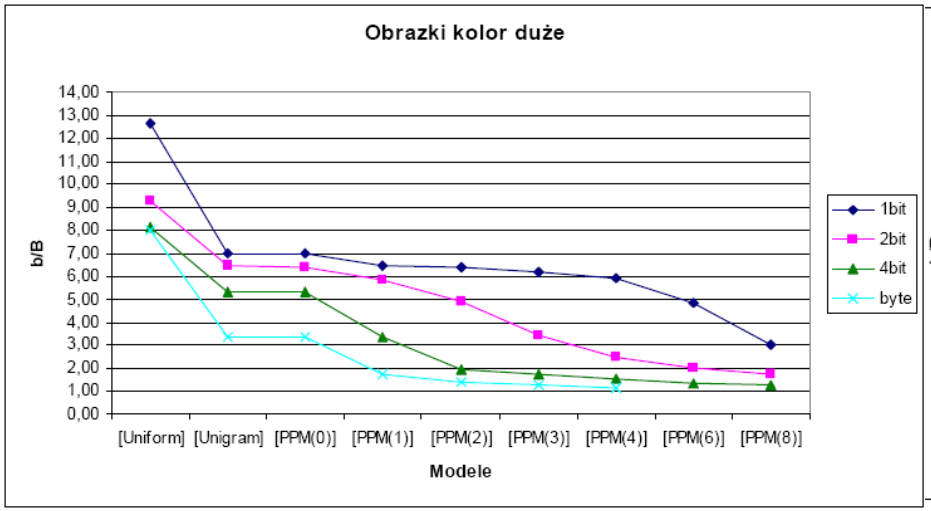
Przykład PPM (PPMC)

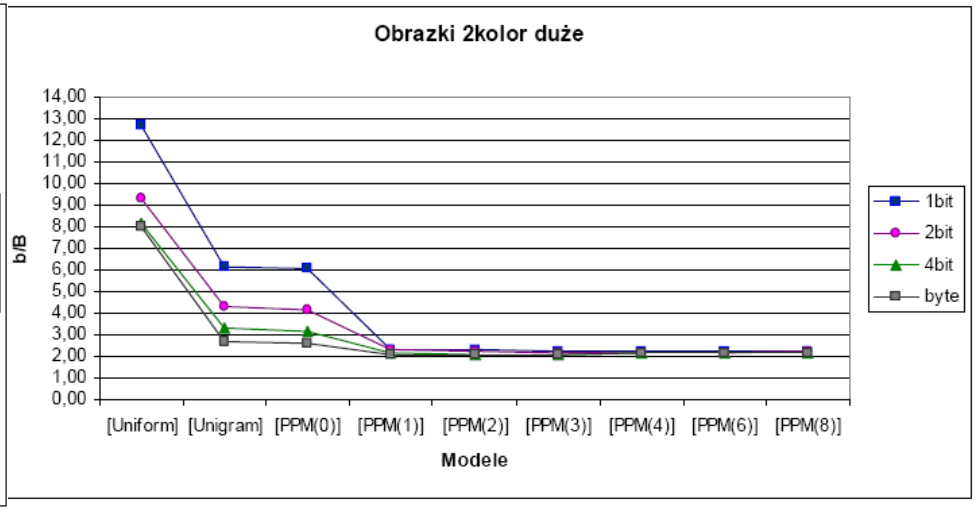
Kontekst rzędu 0			Kontekst rzędu 1			Kontekst rzędu 2		
s	waga	p_i	$\mathbf{c} \rightarrow s$	waga	p_i	$\mathbf{c} \rightarrow s$	waga	p_i
„A”	2	2/17	„A” → „R”	1	1/2	„AR” → „Y”	1	1/2
„R”	1	1/17	*PRZEŁĄCZ*	1	1/2	*PRZEŁĄCZ*	1	1/2
„Y”	2	2/17	„R” → „Y”	1	1/2	„RY” → „T”	1	1/2
„T”	2	2/17	*PRZEŁĄCZ*	1	1/2	*PRZEŁĄCZ*	1	1/2
„M”	1	1/17	„Y” → „T”	1	1/4	„YT” → „M”	1	1/2
„E”	1	1/17	„Y” → „K”	1	1/4	*PRZEŁĄCZ*	1	1/2
„K”	1	1/17	*PRZEŁĄCZ*	2	2/4	„TM” → „E”	1	1/2
PRZEŁĄCZ	7	7/17	„T” → „M”	1	1/4	*PRZEŁĄCZ*	1	1/2
			„T” → „Y”	1	1/4	„ME” → „T”	1	1/2
			PRZEŁĄCZ	2	2/4	*PRZEŁĄCZ*	1	1/2
			„M” → „E”	1	1/2	„ET” → „Y”	1	1/2
			PRZEŁĄCZ	1	1/2	*PRZEŁĄCZ*	1	1/2
			„E” → „T”	1	1/2	„TY” → „K”	1	1/2
			PRZEŁĄCZ	1	1/2	*PRZEŁĄCZ*	1	1/2
			„K” → „A”	1	1/2	„YK” → „A”	1	1/2
			PRZEŁĄCZ	1	1/2	*PRZEŁĄCZ*	1	1/2

