

3138

Spis Treści

88-221C SIN

Politechnika Szczecińska

Wydział Informatyki

**„Metoda kompresji danych z wykorzystaniem
zmiennych alfabetów symboli transmisyjnych”**

PRACA DOKTORSKA

Autor: mgr inż. Grzegorz Ulacha

Promotor: dr hab. inż. Ryszard Stasiński, prof. PS

Szczecin, 2004r.

41442

DISS-33

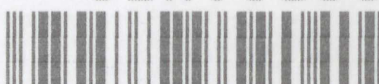
004.6:519.72 (043)



SIN. 3199

Biblioteka Główna
Politechniki Szczecińskiej

SIN. 3199



015-003199-00-0

D. 374 / 2005

Spis Treści

SPIS TABEL	4
SPIS RYSUNKÓW	7
SPIS WAŻNIEJSZYCH SYMBOLI I OZNACZEŃ UŻYWANYCH W PRACY	8
1. WPROWADZENIE	11
1.1. CEL, TEZA ORAZ ZAKRES PRACY	11
1.2. PODSTAWOWE OKREŚLENIA UŻYWANE W PRACY	13
1.3. PODSTAWOWE CECHY KODÓW	13
2. CHARAKTERYSTYKA PODSTAWOWYCH KODÓW ENTROPIJNYCH – STAN AKTUALNY	18
2.1. BINARNY KOD HUFFMANA	20
2.2. BINARNY KOD HUFFMANA O MINIMALNEJ WARIANCJI	23
2.3. KOD DWUFAZOWY	24
2.4. WIELOWARTOŚCIOWY KOD HUFFMANA	29
2.5. ROZSZERZONY KOD HUFFMANA	31
2.6. KOD TUNSTALLA I JEGO MODYFIKACJE	34
2.7. KOD ARYTMETYCZNY I INNE KODY ENTROPIJNE	36
2.8. METODY EFEKTYWNEGO ZAPISU PARAMETRÓW KOMPRESJI	38
3. EFEKTYWNA REPREZENTACJA SYMBOLI WIELOWARTOŚCIOWYCH	42
3.1. ANALIZA REPREZENTACJI SYMBOLI WIELOWARTOŚCIOWYCH	42
3.2. ZASTOSOWANIE KODU DWUFAZOWEGO DO KOMPRESJI LICZB B	43
3.3. PROBLEM DOBORU DŁUGOŚCI BLOKÓW ELEMENTÓW R -NARNYCH (Q)	44
3.4. ANALIZA EFEKTYWNOŚCI I SPRAWNOŚCI KONWERSJI	46
3.5. ALGORYTM WYZNACZANIA WZORCOWYCH DŁUGOŚCI BLOKÓW (Q) ELEMENTÓW R -NARNYCH	47
4. MIESZANE KODY HUFFMANA	52
4.1. PODSTAWY KONSTRUKCJI MIESZANYCH KODÓW HUFFMANA	52
4.2. OGRANICZENIA KONSTRUKCYJNE KODU MIESZANEGO	57
4.3. PODSTAWOWE WŁASNOŚCI MIESZANYCH KODÓW HUFFMANA	58
4.4. DOLNE OGRANICZENIE ŚREDNIEJ DŁUGOŚCI SŁOWA KODU HUFFMANA	60

4.5.	ZASTOSOWANIE LICZB PIERWSZYCH DO KONSTRUKCJI BUFORÓW R -NARNYCH.....	61
4.6.	TESTY EFEKTYWNOŚCI	63
4.7.	PRZYKŁADY	66
5.	KONSTRUKCJA OPTYMALNYCH MIESZANYCH KODÓW HUFFMANA	71
5.1.	PODSTAWY	71
5.1.1.	Analiza konstrukcji kodów mieszanych dla $n = 2$ oraz $n = 3$	71
5.1.2.	Analiza konstrukcji kodów mieszanych dla $n = 4$	73
5.1.3.	Analiza konstrukcji kodów mieszanych dla $n = 5$	74
5.2.	KONSTRUOWANIE OPTYMALNYCH KODÓW MIESZANYCH	80
5.3.	METODA INTEGRALNYCH PODDRZEW	84
5.4.	PODDRZEWIA INTEGRALNE BINARNIE	87
6.	OPTYMALNE I SUBOPTYMALNE MIESZANE KODY HUFFMANA O KRÓTKIM CZASIE KONSTRUKCJI DRZEWA KODOWEGO	94
6.1.	KOD BINARNO-TERNARNY	94
6.2.	SUBOPTYMALNY KOD DWUPOZIOMOWY	97
6.3.	METODA KOLEJNYCH UŚCIŚLEŃ	101
6.4.	KOD DWUPOZIOMOWY Z WYKORZYSTANIEM METODY KOLEJNYCH UŚCIŚLEŃ	104
6.5.	SUBOPTYMALNY KOD MIESZANY ZE STAŁYM OPÓŹNIENIEM	105
6.6.	OPTYMALNY I SUBOPTYMALNY KOD MIESZANY Z MAŁYM OPÓŹNIENIEM	111
7.	DYNAMICZNE MIESZANE KODY HUFFMANA	115
7.1.	WŁASNOŚCI DYNAMICZNEGO KODU HUFFMANA	115
7.2.	ALGORYTMY MIESZANYCH DYNAMICZNYCH KODÓW HUFFMANA	116
7.3.	PRZYKŁADY TWORZENIA DYNAMICZNEGO KODU HUFFMANA	118
7.4.	TESTY PORÓWNAWCZE DLA STATYCZNYCH I DYNAMICZNYCH METOD ENTROPIJNYCH	121
8.	ZASTOSOWANIA MIESZANYCH KODÓW HUFFMANA	128
8.1.	KOMPRESJA ELEMENTÓW DC W STANDARDZIE JPEG	128
8.2.	ZASTOSOWANIE KODU BINARNO-TERNARNEGO DO BEZSTRATNEJ KOMPRESJI OBRAZÓW	130
8.3.	ALGORYTM SZYBKIEGO KONWERTERA BINARNO-TERNARNEGO	131
8.4.	ZASTOSOWANIE KODÓW MIESZANYCH DO ZWIĘKSZENIA EFEKTYWNOŚCI KODU GOLOMBA	132

8.4.1.	Własności podstawowego kodu Golomba.....	132
8.4.2.	Analiza własności liczby grupowej m	136
8.4.3.	Modyfikacja kodu Golomba	138
8.4.4.	Przykłady zmodyfikowanego kodu Golomba	140
9.	PODSUMOWANIE.....	145
DODATEK A	- WYPROWADZENIA WZORÓW (8.7), (8.9) ORAZ (8.15)	148
DODATEK B	- KODY ŹRÓDŁOWE.....	151
LITERATURA		155

Spis tabel

2.1	Przykładowa tabela kodu Huffmana	21
2.2	Kolejne kroki budowy drzewa kodu Huffmana	22
2.3	Kod dwufazowy dla $n = 7$	25
2.4	Efektywność minimalna dla $t \leq 16$	28
2.5	Kod Huffmana z przykładu 2.2	30
2.6	Kod Huffmana z przykładu 2.3	32
2.7	Kod Huffmana dla $k = 2$	33
2.8	Kod Huffmana dla $k = 3$	33
2.9	Kody Tunstalla i Tunstalla-Huffmana z przykładu 2.3.....	35
3.1	Parametry dla liczby B złożonej z $q < 10$ cyfr ternarnych.....	45
3.2	Parametry dla liczby B złożonej z $q < 200000$ cyfr, przy $r = 3$	49
3.3	Parametry dla liczby B złożonej z $q < 100000$ cyfr, przy $r = 5$	50
3.4	Parametry dla liczby B złożonej z $q < 100000$ cyfr, przy $r = 7$	50
3.5	Parametry dla przykładowych wzorcowych q przy $r < 30$	51
4.1	Kody Huffmana z przykładu 2.2	53
4.2	Kody Huffmana z przykładu 4.1	55
4.3	Kody Huffmana z przykładu 4.2 z parametrami $r_1 = 3, r_2 = 5$	56
4.4	Kod mieszany z przykładu 4.3	58
4.5	Wartości p_m , powyżej których efektywność kodu Huffmana jest mniejsza od 95%.....	61
4.6	Testowa klasa kodów mieszanych	64
4.7	Wyniki testów kodów mieszanych.....	64
4.8	Kody Huffmana z przykładu 4.4	66
4.9	Kody Huffmana z przykładu 4.5	67
4.10	Kody Huffmana z przykładu 4.6	69
5.1	Typy drzew binarnego kodu Huffmana dla $n = 4$	73
5.2	Typy drzew mieszanego kodu Huffmana dla $n = 4$	74
5.3	Przykłady mieszanego kodu Huffmana dla $n = 4$	74
5.4	Typy drzew binarnego kodu Huffmana dla $n = 5$	76
5.5	Typy drzew mieszanego kodu Huffmana dla $n = 5$	77
5.6	Zestaw warunków dla drzewa binarnego typu 2I, przy których dany typ drzewa mieszanego daje najkrótszą średnią długość słowa kodowego	77
5.7	Przykłady mieszanego kodu Huffmana dla $n = 5$ spełniające warunki 2I	78
5.8	Zestaw warunków dla drzewa binarnego typu 2II, przy których dany typ drzewa mieszanego daje najkrótszą średnią długość słowa kodowego	78
5.9	Przykłady mieszanego kodu Huffmana dla $n = 5$ spełniające warunki 2II.....	78

5.10 Zestaw warunków dla drzewa binarnego typu 2III, przy których dany typ drzewa mieszane daje najkrótszą średnią długość słowa kodowego.....	78
5.11 Przykłady mieszane kodu Huffmana dla $n = 5$ spełniające warunki 2III.....	79
5.12 Zestaw warunków dla drzewa binarnego typu 2IV, przy których dany typ drzewa mieszane daje najkrótszą średnią długość słowa kodowego.....	79
5.13 Przykłady mieszane kodu Huffmana dla $n = 5$ spełniające warunki 2IV.....	79
5.14 Liczba wszystkich drzew mieszanych dla $n \leq 25$	81
5.15 Kolejne kroki poszukiwania optymalnego mieszane kodu Huffmana dla danych z przykładu 4.1 ($n = 8$).....	83
5.16 Zestaw rozwiązań równania (5.7).....	84
5.17 Optymalny kod mieszany z przykładu 5.2 w klasie $r_0 = 2, r_1 = 3, r_2 = 7, q_1 = 41, q_2 = 16$	86
5.18 Optymalny kod mieszany z przykładu 5.3 w klasie $r_0 = 2, r_1 = 3, r_2 = 5, q_1 = 5, q_2 = 3$	89
5.19 Optymalny kod mieszany z przykładu 5.4 w klasie $r_0 = 2, r_1 = 3, r_2 = 5, q_1 = 41, q_2 = 31$	92
6.1 Wyniki testów kodów binarno-ternarnych.....	95
6.2 Mieszany kod Huffmana dla przykładu 6.1	99
6.3 Suboptymalny mieszany kod Huffmana dla źródła z przykładu 4.6	103
6.4 Wyniki testów dla suboptymalnych kodów mieszanych.....	105
6.5 Dwa optymalne kody mieszane dla źródła z przykładu 6.2.....	108
6.6 Procentowy udział niebinarnych korzeni w kodzie mieszanym pierwszego rzędu	108
6.7 Procentowy udział ternarnych korzeni w kodzie binarno-ternarnym pierwszego rzędu dla $q = 41$	110
6.8 Procentowy udział niebinarnych korzeni w optymalnym kodzie mieszanym	111
6.9 Dwubuforowe kody mieszane o efektywności wyższej, niż dla binarnego kodu Huffmana (dla danych z przykładu 4.1).....	113
7.1 Liczność symboli dla przykładu 7.1.....	122
7.2 Wyniki testów dla przykładu 7.1.....	123
7.3 Liczba symboli r_i -narnych w kodzie wynikowym dla przykładu 7.1.....	123
7.4 Wyniki testów dla przykładu 7.2.....	124
7.5 Liczba symboli r_i -narnych w kodzie wynikowym dla przykładu 7.2.....	124
7.6 Wyniki testów dla przykładu 7.3.....	124
7.7 Liczba symboli r_i -narnych w kodzie wynikowym dla przykładu 7.3.....	125
7.8 Liczność symboli dla przykładu 7.4.....	125
7.9 Wyniki testów dla przykładu 7.4.....	126
7.10 Liczba symboli r_i -narnych w kodzie wynikowym dla przykładu 7.4.....	126
8.1 Przykładowa tabela kodowa dla współczynników DC proponowana przez JPEG.....	129
8.2 Proponowane tabele kodu Huffmana.....	129
8.3 Średnia liczba bitów na piksel dla czterech standardowych obrazów testowych	131
8.4 Kod Golomba z przykładu 8.1	138

Spis rysunków

2.1 Drzewa kodowe z przykładu 2.1 (kod II) 22

2.2 Zmniejszenie liczby równoprawdopodobnych symboli źródła z $n = 4$ do $n = 2$ 26

2.3 Zależność procentowej efektywności w funkcji n dla $t = 8$ 27

2.4 Drzewo kodu ternarnego z przykładu 2.2 31

4.1 Drzewo kodu binarno-ternarnego z przykładu 4.1 33

4.2 Drzewo mieszanego kodu Huffmana z przykładu 4.1 56

4.3 Porównanie średnie wartości prawdopodobieństwa wystąpienia symboli dla źródła z $n = 24$ symbolach 63

4.4 Drzewo kodu binarno-ternarnego z przykładu 4.4 66

4.5 Drzewo kodu mieszanego (ternarnego) z przykładu 4.5 68

4.6 Drzewo mieszanego kodu Huffmana z przykładu 4.6 69

4.7 Drzewo kodu binarno-ternarnego z przykładu 4.6 70

5.1 Binarny(a) i ternarny(b) kod Huffmana dla $n = 3$ 72

5.2 Procentowa efektywność binarnego (linia przerywana) i ternarnego (linia ciągła) kodu Huffmana w funkcji p 72

5.3 Dwa typy binarnych kodów Huffmana dla $n = 4$ 73

5.4 Dwa typy mieszanych kodów Huffmana dla $n = 4$ 73

5.5 Cztery typy binarnych kodów Huffmana dla $n = 5$ 75

5.6 Sześć typów mieszanych kodów Huffmana dla $n = 5$ 76

5.7 Pierwsza integralna poddrzewo z przykładu 5.2 86

5.8 Nadmiarowa część drzewa z przykładu 5.2 87

5.9 Poddrzewo binarne dla $n_0 = 1$ integralnych symboli z przykładu 5.3 90

5.10 Drzewo optymalnego kodu mieszanego z przykładu 5.3 90

5.11 Drzewo optymalnego kodu mieszanego z przykładu 5.4 93

6.1 Drzewo mieszanego kodu Huffmana z przykładu 6.1 100

6.2 Drzewo suboptymalnego mieszanego kodu Huffmana z przykładu 4.6 103

6.3 Kolejne etapy zamiany drzewa kodu mieszanego I na kod mieszany II 107

8.1 Zależność procentowej efektywności kodu Golomba w funkcji p 133

8.2 Zależność średniej L_0 kodu Golomba w funkcji p 135

8.3 Porównanie procentowej efektywności kodów Golomba w funkcji p . Kod podstawowy (linia przerywana) i zmodyfikowany (linia ciągła) 141

8.4 Zależność L_0 (linia przerywana), L_0 dla $t = 3$, $q = 41$ (linia ciągła), H_0 (linia kropkowana) w funkcji p 142

8.5 Zależność L_0 (linia przerywana), L_0 dla $t = 5$, $q = 31$ (linia ciągła), H_0 (linia kropkowana) w funkcji p 143

Spis rysunków

2.1	Drzewo kodowe z przykładu 2.1 (kod II).....	22
2.2	Zmniejszenie liczby równoprawdopodobnych symboli źródła z $n = 4$ do $n = 3$	26
2.3	Zależność procentowej efektywności w funkcji n dla $t = 8$	27
2.4	Drzewo kodu ternarnego z przykładu 2.2.....	31
4.1	Drzewo kodu binarno-ternarnego z przykładu 4.1.....	55
4.2	Drzewo mieszanego kodu Huffmana z przykładu 4.1.....	56
4.3	Posortowane średnie wartości prawdopodobieństw wystąpienia symboli dla źródła o $n = 24$ symbolach	65
4.4	Drzewo kodu binarno-ternarnego z przykładu 4.4.....	66
4.5	Drzewo kodu mieszanego (ternarnego) z przykładu 4.5	68
4.6	Drzewo mieszanego kodu Huffmana z przykładu 4.6.....	69
4.7	Drzewo kodu binarno-ternarnego z przykładu 4.6.....	70
5.1	Binarny(a) i ternarny(b) kod Huffmana dla $n = 3$	72
5.2	Procentowa efektywność binarnego (linia przerywana) i ternarnego (linia ciągła) kodu Huffmana w funkcji p_3	72
5.3	Dwa typy binarnych kodów Huffmana dla $n = 4$	73
5.4	Dwa typy mieszanych kodów Huffmana dla $n = 4$	73
5.5	Cztery typy binarnych kodów Huffmana dla $n = 5$	75
5.6	Sześć typów mieszanych kodów Huffmana dla $n = 5$	76
5.7	Pierwsze integralne poddrzewo z przykładu 5.2	86
5.8	Nadrzędna część drzewa z przykładu 5.2	87
5.9	Poddrzewo binarne dla $n_0 = 7$ integralnych symboli z przykładu 5.3.....	90
5.10	Drzewo optymalnego kodu mieszanego z przykładu 5.3	90
5.11	Drzewo optymalnego kodu mieszanego z przykładu 5.4	93
6.1	Drzewo mieszanego kodu Huffmana z przykładu 6.1.....	100
6.2	Drzewo suboptymalnego mieszanego kodu Huffmana z przykładu 4.6.....	103
6.3	Kolejne etapy zamiany drzewa kodu mieszanego I na kod mieszany II	107
8.1	Zależność procentowej efektywności kodu Golomba w funkcji p	135
8.2	Zależność średniej L_G kodu Golomba w funkcji p	135
8.3	Porównanie procentowej efektywności kodów Golomba w funkcji p . Kod podstawowy (linia przerywana) i zmodyfikowany (linia ciągła).....	141
8.4	Zależność L_G (linia przerywana), L_G dla $t = 3$, $q = 41$ (linia ciągła), H_{01} (linia kropkowana) w funkcji p	142
8.5	Zależność L_G (linia przerywana), L_G dla $t = 5$, $q = 31$ (linia ciągła), H_{01} (linia kropkowana) w funkcji p	143

Spis ważniejszych symboli i oznaczeń używanych w pracy

- Domknięty przedział a liczb całkowitych od b do c oznaczany jest jako: $a = \overline{b;c}$.
- Ze względu na stosowane w pracy cyfry r -wartościowych kodów (r -narnych) ich wartościowości oznaczane będą w indeksie dolnym, np. 3_5 oznacza cyfrę 3 w kodzie pięciowartościowym. W przypadku użycia kodu mieszanego (w którym występują różne wartościowości r_i) przy przedstawianiu średniej długości słowa kodowego i efektywności pojawiać się będzie w indeksie dolnym oznaczenie *mix* (lub zestaw użytych w kodzie wartościowości), np. efektywność kodu binarno-ternarnego $E_{mix} = E_{2,3}$. W pracy przyjęto kropkę „.” jako symbol oddzielający część ułamkową od części całkowitej w liczbach dziesiętnych.