Testy


Pliki text dla alfabetu 1 bajt
9,00
8,00
7,00
6,00
5,00
4,00
3,00
2,00
1,00
0,00
[Uniform] [Unigram] [PPM(0)] [PPM(1)] [PPM(2)] [PPM(3)] [PPM(4)] [PPM(6)] [PPM(8)]
Modele
< 1000
średnie
>400000

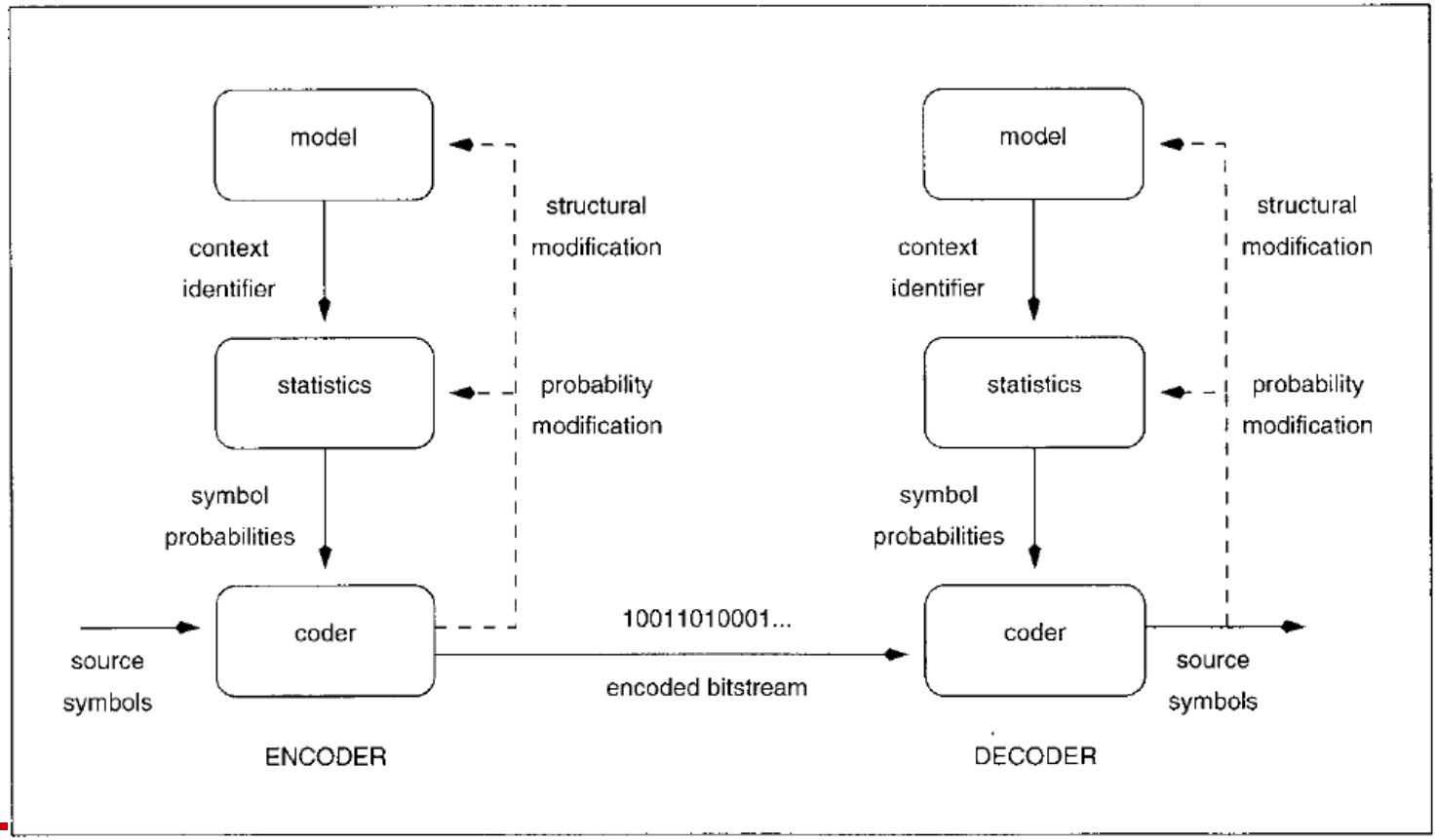

Wav duże
14,00
13,00
12,00
11,00
10,00
9,00
8,00
7,00
6,00
5,00
4,00
3,00
2,00
1,00
0,00
[Uniform] [Unigram] [PPM(0)] [PPM(1)] [PPM(2)] [PPM(3)] [PPM(4)] [PPM(6)] [PPM(8)]
Modele
1bit
2bit
4bit
byte


Obrazki kolor duże
b/B
14,00
13,00
12,00
11,00
10,00
9,00
8,00
7,00
6,00
5,00
4,00
3,00
2,00
1,00
0,00
[Uniform] [Unigram] [PPM(0)] [PPM(1)] [PPM(2)] [PPM(3)] [PPM(4)] [PPM(6)] [PPM(8)]
Modele
1bit
2bit
4bit
byte


Obrazki 2kolor duże
b/B
14,00
13,00
12,00
11,00
10,00
9,00
8,00
7,00
6,00
5,00
4,00
3,00
2,00
1,00
0,00
[Uniform] [Unigram] [PPM(0)] [PPM(1)] [PPM(2)] [PPM(3)] [PPM(4)] [PPM(6)] [PPM(8)]
Modele
1bit
2bit
4bit
byte

Inne pomysły