- Symbol $\lfloor x \rfloor$ oznacza największą liczbę całkowitą nie większą od x .
- Symbol $\lceil x \rceil$ oznacza najmniejszą liczbę całkowitą nie mniejszą od x .
- Wykaz pozostałych ważniejszych symboli z rozdziałów od 1 do 7:

B – liczba złożona z q cyfr r -narnych – patrz wzór (3.5);

$b(r)$ – liczba bitów niezbędna do reprezentacji cyfry r -narnej – patrz wzór (3.1);

$\underline{b}(r)$ – liczba bitów niezbędna do reprezentacji cyfry r -narnej skompresowanej kodem dwufazowym – patrz wzór (3.3);

$\bar{b}_q(r)$ – średnia liczba bitów niezbędna do reprezentacji pojedynczej cyfry r -narnej zawartej w q cyfrowej liczbie B – patrz wzór (3.7);

$\underline{b}_q(r)$ – średnia liczba bitów niezbędna do reprezentacji pojedynczej cyfry r -narnej zawartej w q cyfrowej liczbie B skompresowanej kodem dwufazowym – patrz wzór (3.9);

D – liczba możliwych do uzyskania mieszanych drzew Huffmana – patrz wzór (5.5);

$D(r)$ – liczba możliwych do uzyskania dwubuforowych (binarno- r -narnych) drzew Huffmana – patrz wzór (6.13);

E – procentowa efektywność kompresji – patrz wzory (1.7), (3.2);

$\hat{E}$ – efektywność kompresji – patrz wzór (1.6);

$E_{q(r)}$ – procentowa efektywność kompresji liczby B złożonej z q cyfr r -narnych – patrz wzór (3.8);

$\underline{E}_{q(r)}$ – procentowa efektywność kompresji – patrz wzór (3.10);

$g(r)$ – suma prawdopodobieństw p_{w_i} wszystkich n_r węzłów r_i -narnych występujących w mieszanym drzewie kodowym – patrz wzór (4.1);
 $F(r)$ – wartość ciągu Fibonacciego – patrz wzory (6.4), (6.7);
 $H(S)$ – entropia bezpamięciowego źródła S – patrz wzór (1.2);
 K – rozmiar pliku poddawanego kompresji;
 l_i – długość i -tego słowa kodowego wyrażona w bitach;
 L_{calc} – wynikowa długość skompresowanego pliku wyrażona w bajtach – patrz wzór (2.20);
 L_F – wynikowa długość całego strumienia po kompresji wyrażona w bitach – patrz wzór (2.1);
 L_{mix} – średnia długość słowa kodu mieszanego – patrz wzór (4.4);
 L_{sr} – średnia długość słowa kodowego – patrz wzory (1.4), (2.19);
 n – liczba symboli źródła S ;
 N – liczba symboli poddrzewa S ;
 N_j – liczba symboli kodu Tunstalla – patrz punkt 2.6;
 N_k – liczba symboli rozszerzonego kodu Huffmana – patrz punkt 2.5;
 p_i – prawdopodobieństwo i -tego symbolu źródła S ;
 P – zestaw prawdopodobieństw symboli źródła S ;
 $p(r)$ – prawdopodobieństwo wystąpienia symbolu r_i -narnego w strumieniu wyjściowym kodu mieszanego – patrz wzór (4.2);
 q – liczba cyfr r -narnych tworzących liczbę B ;
 r – wartościowość kodu;
 R – redundancja kodu – patrz wzór (1.5);
 r_w – skrótowy zapis konstrukcji mieszanego kodu Huffmana – patrz punkt 4.1;
 S – zbiór elementów źródła;
 $S^{(k)}$ – zbiór elementów rozszerzonego źródła – patrz punkt 2.5;
 $T(r)$ – opóźnienie średnie kodu mieszanego – patrz wzór (6.12);
 $U_{q(r)}$ – procentowa sprawność konwersji liczby B złożonej z q cyfr r -narnych – patrz wzór (3.11);
 w_i – i -ty węzeł drzewa kodowego;
 W – procentowa waga zysku – patrz wzór (1.10);
 $w(r)$ – waga cyfry r -narnej w danym kodzie mieszanym – patrz wzór (4.3);
 x_i – i -ty symbol źródła S ;

- i, j, k, m – pomocnicze zmienne (ich znaczenie zależy od kontekstu danego zagadnienia);
- Wykaz symboli związanych z kodem Golomba opisywanym w punkcie 8.4 (niektóre symbole mogą mieć inne znaczenie niż we wcześniejszej części rozprawy, co wynika z zachowania spójności z literaturą dotyczącą kodów Golomba):

E_G – procentowa efektywność kodu Golomba – patrz wzór (8.8);

$G(n)$ – wartość prawdopodobieństwa n -tego symbolu źródła o rozkładzie geometrycznym – patrz wzór (8.1);

L_G – średnia liczba bitów potrzebna do zakodowania jednego bitu źródła binarnego – patrz wzór (8.7);

$L_{\underline{G}}$ – średnia liczba bitów potrzebna do zakodowania jednego bitu źródła binarnego przy użyciu zmodyfikowanego kodu Golomba – patrz wzór (8.15);

$L_{\underline{\underline{G}}}$ – średnia liczba bitów potrzebna do zakodowania jednego bitu źródła binarnego przy użyciu zmodyfikowanego kodu Golomba wykorzystującego kompresję kodem dwufazowym – patrz wzór (8.21);

m – liczba grupowa kodu Golomba – patrz wzór (8.3);

n – numer kodowanego symbolu (nieujemna liczba całkowita).

1. Wprowadzenie

W systemach informatycznych istotnym problemem jest sposób efektywnego przechowywania i transmisji danych. Należy dążyć do minimalizacji kosztów użytkowania systemów informatycznych, np. dzięki stosowaniu wydajnych technik kompresji danych. Ogólnie metody kompresji danych można podzielić na dwa główne typy: *stratną* i *bezstratną* (odwracalną). **Tematem niniejszej rozprawy doktorskiej jest:**

„Metoda kompresji danych z wykorzystaniem zmiennych alfabetów symboli transmisyjnych”.

Zaproponowana metoda dotyczy wyłącznie bezstratnej kompresji danych, choć w wielu algorytmach stratnej kompresji, jako jeden z etapów, często wykorzystuje się metodę kompresji bezstratnej.

1.1. Cel, teza oraz zakres pracy

Opracowana metoda kompresji danych jest rozwinięciem binarnego kodu Huffmana. **Celem pracy jest:**

„Opracowanie algorytmu kompresji danych opartego na mieszanych kodach Huffmana, wykorzystujących jednocześnie zastosowanie wielu kodów r -narnych”.

Nowością tej metody jest łączne wykorzystanie zalet binarnych i wielowartościowych kodów Huffmana, poprzez użycie w trakcie kodowania wielu różnych symboli r -narnych. Do uzyskania wysokiej efektywności zapisu cyfr r -narnych wykorzystywana jest ich reprezentacja w postaci liczb złożonych z q cyfr r -narnych zwanych w pracy liczbami B , oraz dalsza kompresja tych liczb przy użyciu kodu dwufazowego. Proponowana przez autora praktyczna realizacja metody kompresji określana jest mianem *mieszanego kodu Huffmana* ze względu na duże podobieństwa do binarnego kodu Huffmana pod względem ich własności. Poniżej przedstawiona jest **teza pracy**:

„Proponowany algorytm zapewnia efektywność kompresji nie mniejszą od efektywności binarnego kodu Huffmana przy jednocześnie niskiej złożoności implementacyjnej: $E_{mix} \geq E_2$ ”.

W tezie pracy zawarto dwie główne zalety proponowanej metody kompresji: ma ona niską złożoność implementacyjną, a także charakteryzuje się wzrostem wydajności kompresji

w stosunku do binarnego kodu Huffmana. Poziom wzrostu efektywności zależy od rozkładu prawdopodobieństwa kodowanego źródła i w najgorszym przypadku mieszany kod Huffmana jest identyczny z binarnym kodem Huffmana, który jest podzbiorem kodów mieszanych dla danego źródła.

Przy omawianiu poszczególnych zagadnień autor starał się zilustrować je przykładami, co ma na celu zwiększenie przejrzystości opisów matematycznych. Większość rozważań teoretycznych jest powiązana z praktyczną realizacją poszczególnych zagadnień. Stąd w pracy znajdują się propozycje rozwiązań suboptymalnych w sytuacji, gdy czas realizacji danego zagadnienia (np. znalezienie optymalnego mieszanego kodu Huffmana) nie jest akceptowalny przez użytkownika (np. w systemach czasu rzeczywistego).

Do badań autor rozprawy korzystał z własnego oprogramowania w języku C oraz z pakietu Matlab [55, 92]. Prawie wszystkie rysunki i tabele stanowią źródło własne. W pozostałych przypadkach źródło będzie podane w miejscu cytowania.

W kolejnych dwóch punktach zostaną omówione podstawowe określenia używane w pracy oraz definicje i cechy kodów omawianych w następnych rozdziałach. Na stronie 8 znajduje się punkt zawierający wykaz ważniejszych oznaczeń i symboli używanych w pracy. Do prawidłowego zrozumienia dalszej części tej pracy niezbędna jest znajomość informacji zawartych w punktach 1.2, 1.3 oraz w rozdziale 2, w którym opisano stan aktualny najważniejszych kodów entropijnych. Podstawowym celem tej pracy jest poszukiwanie kodów efektywniejszych niż binarny kod Huffmana. Przedstawione w rozdziale 2 kody o tej własności posiadają wady, które utrudniają ich praktyczne wykorzystanie (duże wymagania pamięciowe kodera i dekodera lub długi czas wykonania zadania). W rozdziale 3 przedstawiono sposób efektywnej reprezentacji cyfr wielowartościowych, co pozwoliło na praktyczne wykorzystanie wielowartościowych kodów Huffmana. W rozdziale 4 autor proponował łączenie różnych wartościowości w jednym *mieszanym kodzie Huffmana*, przedstawiając własności takiego kodu wraz z przykładami. Kolejny rozdział przedstawia sposób konstrukcji drzewa optymalnego kodu mieszanego dla danego źródła, a także metody przyspieszania etapu konstrukcji takiego kodu. Dla zastosowań, w których podstawowym problemem jest krótki czas konstrukcji drzewa kodowego dla źródeł o dużej liczbie symboli n , autor rozprawy w rozdziale 6 przedstawił kilka propozycji tworzenia suboptymalnych kodów mieszanych. Ponadto omówione zostały propozycje kodów suboptymalnych o stałym oraz o małym opóźnieniu wprowadzanym przez koder i dekodery. Rozdział 7 zawiera analizę działania dynamicznej wersji kodów mieszanych. W kolejnym rozdziale umieszczono przykłady zastosowania mieszanych kodów Huffmana do wzrostu efektywności kodu Golomba, a także algorytmu stratnej (JPEG) i bezstratnej (z użyciem predykcji

liniowej) kompresji obrazów. Ostatni rozdział przedstawia wnioski dotyczące zaproponowanej przez autora metody poprawy efektywności kompresji danych.

1.2. Podstawowe określenia używane w pracy

Wszystkie metody opisywane w pracy dotyczą kodowania bezstratnego, czyli takiego, w którym każdy komunikat po zdekodowaniu jest identyczny jak przed etapem kodowania.

Kodowanie mające na celu zmniejszenie liczby bitów potrzebnej do przechowywania lub transmisji danych określane jest mianem kompresji danych.

Zbiór wynikowy poddany kompresji jest nazywany ciągiem danych o długości L_F bitów lub plikiem wynikowym w przypadku zbioru przechowywanego (archiwizowanego).

Dane podlegające kodowaniu określane będą jako symbole (elementy). Ciąg kodowanych elementów jest określany jako zbiór posiadający n różnych symboli źródła o alfabecie $S = \{x_1, x_2, \dots, x_n\}$ i prawdopodobieństwach odpowiednio $P = \{p_1, p_2, \dots, p_n\}$, takich że $p_n \geq 0$ oraz $p_1 \leq p_2 \leq \dots \leq p_n$. Wówczas *kodek blokowy* nazywamy odwzorowanie z S na C , czyli przypisanie słowa kodowego c_i ze zbioru $C = \{c_1, c_2, \dots, c_n\}$ każdemu symbolowi x_i źródła S [2, 21].

W przypadku, gdy mowa jest o *pliku danych* przyjmujemy, że źródło S jest zbiorem bajtów (maksymalnie 256 symboli). Natomiast *źródło binarne* składa się z dwóch symboli $S = \{x_1, x_2\}$ o prawdopodobieństwach $P = \{p_1, p_2\}$. Najczęściej do oznaczania tych symboli używa się cyfr 0 i 1. Wówczas ze źródłem binarnym $S = \{0, 1\}$ skojarzone są prawdopodobieństwa $P = \{p(0), p(1)\}$, przy czym $p(1) = 1 - p(0)$.

W tej pracy domyślnie rozpatrywane są *źródła bez pamięci* (zerowego rzędu), czyli takie, w których kolejne elementy są generowane niezależnie przy użyciu tego samego rozkładu [62].

1.3. Podstawowe cechy kodów

W tym punkcie przedstawione zostaną podstawowe definicje i twierdzenia dotyczące omawianych w pracy kodów blokowych na podstawie [2, 16, 21, 25, 47, 49, 62, 67].

Definicja 1.1 Kod nazywamy *r-narnym* (*r*-wartościowym), jeżeli słowa kodu składają się z cyfr *r*-narnych (z zakresu od zera do $r - 1$). W szczególności w kodzie binarnym ($r = 2$) słowa kodu składają się z zer i jedynek.

Definicja 1.2 Kod nazywamy *jednoznacznie dekodowalnym*, jeżeli istnieje tylko jeden sposób podziału dowolnego ciągu słów kodu $c_{i_1}c_{i_2}...c_{i_k}$ na oddzielenie słowa kodu. Czyli gdy:

$$c_{i_1}c_{i_2}...c_{i_k} = c_{j_1}c_{j_2}...c_{j_k}, \text{ to dla każdego } s, i_s = j_s, \text{ zatem } c_{i_s} = c_{j_s}.$$

Definicja 1.3 Kod nazywamy *kodem przedrostkowym* (prefiksowym), jeżeli nie możemy otrzymać żadnego słowa kodu z innego słowa kodu przez dopisanie do niego cyfr r -narnych, czyli żadne słowo nie jest przedrostkiem żadnego innego słowa kodu.

Przykładem binarnego kodu, który nie jest jednoznacznie dekodowalny może być kod $C = \{1, 01, 011\}$, gdyż np. z dwóch symboli „01” + „1” możemy utworzyć inny symbol „011”. Kod $C = \{1, 10, 100, 1000, 0000\}$ jest jednoznacznie dekodowalny, choć nie jest kodem przedrostkowym [44]. Jest to *kod z opóźnieniem*, co oznacza, że do prawidłowego zdekodowania słowa kodowego trzeba poczekać do momentu, gdy na wejściu dekodera pojawi się pierwsza cyfra następnego słowa kodowego. Natomiast kod $C = \{1, 01, 001, 000\}$ jest kodem przedrostkowym, bo dowolne słowo kodowe nie jest przedrostkiem innego słowa kodowego. Kody przedrostkowe są *kodami bez opóźnienia* [2]. W dalszej części pracy rozpatrywane będą tylko kody przedrostkowe stanowiące podzbiór kodów jednoznacznie dekodowalnych. Każdy kod przedrostkowy może być opisany drzewem kodowym, którego definicja została przedstawiona poniżej [4, 5, 71, 92]:

Definicja 1.4 Drzewo r -narne (w szczególności binarne) składa się z węzłów, z których wychodzi maksymalnie r gałęzi w kierunku węzłów potomnych. Węzeł posiada maksymalnie jednego rodzica. Węzeł nie posiadający rodzica nazywany jest *korzeniem drzewa*, a węzeł nie posiadający potomków określany jest *liściem*.

W literaturze węzeł określa się czasem mianem wierzchołka. Drzewo r -narne jest kompletne, gdy poza liśćmi każdy z węzłów posiada dokładnie r potomków. Pojedynczy węzeł nie posiadający rodzica ani potomków także jest drzewem (węzeł ten jest wówczas jednocześnie korzeniem i liściem). Przez analogię do kodów mieszanych w tej pracy niekompletne drzewo r -narne nazywane będzie niekiedy drzewem mieszanym.

Definicja 1.5 Drzewo mieszane to takie drzewo, w którym każdy i -ty węzeł nie będący liściem może mieć dowolną liczbę $r_i \geq 2$ potomków.

Poniżej zostało przedstawione twierdzenie Leona G. Krafta z 1949 roku dotyczące kodów przedrostkowych.

Twierdzenie 1.1 (*Twierdzenie Krafta*) r -narny (r wartościowy) kod przedrostkowy C złożony ze słów kodu $\{c_1, c_2, \dots, c_n\}$ o długościach $\{l_1, l_2, \dots, l_n\}$ istnieje wtedy i tylko wtedy, gdy:

$$\sum_{i=1}^n r^{-l_i} \leq 1 \quad (1.1)$$

Nierówność staje się równością Krafta dla *kompletnych kodów przedrostkowych*, czyli takich, które dla każdego węzła drzewa kodowego nie będącego liściem posiadają r potomków. Dowód twierdzenia znajduje się w pracy [62].

Jedną z podstawowych cech charakteryzujących dany kod jest entropia.

Definicja 1.6 Entropią źródła złożonego z n statystycznie niezależnych symboli o alfabecie $S = \{x_1, x_2, \dots, x_n\}$ i odpowiadającym im zestawie prawdopodobieństw wystąpienia $P = \{p_1, p_2, \dots, p_n\}$ jest funkcja:

$$H(S) = -\sum_{i=1}^n p_i \cdot \log_2 p_i \quad (1.2)$$

Entropia określa średnią ilość informacji, którą generuje źródło. W przypadku binarnego kodowania komunikatów entropia jest miarą określającą minimalną średnią liczbę bitów potrzebnych do zakodowania pojedynczego symbolu z ciągu utworzonego z kolejnych symboli wygenerowanych przez źródło. Funkcja entropii jest ograniczona w następujący sposób:

$$0 = H(1, 0, \dots, 0) \leq H(p_1, \dots, p_n) \leq H\left(\frac{1}{n}, \dots, \frac{1}{n}\right) = \log_2 n \quad (1.3)$$

Jak wykazał Shannon, dla dowolnej bezstratnej metody kompresji średnia liczba bitów potrzebnych do zakodowania symboli źródła jest nie mniejsza od wartości entropii [64]. Przy testowaniu metod bezstratnej kompresji danych często używa się bezpamięciowych źródeł o z góry zdefiniowanej entropii [69].

Dla kodów przedrostkowych można wyznaczyć *średnią długość* L_{sr} słowa kodowego (wartość oczekiwaną), gdzie l_i jest długością słowa kodu c_i kodującego symbol x_i :

$$L_{sr} = \sum_{i=1}^n p_i \cdot l_i \quad (1.4)$$

Jeżeli binarny kod przedrostkowy zostanie przedstawiony w postaci drzewa, to można wyliczyć L_{sr} także jako sumę prawdopodobieństw wystąpienia wszystkich węzłów tego drzewa (bez prawdopodobieństw liści tego drzewa) lub jako sumę prawdopodobieństw wszystkich węzłów i liści tego drzewa (oprócz prawdopodobieństwa korzenia równego 1) [45].

W przypadku kodów rozszerzonych L_{sr} oznacza średnią liczbę bitów potrzebną do zakodowania jednego symbolu pierwotnego źródła (patrz wzór (2.19)).

Definicja 1.7 Redundancja R jest różnicą między średnią długością kodu i entropią.

$$R = L_{sr} - H(S) \quad (1.5)$$

Definicja 1.8 Efektywnością kompresji $\hat{E}$ nazywamy stosunek entropii do średniej długości kodu.

$$\hat{E} = \frac{H(S)}{L_{sr}} \quad (1.6)$$

W pracy będzie często używana procentowa efektywność kompresji E :

$$E = \hat{E} \cdot 100\% = \frac{H(S)}{L_{sr}} \cdot 100\% \quad (1.7)$$

Redundancja stanowi rodzaj miary bezwzględnej efektywności kodu i wyrażana jest w bitach (dla kodów binarnych) [62]. W tej pracy będzie używana miara względna $\hat{E}$ z powodu trudności w porównaniu redundancji R dla kodów o różnych wartościowościach r , zwłaszcza w przypadku łączenia ze sobą różnych kodów r -narnych. Dokładniej sens użycia miary względnej uzasadnia pojęcie *niepewności względnej* opisane w pracy [73]. Wystarczy podać przykład dwóch źródeł S_1 i S_2 oraz odpowiadających im kodów o redundancji $R = 0.1$ bita, gdzie pierwszy kod ma entropię $H(S_1) = 8$ bitów, a drugi $H(S_2) = 0.4$ bita. Na pytanie: „czy oba kody są równie atrakcyjne pod względem efektywności?” można odpowiedzieć posługując się wzorem (1.7) na procentową efektywność kompresji, która w pierwszym przypadku wynosi aż $E(S_1) = 98.76543\%$, natomiast w drugim przypadku jedynie $E(S_2) = 80\%$. Stosunek redundancji do średniej długości słowa kodowego określamy mianem *rozwlekłości kodu* [2]. W pracy [46] przyjęto inną definicję redundancji jako zamienne określenie rozwlekłości. Powiązanie efektywności kompresji i redundancji przedstawia poniższy wzór:

$$\hat{E} = \frac{H(S)}{H(S) + R} \quad (1.8)$$

W przypadku analizy każdej modyfikacji binarnego kodu Huffmana możemy oceniać wzrost efektywności dla zmodyfikowanego kodu Huffmana. Dobrym wskaźnikiem może się okazać *waga zysku*, która jest stosunkiem zysku między średnią długością L_{mix} słowa

nowego kodu a średnią długością L_2 binarnego kodu Huffmana, do maksymalnego możliwego zysku, jaki można uzyskać (osiągając 100% efektywności kodu):

$$\hat{W} = \frac{L_2 - L_{mix}}{L_2 - H(S)} = \frac{1 - \frac{L_{mix}}{L_2}}{1 - \frac{H(S)}{L_2}} = \frac{1 - \frac{\hat{E}_2}{\hat{E}_{mix}}}{1 - \hat{E}_2} \quad (1.9)$$