- wykorzystanie statystyk *a priori*
- przełączany alfabet: znaki, słowa, nie-słowa wymagają rzetelnego przełączania i uaktualniania linijek
- koncepcja podkreślająca modelowanie kontekstu:



PROBLEM ZŁOŻONOŚCI CZYLI BAC

Kodery BAC – recepta na złożoność obliczeniową

- Alfabet binarny upraszcza modyfikację przedziału kodu

  - kodowanie:

$$R_1^{(i-1)} = R^{(i-1)} p_1$$

a) jeśli kodujemy $s_i = a_1$ to $\pi^{(i)} = [D^{(i-1)}, D^{(i-1)} + R_1^{(i-1)})$; $R^{(i)} = R_1^{(i-1)}$

b) jeśli kodujemy $s_i = a_2$ to $\pi^{(i)} = [D^{(i-1)} + R_1^{(i-1)}, G^{(i-1)})$; $R^{(i)} = R^{(i-1)} - R_1^{(i-1)}$

  - dekodowanie:

a) jeśli $\mathcal{L} - D^{(i-1)} < R_1^{(i-1)}$ to dekodujemy $s_i = a_1$ oraz $\pi^{(i)} = [D^{(i-1)}, D^{(i-1)} + R_1^{(i-1)})$;

b) jeśli $\mathcal{L} - D^{(i-1)} \geq R_1^{(i-1)}$ to dekodujemy $s_i = a_2$ oraz $\pi^{(i)} = [D^{(i-1)} + R_1^{(i-1)}, G^{(i-1)})$.