Gdzie $\hat{E}_{mix}$ oznacza efektywność zmodyfikowanego kodu, natomiast $\hat{E}_2$, to efektywność binarnego kodu Huffmana. Podobnie jak w przypadku efektywności, w pracy będzie używana procentowa waga zysku:

$$W = \hat{W} \cdot 100\% \quad (1.10)$$

Definicja 1.9 *Kodem optymalnym* nazywamy kod, dla którego średnia długość słowa kodowego L_{sr} jest najmniejsza spośród wszystkich kodów dla danego rozkładu prawdopodobieństwa P . Dla takiego kodu zachodzi zależność:

$$H(p_1, \dots, p_n) \leq L_{sr} < H(p_1, \dots, p_n) + 1 \quad (1.11)$$

Zależność (1.11) dotyczy kodów blokowych, dla których każdy symbol c_i reprezentuje pojedynczy symbol źródła x_i . Jej dowód przedstawiono w pracach [21, 62].

2. Charakterystyka podstawowych kodów entropijnych – stan aktualny

Metody kodowania źródła znane były jeszcze przed pojawieniem się pierwszych komputerów. Zarówno opracowania teoretyczne, jak i praktyczne metody dotyczyły zastosowań między innymi w transmisji danych (np. kod Morse'a). Teorię zawartości informacyjnej sygnałów i związanych z nimi kodów (w tym kodów dla kanałów bezszumowych) usystematyzował Claude Shannon przedstawiając ją w 1948 roku w pracy [64]. Praca ta stała się bazą wiedzy dla podstaw teorii informacji, której szybki rozwój związany był z rozwojem telekomunikacji i maszyn cyfrowych. Shannon przedstawił wiele cech optymalnych kodów blokowych, lecz algorytm konstrukcji optymalnego kodu blokowego dla źródła o skończonej liczbie symboli opisał dopiero w roku 1952 Dawid Huffman w pracy [38]. Prosta konstrukcja drzewa kodowego i wysoka efektywność kompresji sprawiły, iż algorytm ten stał się jednym z najczęściej wykorzystywanych kodów kompresji danych różnego typu.

Analiza własności kodu Huffmana pozwoliła naukowcom na rozwój wielu metod kompresji danych także dla źródeł o nieskończonej liczbie symboli. Do takich kodów należą metody uniwersalnego kodowania źródła [63] takie, jak np. kod Golomba (patrz rozdział 8), który służy do kodowania źródeł binarnych. Golomb opisał zasadę tworzenia takiego kodu w 1966 roku w pracy [33], natomiast dokładną analizę tego kodu przeprowadzili w 1975 roku Gallager i Van Voorhis w pracy [26]. Jest to jeden z najpopularniejszych kodów dla źródeł o rozkładzie geometrycznym, często wykorzystywany do kompresji obrazu i dźwięku. Składa się z przedrostka unarnego oraz z drugiej części będącej kodem Huffmana dla m równoprawdopodobnych symboli (tzw. kod dwufazowy opisany szerzej w punkcie 2.3).

W 1973 roku Newton Faller oraz w 1978 roku Gallager przedstawili zasadę działania oraz analizę dynamicznej wersji kodu Huffmana [27]. Rozwinięcia tych propozycji powstały w latach osiemdziesiątych. Między innymi Jeffrey Vitter w pracy [86] zaprezentował zalety binarnego kodu Huffmana o minimalnej wariancji (patrz punkt 2.2). Dokładniejszy opis kodów dynamicznych znajduje się w rozdziale 7.

Badania własności binarnego kodu Huffmana są prowadzone do dziś, czego najlepszym przykładem jest praca [70]. Wcześniej analizą redundancji oraz maksymalnej długości słów kodowych zajmowali się między innymi Gallager, Capocelli i De Santis, Manstetten, Katona, Buro [10, 12, 27, 41, 48] (patrz punkt 2.1).

Znacznie mniej publikacji odnosi się do wielowartościowych kodów Huffmana, gdyż dotyczą one głównie opracowań teoretycznych (patrz punkt 2.4).

Książka Abramsona z 1963 roku (polskie wydanie z roku 1969 [2]) przedstawia, obok binarnej i wielowartościowej wersji kodu Huffmana, także jego wersję rozszerzoną (rozszerzony kod Huffmana), która polega na tworzeniu makrosymboli. Kod rozszerzony pozwala uzyskać zwiększenie efektywności względem binarnej wersji kodu Huffmana (patrz punkt 2.5). Propozycja ta ma jednak istotne wady (między innymi duże wymagania pamięciowe), które często uniemożliwiają jej praktyczne zastosowanie [62].

Nieco lepszy sposób tworzenia makrosymboli zaproponował w 1967 roku w swojej rozprawie doktorskiej Tunstall (patrz punkt 2.6) [74]. Wzrost efektywności po zastosowaniu jednocześnie kodów Tunstalla i Huffmana nie wymaga już tak dużej liczby makrosymboli, lecz i ta propozycja może być uciążliwa np. dla sprzętowej realizacji takiego algorytmu kodera.

W ostatnich dziesięciu latach coraz popularniejsza staje się arytmetyczna metoda kompresji danych (patrz punkt 2.7), po raz pierwszy opisana przez Abramsona już w roku 1963 w pracy [2], na podstawie pomysłu opracowanego przez Eliasa. Jak podaje Sayood [62], najbardziej znaną publikacją zawierającą opis praktycznej wersji kodu arytmetycznego jest praca Rissanena i Langdona z 1979 roku [59]. Podobnie jak rozszerzenia kodu Huffmana, koder arytmetyczny pozwala na uzyskanie wysokiej efektywności kompresji. Niestety odbywa się to kosztem dużej złożoności obliczeniowej kodera a zwłaszcza dekodera (w porównaniu do binarnego kodu Huffmana).

Dokładniejsze informacje dotyczące zasygnalizowanych wcześniej metod kompresji entropijnej znajdują się w kolejnych punktach tego rozdziału. Punkt 2.8 zawiera dwie metody efektywnego zapisu w nagłówku pliku wynikowego pełnej wiedzy o liczności symboli źródła. Zostały one opracowane przez autora rozprawy na podstawie własnych badań i propozycji opisanych w pracy [7].

Badania dotyczące kodów przedrostkowych trwają nieustannie od kilkudziesięciu lat i dotyczą nie tylko kodów optymalnych takich jak kod Huffmana, ale także kodów suboptymalnych, o czym świadczy choćby artykuł [61] z 2004 roku opisujący udoskonalenie kodu Fano (pod nazwą Fano⁺).

Użycie kodu Huffmana zamieniającego zestaw słów kodowych o stałej długości (np. źródło będące zbiorem bajtów) na zestaw słów kodowych o zmiennej długości bitowej pozwala nie tylko na kompresję, ale także utrudnia odczyt jawnych danych i może być używany jako jeden z etapów szyfrowania. Do innych zastosowań kodu Huffmana można

zaliczyć projektowanie języków maszynowych o zmiennej długości słowa kodowego, gdzie najczęściej używanym instrukcjom przypisuje się najkrótsze słowa kodowe [60].

Podstawy autorskiej propozycji użycia *mieszanych kodów Huffmana* do uzyskania wysokiej efektywności kompresji omówione zostaną w rozdziale 4. Zarówno koder, jak i dekodek mieszanego kodu Huffmana nie wymagają użycia dużego rozmiaru pamięci, co jest główną wadą rozszerzonych kodów Huffmana oraz kodu Tunstalla-Huffmana. Mieszane kody Huffmana wykorzystują symbole wielowartościowe, lecz ich stosowanie związane jest wyłącznie z binarnymi systemami zapisu i transmisji danych. Nie należy ich zatem kojarzyć z rozważaniami na temat optymalnych lub suboptymalnych kodów przedrostkowych, w których rozpatruje się możliwość zapisu słów kodu o poszczególnych cyfrach (sygnałach) charakteryzujących się różnymi kosztami [1, 29, 31].

2.1. Binarny kod Huffmana

W 1952 roku David Huffman przedstawił w pracy [38] algorytm i własności kodu, który jest optymalnym binarnym kodem przedrostkowym dla źródła S , tzn. posiada najkrótszą średnią długość słowa kodowego. Jest to kod optymalny (związły) wśród kodów binarnych, dla których każdy symbol c_i reprezentuje pojedynczy symbol źródła x_i . Poniżej znajdują się własności binarnego kodu Huffmana dla źródła o alfabecie $S = \{x_1, x_2, \dots, x_n\}$, prawdopodobieństwach odpowiednio $P = \{p_1, p_2, \dots, p_n\}$ oraz długościach słów kodowych $\{l_1, l_2, \dots, l_n\}$:

- kod jest optymalny i kompletny, więc spełnia nierówność Krafta (która ma postać równości) oraz zależność (1.11);
- jeżeli $p_j > p_i$, to $l_j \leq l_i$;
- dwa symbole o najmniejszych prawdopodobieństwach mają słowa kodowe o tej samej, maksymalnej długości $l_{\max} = l_1 = l_2$, różniące się tylko ostatnim bitem.

Poniżej znajduje się algorytm tworzenia binarnego kodu Huffmana opracowany na podstawie [37, 66]:

- Utwórz listę n pojedynczych drzew z każdego symbolu źródła i ustaw te drzewa według niemalejącej kolejności odpowiadających im prawdopodobieństw (lub liczności);
- Dopóki istnieją co najmniej dwa drzewa:
 - pobierz dwa drzewa t_1 i t_2 o najmniejszych prawdopodobieństwach p_1 i p_2 oraz użyj ich jako potomków dla nowego drzewa (węzła) o prawdopodobieństwie $p_1 + p_2$;
 - usuń dwa pobrane z listy drzewa i wstaw tam nowopowstałe;

- Na podstawie powstałego drzewa głównego utwórz słowa kodu C dla każdego symbolu x_i przechodząc drzewo od korzenia poprzez kolejne węzły do liścia zawierającego prawdopodobieństwo skojarzone z danym symbolem x_i łącząc kolejne napotkane zera (lewy potomek) i jedyinki (prawy potomek) w symbol kodowy c_i .

Algorytm ten nie generuje jednoznacznie drzewa kodu Huffmana, zatem mogą istnieć dla jednego źródła S różne drzewa kodu Huffmana. Każde z nich ma jednak tę samą średnią długość kodu L_{sr} . Ważne jest zatem, aby algorytmy kodera i dekodera tworzyły na podstawie zestawu prawdopodobieństw identyczne drzewo kodowe. Zarówno koder, jak i dekodery binarnego kodu Huffmana mają niewielką złożoność implementacyjną. W związku z tym w rozprawie tej pominięto opisy proponowanych udoskonaleń zmniejszania złożoności obliczeniowej tworzenia kodu Huffmana. Na uwagę zasługuje jednak propozycja tablicowego dekodera, gwarantująca szybkie działanie dekodera kodu Huffmana przedstawiona w pracy [13].

Przykład 2.1

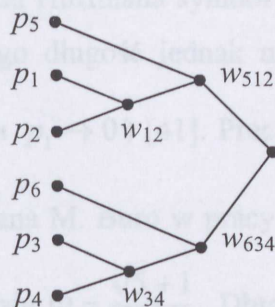
Dane jest źródło $S = \{x_1, x_2, x_3, x_4, x_5, x_6\}$, którego rozkład przedstawia tabela 2.1. W tabeli tej przedstawiono przykłady dwóch różnych kodów Huffmana dla tego źródła. Średnia długość kodu I wynosi $L_{sr} = \frac{2 \cdot 4 + 3 \cdot 2 \cdot 3 + 4 \cdot 1}{12} = 2.5$ bita na symbol. Taką samą wartość otrzymamy dla kodu II: $L_{sr} = \frac{2 \cdot 3 + 2 \cdot 2 \cdot 3 + 2 \cdot 2 + 4 \cdot 2}{12} = 2.5$.

Tabela 2.1 Przykładowa tabela kodu Huffmana

p_i	Kod I	Kod II
$p_1 = \frac{1}{12}$	0100	010
$p_2 = \frac{1}{12}$	0101	011
$p_3 = \frac{1}{6}$	011	110
$p_4 = \frac{1}{6}$	000	111
$p_5 = \frac{1}{6}$	001	00
$p_6 = \frac{1}{3}$	1	10

Tabela 2.2 Kolejne kroki budowy drzewa kodu Huffmana

S	p_i	S_1	S_2	S_3	S_4
x_6	$p_6 = \frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{2}{3}$
x_5	$p_5 = \frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$
x_4	$p_4 = \frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{3}$	
x_3	$p_3 = \frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$		
x_2	$p_2 = \frac{1}{12}$	$\frac{1}{6}$			
x_1	$p_1 = \frac{1}{12}$				



Rysunek 2.1 Drzewo kodowe z przykładu 2.1 (kod II)

Przeanalizujemy kolejne kroki tworzenia kodu II (patrz tabela 2.2). Łączymy dwa najmniej prawdopodobne symbole x_1 oraz x_2 węzłem w_{12} o prawdopodobieństwie $p_{12} = \frac{1}{6}$, a następnie usuwamy je z listy. Węzeł w_{12} umieszczamy na liście. Posiada on potomków x_1 oraz x_2 (o binarnych gałęziach, oznaczonych odpowiednio cyfrą „0” dla pierwszego z dwóch łączonych symboli oraz „1” dla drugiego symbolu). Po posortowaniu otrzymujemy zbiór $S_1 = \{x_3, x_4, x_5, w_{12}, x_6\}$ składający się z pięciu symboli. W drugim kroku łączymy symbole x_3 oraz x_4 węzłem w_{34} o prawdopodobieństwie $p_{34} = \frac{1}{3}$, a następnie usuwamy je z listy. Po posortowaniu otrzymujemy zbiór $S_2 = \{x_5, w_{12}, x_6, w_{34}\}$ składający się z czterech symboli. Następnie łączymy symbol x_5 oraz węzeł w_{12} węzłem w_{512} o prawdopodobieństwie $p_{512} = \frac{1}{3}$, a następnie usuwamy je z listy. Po posortowaniu otrzymujemy zbiór $S_3 = \{x_6, w_{34}, w_{512}\}$ składający się z trzech symboli. W kolejnym kroku łączymy symbol x_6 oraz węzeł w_{34} węzłem w_{634} o prawdopodobieństwie $p_{634} = \frac{2}{3}$, a następnie usuwamy je z listy. Po posortowaniu otrzymujemy zbiór składający się z dwóch symboli, które łączymy korzeniem drzewa przedstawionego na rysunku 2.1. Dla tak utworzo-

negu drzewa budujemy tabelę kodową, rozpoczynając zapis ścieżek od korzenia do kolejnych liści drzewa (bardziej prawdopodobny potomek oznaczany jest jako „1”, a mniej prawdopodobny jako „0”).

Dla binarnego kodu Huffmana, znając dokładną licznosc każdego z n symboli źródła, możemy wyliczyć długość L_F całego strumienia po kompresji wyrażoną w bitach:

$$L_F = \lceil K \cdot L_{sr} \rceil \quad (2.1)$$

Gdzie K jest całkowitą liczbą elementów kompresowanego strumienia danych, a L_{sr} jest średnią długością kodu Huffmana dla kodowanego źródła.

Zgodnie z własnościami kodu Huffmana symbol najmniej prawdopodobny ma najdłuższe słowo kodowe $l_{\max}$, którego długość jednak nie powinna przekroczyć wartości

$$-\frac{\log_2 p_1}{\log_2 \frac{1+\sqrt{5}}{2}} \approx -1.44042 \cdot \log_2 p_1 \quad (\text{dla } p_1 \rightarrow 0) \quad [41].$$

Precyzyjniej określił maksymalną długość słowa binarnego kodu Huffmana M. Buro w pracy [10] jako mniejszą z dwóch liczb

$$(n-1) \text{ oraz } \left\lceil \log_{\Phi} \frac{\Phi+1}{p_1 \cdot \Phi + p_2} \right\rceil, \text{ gdzie } \Phi = \frac{\sqrt{5}+1}{2}.$$

$$\text{Długość } l_{\max} \text{ jest liczbą całkowitą, która dla } 0 < p_1 \leq 0.5 \text{ spełnia zależność } \frac{1}{F(l_{\max}+3)} < p_1 \leq \frac{1}{F(l_{\max}+2)}, \text{ gdzie } F(n) \text{ jest warto-}$$

ścią ciągu Fibonacciego (opis własności ciągu znajduje się w punkcie 6.1) [3]. W 1978 roku

Gallager w pracy [27] przedstawił dokładniejsze, niż to wynikające z zależności (1.11),

ograniczenie górne na średnią długość binarnego kodu Huffmana L_{sr} jako

$$L_{sr} \leq H(S) + p_{\max} + \sigma, \text{ dla } p_{\max} < 0.5, \text{ natomiast } L_{sr} \leq H(S) + p_{\max}, \text{ dla } p_{\max} \geq 0.5. \text{ Gdzie } \sigma$$

wynosi $1 - \log_2 e + \log_2(\log_2 e) \approx 0.08607$. W latach osiemdziesiątych i dziewięćdziesiątych

bardziej szczegółowo ograniczenia górne na średnią długość kodu L_{sr} prezentowali

w swoich publikacjach między innymi Capocelli i De Santis np. w pracy [12], a także Man-

stetten w [48].

2.2. Binarny kod Huffmana o minimalnej wariancji

Propozycja wyboru odpowiedniego drzewa spośród kodów optymalnych została przedstawiona przez Vittera w pracy [86]. Algorytm tworzenia kodu Huffmana został uszczegółowiony w taki sposób, aby podczas łączenia dwóch drzew (węzłów lub symboli) o najmniejszych prawdopodobieństwach, umieścić nowopowstałe drzewo (węzeł ostatecznego drzewa) na posortowanej niemalejąco liście na końcu drzew o prawdopodobieństwie równym sumie prawdopodobieństw aktualnie łączonych drzew. Praktyczna realizacja ta-

kiego kodu polega na przypisaniu symbolom źródła numerów od 0 do $n - 1$, natomiast powstające węzły numeruje się kolejnymi liczbami od n do $2 \cdot n - 2$. Powstały w ten sposób kod ma najmniejszą łączną długość ścieżek $\sum_{i=1}^n l_i$ oraz najmniejszą wariancję długości ścieżek. Najdłuższy z elementów kodowych ma minimalną możliwą długość wśród kodów Huffmana odpowiadających danemu źródłu. W praktyce oznacza to mniejsze zapotrzebowanie np. na pasmo dla transmisji danych kodowanych w czasie rzeczywistym (w przypadku wystąpienia obok siebie wielu symboli o małym prawdopodobieństwie zmniejsza się możliwość przepełnienia bufora wyjściowego). Dla obu kodów z tabeli 2.1 możemy wyliczyć wariancję σ^2 długości symboli kodu [9]:

$$\sigma^2 = E[X^2] - L_{sr}^2 = \sum_{i=1}^n p_i \cdot l_i^2 - L_{sr}^2 \quad (2.2)$$

Dla kodu *I* z tabeli 2.1 o długościach słów kodowych $\{1, 3, 3, 3, 4, 4\}$ otrzymujemy:

$$\sigma^2 = \frac{2 \cdot 4^2 + 3 \cdot 2 \cdot 3^2 + 4 \cdot 1^2}{12} - \left(\frac{5}{2}\right)^2 = 1.25,$$

natomiast dla kodu *II* o długościach słów kodowych $\{2, 2, 3, 3, 3, 3\}$, który powstał zgodnie z zasadą Vittera, otrzymujemy jedynie:

$$\sigma^2 = \frac{2 \cdot 3^2 + 2 \cdot 2 \cdot 3^2 + 2 \cdot 2^2 + 4 \cdot 2^2}{12} - \left(\frac{5}{2}\right)^2 = 0.25.$$

2.3. Kod dwufazowy

W przypadku źródła o rozkładzie jednostajnym dane jest n równoprawdopodobnych zdarzeń elementarnych o $p_1 = p_2 = \dots = p_n = \frac{1}{n} = p$. Symbole x_i tego kodu numerujemy od 0 do $n - 1$ i w przypadku kodu o stałej długości są one zapisane jako słowa t bitowe, gdzie t jest liczbą całkowitą, taką że:

$$2^{t-1} < n \leq 2^t \quad (2.3)$$

Dla kodu o stałej długości, gdy n nie jest potęgą dwójki, kody o numerach od n do $2^t - 1$ nie są używane. Dla źródeł o rozkładzie jednostajnym można użyć binarnego kodu Huffmana, który nazywamy *kodem dwufazowym*, gdyż część symboli reprezentowana jest przez $t - 1$ bitów (naturalna reprezentacja dla $x_i < 2^t - n$), a pozostała część przez t bitów

w postaci liczby $x_i + 2^t - n$ [67]. Przykładową tabelę kodową dla $n = 7$ zawiera tabela 2.3. Warunki użycia kodu dwufazowego opartego na wielowartościowym kodzie Huffmana przedstawiono w pracy [28].