- Nie rezerwuje się miejsca dla EOF

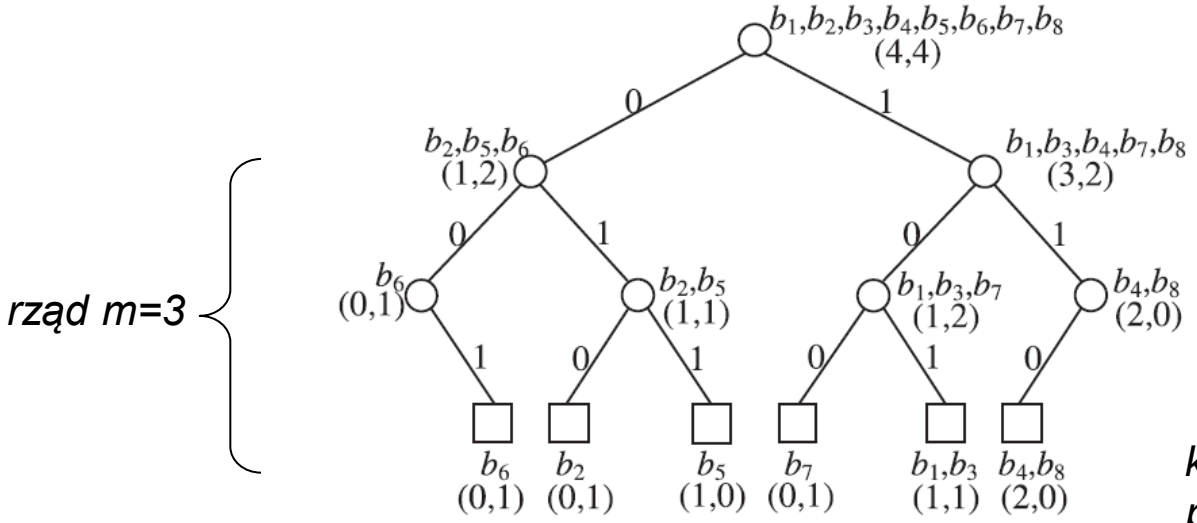
Metoda CTW (*context-tree weighting*)

- Ważenie prawdopodobieństw zamiast wyboru !

$$(P_S^{C_1} + P_S^{C_2})/2$$

- Dane: | kontekst *m* | kodowany blok | 10101100110 → 01100110|101

- Budowanie drzewa kontekstów binarnych (blokowe prawdopodobieństwo warunkowe)



Binarne drzewo kontekstów dla sekwencji źródłowej 01100110|101. *Przyjęto oznaczenie* (n_1, n_2) *liczby wystąpień odpowiednio zer i jedynek.*

Obliczanie prawdopodobieństw

- Blokowe, bezwzględne, a ważenie realizuje pamięć!
- Szacowanie prawdopodobieństw (alfabet binarny):

$$P_e(s = a_1) = \frac{n_1 + 1/2}{n_1 + n_2 + 1}$$

- Prawdopodobieństwo bloku: $P_e(n_1, n_2) = \frac{1/2 \cdot \ldots \cdot (n_1 - 1/2) \cdot 1/2 \cdot \ldots \cdot (n_2 - 1/2)}{1 \cdot 2 \cdot \ldots \cdot (n_1 + n_2)}$