Tabela 2.3 Kod dwufazowy dla $n = 7$

Symbol	Kod Huffmana
x_0	00
x_1	010
x_2	011
x_3	100
x_4	101
x_5	110
x_6	111

Poniżej znajduje się wyprowadzenie wzoru na średnią długość słowa kodu dwufazowego: Jeżeli $n = 2^t$, to wszystkie równoprawdopodobne słowa kodu mają stałą długość t bitów. Jeśli zmniejszymy liczbę symboli źródła o 1, to aby zachować kompletność drzewa binarnego wystarczy, aby pierwsze ze słów kodowych miało długość $t - 1$ bitów. Usuwamy jeden symbol x_0 , który jest potomkiem węzła w_{01} , a następnie usuwamy węzeł w_{01} i w jego miejscu wstawiamy dotychczasowy drugi symbol x_1 potomny węzła w_{01} . Przykład dla $n = 4$ przedstawia rysunek 2.2. Kolejne zmniejszenie liczby symboli o 1, to zastąpienie następnej pary symboli jednym symbolem. Ogólnie po usunięciu X symboli z początkowych $n = 2^t$ otrzymujemy drzewo kodowe, w którym X słów kodowych ma długość $t - 1$ bitów, a pozostałych $n - X$ słów ma długość t bitów. Wówczas korzystając z nierówności Krafta (dla kodu Huffmana ma ona postać równania):

$$X \cdot 2^{-(t-1)} + (n - X) \cdot 2^{-t} = 1$$

Po uproszczeniu otrzymujemy:

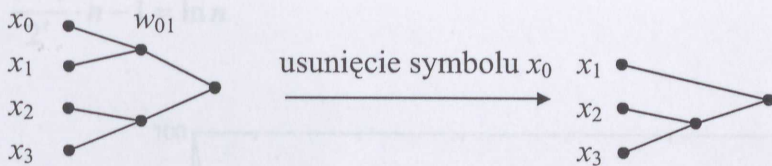
$$X = 2^t - n \tag{2.4}$$

Wówczas średnia długość kodu dwufazowego wynosi:

$$L_{sr} = \frac{X \cdot (t - 1) + (n - X) \cdot t}{n} = t + 1 - \frac{2^t}{n} \tag{2.5}$$

Natomiast procentowa efektywność takiego kodu dana jest wzorem:

$$E = \hat{E} \cdot 100\% = \frac{\log_2 n}{t + 1 - \frac{2^t}{n}} \cdot 100\% \tag{2.6}$$



Rysunek 2.2 Zmniejszenie liczby równoprawdopodobnych symboli źródła z $n = 4$ do $n = 3$

Wykorzystując ograniczenia na średnią długość słowa kodowego zaprezentowane w pracy [27], możemy oszacować dolne ograniczenie efektywności kodu dwufazowego. Ponieważ dla $n > 2$ mamy $p = \frac{1}{n} < 0.5$, to:

$$L_{sr} \leq H(S) + p + 0.08607$$

$$L_{sr} \leq \log_2 n + \frac{1}{n} + 0.08607$$

$$\frac{1}{L_{sr}} \geq \frac{1}{\log_2 n + \frac{1}{n} + 0.08607}$$

$$\frac{\log_2 n}{L_{sr}} \geq \frac{\log_2 n}{\log_2 n + \frac{1}{n} + 0.08607}$$

Wynika z tego, że procentowa efektywność kodu dwufazowego jest ograniczona z dołu przez:

$$E_{\min} \geq \frac{\log_2 n}{\log_2 n + \frac{1}{n} + 0.08607} \cdot 100\% \quad (2.7)$$

Ponieważ dla zdefiniowanej wartości t ograniczenie jest funkcją rosnącą (lokalnie), to przyjmujemy, że $n = 2^{t-1}$ i wtedy otrzymujemy $E_{\min}$ w postaci zależności (2.8):

$$E_{\min} \geq \frac{t-1}{t-0.91393+\frac{1}{2^{t-1}}} \cdot 100\% \quad (2.8)$$

Np. dla $n = \overline{129;256}$ mamy $t = 8$, wtedy $E_{\min} \geq 98.67655\%$, przy rzeczywistej wartości $E_{\min} = 98.85989\%$. Rysunek 2.3 przedstawia zależność procentowej efektywności E w funkcji n dla przedziału $n = \overline{129;256}$.

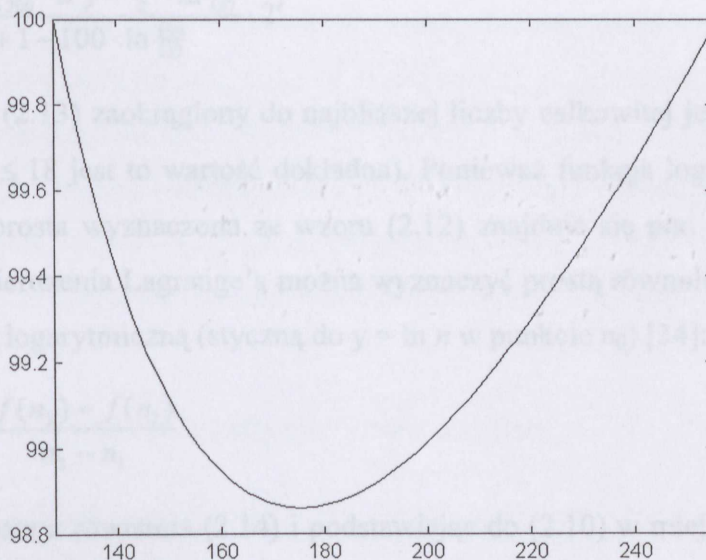
Można wyznaczyć minimum funkcji efektywności przyrównując jej pochodną do zera

$$\frac{d\hat{E}}{dn} = 0, \text{ gdzie:}$$

$$\frac{d\hat{E}}{dn} = \frac{\frac{1}{n \ln 2} \cdot (t+1 - \frac{2^t}{n}) - \log_2 n \cdot \frac{2^t}{n^2}}{(t+1 - \frac{2^t}{n})^2} \quad (2.9)$$

Po uproszczeniu otrzymujemy równanie w następującej postaci:

$$\frac{t+1}{2^t} \cdot n - 1 = \ln n \quad (2.10)$$



Rysunek 2.3 Zależność procentowej efektywności w funkcji n dla $t = 8$

Równanie (2.10) ma jedno rozwiązanie $n_{\min}$ w przedziale $n = \overline{2^{t-1} + 1; 2^t}$, przy czym minimum rozpatrujemy jedynie dla całkowitych wartości n . Oznacza to, że wynik równania należy zaokrąglić do najbliższej liczby całkowitej. Na podstawie badań można zauważyć, że wartość $n_{\min}$ dla każdej wartości t spełnia zależność $\frac{n_{\min}}{t} \approx 0.69$. Aby uprościć równanie

(2.10) można wyznaczyć przybliżenie funkcji $y = \ln n$ przy pomocy funkcji liniowej. Tworzymy zatem prostą przechodzącą przez dwa punkty krzywej logarytmicznej o argumentach $n_1 = 0.685 \cdot t$ oraz $n_2 = 0.695 \cdot t$ zgodnie ze wzorem [24]:

$$y - f(n_1) = \frac{f(n_2) - f(n_1)}{n_2 - n_1} \cdot (n - n_1) \quad (2.11)$$

Po podstawieniu argumentów n_1 i n_2 do wzoru (2.11):

$$y - \ln(0.685 \cdot 2^t) = \frac{\ln(0.695 \cdot 2^t) - \ln(0.685 \cdot 2^t)}{(0.695 - 0.685) \cdot 2^t} \cdot (n - 0.685 \cdot 2^t)$$

otrzymujemy funkcję liniową postaci:

$$y = \frac{100 \cdot \ln \frac{139}{137}}{2^t} \cdot (n - 0.685 \cdot 2^t) + \ln(0.685 \cdot 2^t) \quad (2.12)$$

Podstawiając (2.12) do (2.10) otrzymujemy równanie liniowe, które ma następujące rozwiązanie:

$$n = \frac{1 + \ln(\frac{137}{200} \cdot 2^t) - \frac{137}{2} \cdot \ln \frac{139}{137}}{t + 1 - 100 \cdot \ln \frac{139}{137}} \cdot 2^t \tag{2.13}$$

Wynik równania (2.13) zaokrąglony do najbliższej liczby całkowitej jest przybliżoną wartością $n_{\min}$ (dla $t \leq 18$ jest to wartość dokładna). Ponieważ funkcja logarytmu naturalnego jest wklęsła, to prosta wyznaczona ze wzoru (2.12) znajduje się pod wykresem $y = \ln n$. Korzystając z twierdzenia Lagrange’a można wyznaczyć prostą równoległą do (2.12) położoną nad funkcją logarytmiczną (styczną do $y = \ln n$ w punkcie n_0) [34]:

$$f'(n_0) = \frac{f(n_2) - f(n_1)}{n_2 - n_1} \tag{2.14}$$

Wyznaczając prostą z równania (2.14) i podstawiając do (2.10) w miejsce funkcji logarytmicznej otrzymujemy równanie liniowe, które ma następujące rozwiązanie:

$$n = \frac{\ln \frac{2^t}{100 \cdot \ln \frac{139}{137}}}{t + 1 - 100 \cdot \ln \frac{139}{137}} \cdot 2^t \tag{2.15}$$

Wynik równania (2.15) zaokrąglony do najbliższej liczby całkowitej jest przybliżoną wartością $n_{\min}$ (dla $t \leq 25$ jest to wartość dokładna). W tabeli 2.4 zostały przedstawione wyniki pomiarów minimalnej efektywności dla przedziałów n odpowiadających $t \leq 16$.