$$P_e(n_1 + 1, n_2) = \frac{n_1 + 1/2}{n_1 + n_2 + 1} \cdot P_e(n_1, n_2)$$

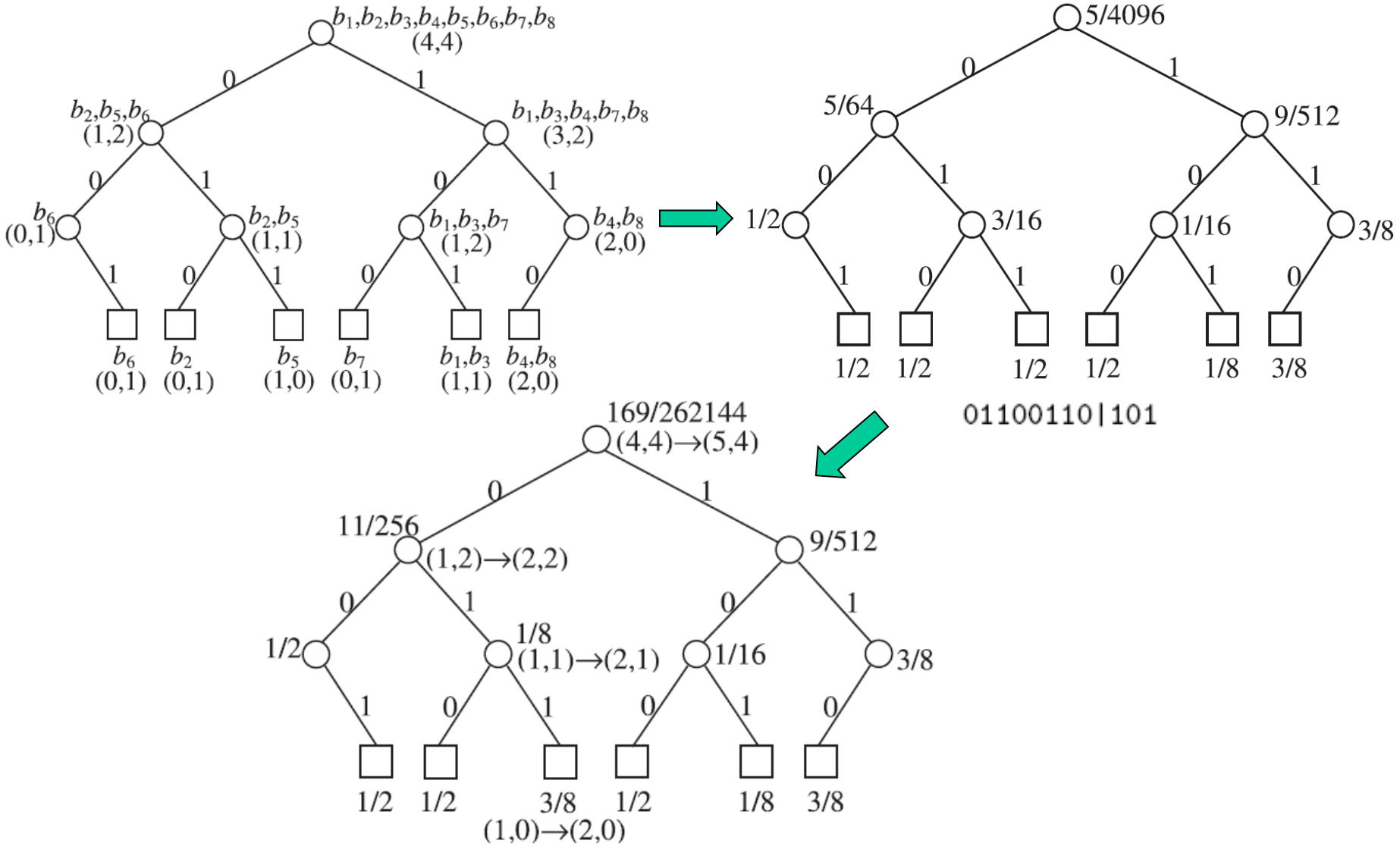
$$P_e(n_1, n_2 + 1) = \frac{n_2 + 1/2}{n_1 + n_2 + 1} \cdot P_e(n_1, n_2)$$

$n_1 \vert n_2$	0	1	2	3	4	5
0	1	1/2	3/8	5/16	35/128	63/256
1	1/2	1/8	1/16	5/128	7/256	21/1024
2	3/8	1/16	3/128	3/256	7/1024	9/2048
3	5/16	5/128	3/256	5/1024	5/2048	45/32768
4	35/128	7/256	7/1024	5/2048	35/32768	35/65536
5	63/256	21/1024	9/2048	45/32768	35/65536	63/262144

- Ważenie prawdopodobieństw

$$P_w^{\mathsf{w}} = \begin{cases} P_e(n_1^{\mathsf{w}}, n_2^{\mathsf{w}}), & g(\mathsf{w}) = m \\ \frac{P_e(n_1^{\mathsf{w}}, n_2^{\mathsf{w}}) + P_w^{\mathsf{w}0} P_w^{\mathsf{w}1}}{2}, & 0 \leq g(\mathsf{w}) < m \end{cases}$$

CTW - przykład



Wykorzystanie w BAC: drzewo wskazuje przedział kodowy

- **Idea**:

$$P(b_1 \dots b_{l-1}|\mathbf{c_b}^{(m)}) \rightarrow P(b_1 \dots b_{l-1}0|\mathbf{c_b}^{(m)}) \rightarrow \text{kodujemy } b_l \rightarrow b_1 \dots b_l$$

- Szacowanie (długości) przedziału kodu

$$P(b_1 \dots b_{l-1}|\mathbf{c_b}^{(m)}) = P(b_1 \dots b_{l-1}, b_l = 0|\mathbf{c_b}^{(m)}) + P(b_1 \dots b_{l-1}, b_l = 1|\mathbf{c_b}^{(m)})$$

- Jeśli symbol=1 to 2 razy przeliczamy drzewo, jeśli =0 - tylko raz (prawdopodobieństwo bloku długością przedziału kodowego)

$$R_1^{(i-1)} = R^{(i-1)} p_1$$

a) jeśli kodujemy $s_i = a_1$ to $\pi^{(i)} = [D^{(i-1)}, D^{(i-1)} + R_1^{(i-1)})$; $R^{(i)} = R_1^{(i-1)}$

b) jeśli kodujemy $s_i = a_2$ to $\pi^{(i)} = [D^{(i-1)} + R_1^{(i-1)}, G^{(i-1)})$; $R^{(i)} = R^{(i-1)} - R_1^{(i-1)}$

a) jeśli $\mathcal{L} - D^{(i-1)} < R_1^{(i-1)}$ to dekodujemy $s_i = a_1$ oraz $\pi^{(i)} = [D^{(i-1)}, D^{(i-1)} + R_1^{(i-1)})$;

b) jeśli $\mathcal{L} - D^{(i-1)} \geq R_1^{(i-1)}$ to dekodujemy $s_i = a_2$ oraz $\pi^{(i)} = [D^{(i-1)} + R_1^{(i-1)}, G^{(i-1)})$.

Implementacje BAC

- z DOL i GORA

  - PRZE = GORA − DOL + 1; R1 = PRZE ∗ C1/(C1+C2)
  - jeśli $s_i = a_1$ to GORA = DOL + R1;
    wpp (tj. $s_i = a_2$) DOL = DOL + R1;

  koder

  - PRZE = GORA − DOL + 1; R1 = PRZE ∗ C1/(C1+C2)
  - jeśli KOD − DOL < R1, to dekodujemy X=a_1 oraz GORA = DOL + R1;
    wpp to dekodujemy X=a_2 oraz DOL = DOL + R1;

  dekoder

- z DOL i PRZE

  - R1 = PRZE ∗ C1/(C1+C2);
  - jeśli $s_i = a_1$, to PRZE = R1;
    wpp PRZE = PRZE − R1; DOL = DOL + R1;

  koder

  - R1 = PRZE ∗ C1/(C1+C2)
  - jeśli KOD − DOL < R1 to X=a_1; PRZE = R1;
    wpp X=a_2; PRZE = PRZE − R1; DOL = DOL + R1;

  dekoder

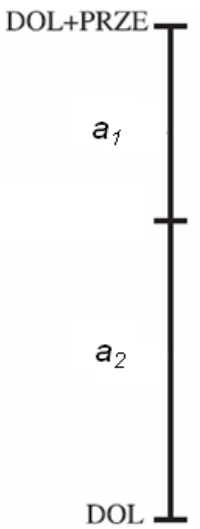
Implementacje BAC

Porządkowanie linii prawdopodobieństw

- *koder z przedziałami z a_1* na dole (tak było poprzednio)
  - $\texttt{R1} = \texttt{PRZE} * \texttt{C1/(C1+C2)};$
  - jeśli $s_i = a_1$, to $\texttt{PRZE} = \texttt{R1};$
    wpp $\texttt{PRZE} = \texttt{PRZE} - \texttt{R1}; \quad \texttt{DOL} = \texttt{DOL} + \texttt{R1};$

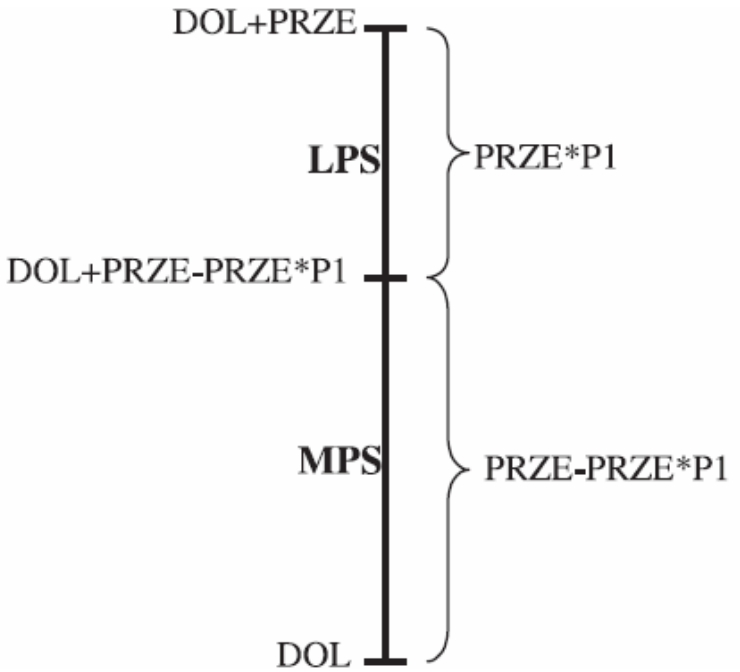
- *poprawa statystyki modelu: ponieważ zaokrąglanie w dół* R1

- a_1 *na górze jako mniej prawdopodobny*



- $\texttt{R1} = \texttt{PRZE} * \texttt{C1/(C1+C2)}; \quad \texttt{R2} = \texttt{PRZE} - \texttt{R1};$
- jeśli $s_i = a_1$, to $\texttt{PRZE} = \texttt{R1}, \texttt{DOL} = \texttt{DOL} + \texttt{R2};$
  wpp $\texttt{PRZE} = \texttt{R2}.$

- *alfabet LPS i MPS*



Inne wskazówki implementacyjne

- precyzja i normalizacja przedziału
  - jeśli precyzja rejestru sumy zliczeń wynosi *c,* to precyzja DOL i GORA powinna być b ≥ c+2 lub precyzja PRZE ≥ c+1
  - kodowanie i niedomiar: na pozycji b-1 w PRZE znika 1
  - różnorodność stosowanych rozwiązań:
    - w Q koderze b=13 bitów, w QM b=16 bitów, w koderze kształtu w MPEG-4 b=32 bity

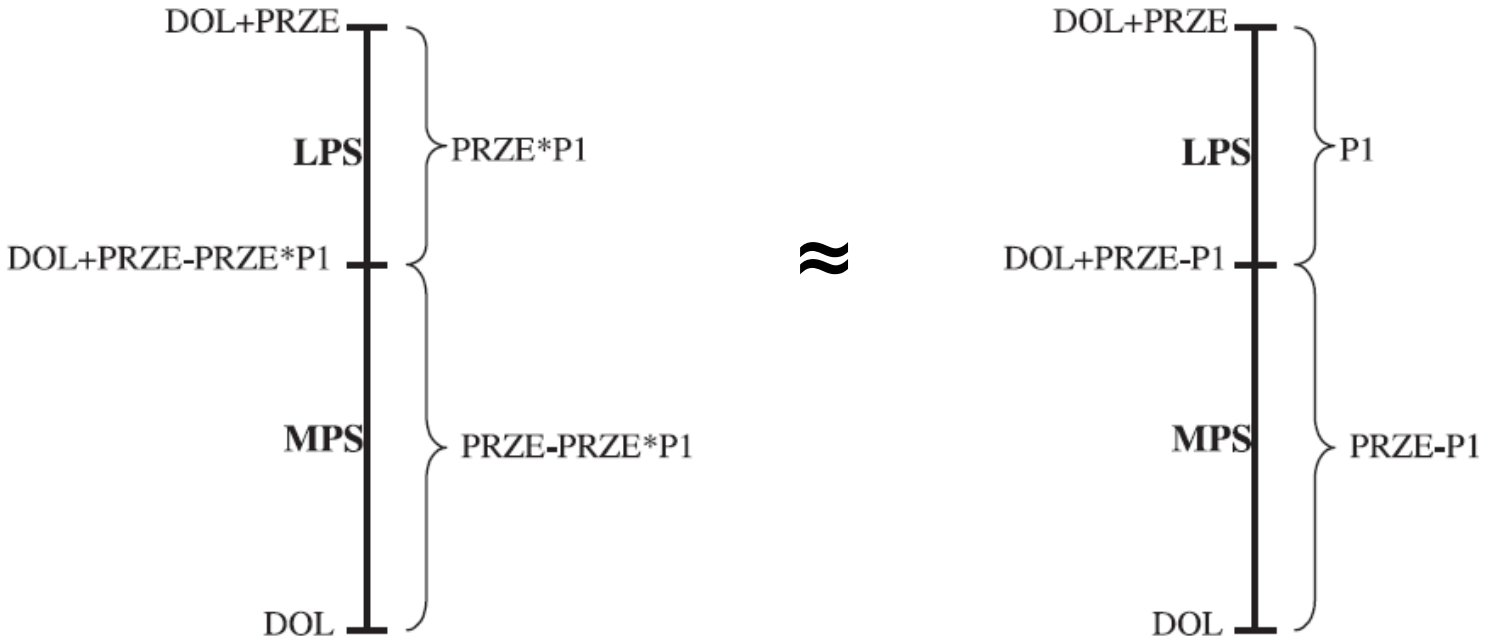
Przykładowa realizacja BAC

```
void bin_koder_aryt(zbior,c1,c2,x)
/* c1, c2 - częstości zliczeń symboli a1 i a2 */
/* x - kodowana wartość symbolu */
{

....
    r1=prze * c1/(c1+c2);
    r2=prze - r1;
    if(x == 'a1') {
       prze=r1;
       dol=dol+r2;
    } else
       prze=r2;
    normalizuj_kod(zbior,dol,prze);

....
....

}

...

#define POL 1 << (b-1)
#define CWIERC 1 << (b-2)
normalizuj_kod(zbior,dol,prze)
{
   while (prze <= CWIERC) {
      if (dol >= POL) {
         WyslijBit(zbior,1);
         while (niedomiar>0) {
            WyslijBit(zbior,0);
            niedomiar--;
         }
         dol -= POL;
      } else if (dol+prze <= POL)
         WyslijBit(zbior,0);
         while (niedomiar>0) {
            WyslijBit(zbior,1);
            niedomiar--;
         }
      else {
         niedomiar++;
         dol -= CWIERC;
      }
      dol <<= 1;
      prze <<= 1;
   }
}
```

```
int bin_dekoder_aryt(zbior,int c1,int c2)
/* c1, c2 -- częstości zliczeń symboli a1 i
/* x -- dekodowana wartość symbolu
{
 int x;

....

   r1=prze * c1/(c1+c2);
   r2=prze - r1;
   if(kod - dol >= r2) {
      prze=r1;
      dol=dol+r2;
      x='a1';
   } else {
      prze=r2;
       x='a2';
    }
   normalizuj_dek(zbior,dol,prze);
   return(x);
}

...

normalizuj_dek(zbior,dol,prze)
    while (prze <= CWIERC) {
      if (dol+prze <= POL) ;
      else if (dol >= POL) {
         dol -= POL;
         kod -= POL;
      } else {
         dol -= CWIERC;
         kod -= CWIERC;
      dol <<= 1;
      prze <<= 1;
      kod <<= 1;
      kod+=PobierzBit(zbior);
    }
```

Szybki BAC (FBAC)

- mnożenie zastąpiono aproksymacją
  - zamiast [0, 1) mamy [0, 1,5)
  - 0,75 ≤ PRZE <1,5
  - PRZE ≈ 1, czyli PRZE*P1 ≈ P1
  - LUT i wektory przejść

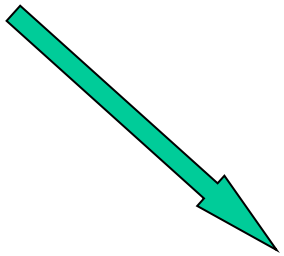


przybliżenia (upraszczanie obliczeń)

PRZE ≈ 1

Koncepcja FBAC

- R1 = PRZE * C1/(C1+C2); R2 = PRZE − R1;
- jeśli $s_i = a_1$, to PRZE = R1, DOL = DOL + R2;
  wpp PRZE = R2.



- R_LPS=Tablica_R[ind]; *LUT*
- R_MPS=PRZE-R_LPS;
- if(X==LPS) {
  PRZE = R_LPS;
  DOL = DOL + R_MPS;
  ind = LPS[ind]; *wektor przejść*
  } else {
  PRZE = R_MPS;
  ind = MPS[ind];
  }