Tabela 2.4 Efektywność minimalna dla $t \leq 16$

t	Przedział n	Wartość n dla $E_{\min}$	Efektywność minimalna [%]
2	3÷4	3	95.09775004326937
3	5÷8	5	96.74700395364010
4	9÷16	11	97.57371232053916
5	17÷32	22	98.10749561002055
6	33÷64	44	98.44876689345948
7	65÷128	88	98.68576084029205
8	129÷256	175	98.85988965660785
9	257÷512	351	98.99332402691464
10	513÷1024	703	99.09880396464439
11	1025÷2048	1408	99.18426534914678
12	2049÷4096	2818	99.25491645399396
13	4097÷8192	5639	99.31430030261441
14	8193÷16384	11284	99.36491397368299
15	16385÷32768	22579	99.40856728412257
16	32769÷65536	45175	99.44660398986694

W przypadku bardzo dużej liczby symboli kodowych pojawia się problem związany z objętością przechowywanej w pamięci całej tabeli kodowej. Kod dwufazowy posiada proste w implementacji algorytmy kodowania i dekodowania. Nie wymagają one przechowywania w pamięci całej tabeli kodowej. Wystarczy jedynie znajomość liczby n symboli źródła, co po podstawieniu do zależności (2.3) pozwala uzyskać potrzebne parametry t oraz X . Dzięki tym własnościom kod dwufazowy ma duże zastosowanie praktyczne np. jako jeden z elementów kodów Golomba-Rice'a. W przypadku realizacji sprzętowej koder i dekodek wymagają jedynie operacji dodawania, odejmowania i przesunięcia bitowego. Poniżej znajduje się *algorytm koder* kodu dwufazowego zaprezentowany przez autora w pracy [78]:

Wejście: Liczba x_i (t bitów) określająca numer symbolu;

Jeżeli ($x_i < X$) to wyślij liczbę x_i ($t - 1$ najmłodszych bitów) na wyjście;

W przeciwnym wypadku:

{

$x_i = x_i + X$;

Wyślij liczbę x_i na wyjście (t bitów – zapis naturalny);

}

Algorytm dekodek kodu dwufazowego:

Wczytaj z wejścia $t - 1$ bitów do liczby x_i ;

Jeżeli ($x_i < X$) to:

{

dodaj z przodu 0, żeby otrzymać naturalną postać liczby t bitowej;

}

W przeciwnym wypadku:

{

wczytaj z wejścia jeden bit do liczby C ;

$x_i = 2 \cdot x_i + C - X$; (liczba t bitowa)

}

2.4. Wielowartościowy kod Huffmana

Wielowartościowym (niebinarnym) kodem Huffmana nazywamy taki kod, w którym elementy kodu zapisywane są w r -narnym alfabecie dla $r > 2$ [62]. Ma on podobne własności do binarnego kodu Huffmana, przy czym w drzewie kodowym łączy jednym węzłem r potomków. Może to jednak prowadzić do sytuacji niekompletności drzewa,

w której jeden z węzłów będzie miał mniej niż r potomków. Jest to pierwszy węzeł tworzony z y symboli o najmniejszych prawdopodobieństwach, gdzie $2 \leq y \leq r$. Tworząc r -narny kod Huffmana dla źródła składającego się z n symboli wartość y wyznaczana jest ze wzoru:

$$y = (n - 2) \bmod (r - 1) + 2 \tag{2.16}$$

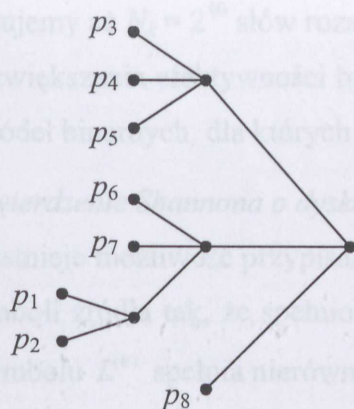
będący prostszą wersją wzoru przedstawionego w pracy [72]. Pozostałe węzły drzewa mają r potomków. Przykład 2.2 uzasadnia małą efektywność praktycznej realizacji kodu ternarnego (w ogólności r -narnego), w porównaniu do binarnego kodu Huffmana, co sprawia, że do tej pory wielowartościowe kody Huffmana nie znajdowały praktycznego zastosowania w metodach kompresji danych. Teoretyczne rozważania przedstawione zostały w wielu pracach takich jak [2, 11, 39, 43]. Praktyczne zastosowania dotyczyły jedynie kodów dla r będących potęgą dwójki [13].

Przykład 2.2

Dane jest ośmiosymbolowe źródło o rozkładzie przedstawionym w tabeli 2.5, dla którego tworzymy ternarny kod Huffmana ($r = 3$). Zgodnie ze wzorem (2.16) łączymy najpierw $y = 2$ symbole o prawdopodobieństwach p_1 i p_2 tworząc węzeł w_{12} . W kolejnym kroku łączymy symbole o prawdopodobieństwach p_3 , p_4 i p_5 tworząc węzeł w_{345} . Następnie łączymy symbole o prawdopodobieństwach p_6 i p_7 z węzłem w_{12} tworząc węzeł w_{6712} . W ostatnim kroku łączymy węzły w_{345} i w_{6712} z symbolem p_8 tworząc korzeń drzewa przedstawionego na rysunku 2.4. Na podstawie utworzonego drzewa budujemy słowa kodu ternarnego, które zostały przedstawione w tabeli 2.5. Jak widać kod „122” nie jest używany. Nie jest to zatem kod kompletny. Taka cecha zmniejszająca efektywność kompresji jest wadą kodu r -narnego, która nie występowała w przypadku binarnego kodu Huffmana.

Tabela 2.5 Kod Huffmana z przykładu 2.2

p_i	Kod ternarny	Kod binarny
$p_1 = 0.05$	120	1010
$p_2 = 0.05$	121	1011
$p_3 = 0.10$	00	000
$p_4 = 0.10$	01	001
$p_5 = 0.10$	02	010
$p_6 = 0.10$	10	011
$p_7 = 0.10$	11	100
$p_8 = 0.40$	2	11



Rysunek 2.4 Drzewo kodu ternarnego z przykładu 2.2

Mimo to teoretyczna średnia długość kodu ternarnego wynosi $L_3 = 1.7 \cdot \log_2 3 = 2.69444$ (daje to teoretyczną efektywność $E_3 = 97.30897\%$) i jest mniejsza niż dla kodu binarnego, dla którego $L_2 = 2.7$. Wynika to z faktu, iż teoretycznie do zapisu jednej cyfry ternarnej wystarczy $\log_2 3$ bitów, jednak przy kodowaniu bezpośrednim potrzeba aż 2 bitów. Zatem faktyczna średnia długość kodu ternarnego wynosi $L_3 = 3.4$ bita na symbol, co daje efektywność na poziomie zaledwie $E_3 = 77.11553\%$ (przy efektywności kodu binarnego $E_2 = 97.10845\%$).

2.5. Rozszerzony kod Huffmana

Jedną z podstawowych metod zwiększania efektywności kodu Huffmana jest tworzenie kodu rozszerzonego. Polega ona na łączeniu ze sobą k symboli źródła S w jeden makrosymbol (blok symboli). W rzeczywistości mamy do czynienia z k -krotnym rozszerzeniem źródła, a rozszerzenie kodu Huffmana jest tylko konsekwencją rozszerzenia źródła. Jeżeli źródło S składa się z n symboli, to rozszerzone źródło $S^{(k)}$ będzie zawierać $N_k = n^k$ słów kodowych rozszerzonego alfabetu (makrosymboli), będących zestawem wszystkich możliwych ciągów k -elementowych utworzonych z symboli pierwotnego źródła S . Dla takiego źródła $S^{(k)}$ tworzymy binarny kod Huffmana. Dla źródła bez pamięci kolejne symbole pierwotnego źródła są od siebie niezależne. Zatem prawdopodobieństwo wystąpienia makrosymbolu jest równe iloczynowi prawdopodobieństw poszczególnych symboli tworzących dany element rozszerzonego alfabetu. Oznacza to, że koder i dekodek do utworzenia drzewa kodowego potrzebują jedynie zestawu n prawdopodobieństw skojarzonych z pierwotnym źródłem S . Jednak dla większych wartości k metoda staje się niepraktyczna ze względu na wykładniczy wzrost rozmiaru alfabetu, dla którego należy stworzyć i przechowywać w pamięci n^k słów rozszerzonego kodu Huffmana [62]. Np. dla źródła o $n = 256$

symbolach przy $k = 5$ potrzebujemy aż $N_k = 2^{40}$ słów rozszerzonego kodu Huffmana. Jest to główna wada tej propozycji zwiększania efektywności binarnego kodu Huffmana. Kod ten stosowany jest głównie dla źródeł binarnych, dla których $n = 2$ [22].

Twierdzenie 2.1 (*Pierwsze twierdzenie Shannona o dyskretnym kodowaniu bezszumowym*). Dla źródła S o entropii $H(S)$ istnieje możliwość przypisania słów kodowych (makrosymboli) blokom złożonym z k symboli źródła tak, że spełniona jest właściwość przedrostkowa oraz średnia długość makrosymbolu $L^{(k)}$ spełnia nierówność:

$$H(S) \leq L_{sr}^{(k)} < H(S) + \frac{1}{k} \tag{2.17}$$

Gdzie $L_{sr}^{(k)} = \frac{L^{(k)}}{k}$ oznacza średnią liczbę bitów potrzebną do zakodowania jednego symbolu pierwotnego źródła S [2, 21]. Teoretycznie ze wzrostem liczby k symboli w bloku powinna rosnać efektywność. W praktyce nie jest to takie oczywiste, gdyż istnieją przypadki, w których zastosowanie większej wartości k może prowadzić do zmniejszenia się procentowej efektywności E [80]. Przy szacowaniu średniej długości słowa kodowego oprócz zależności (2.17) możemy wykorzystać ograniczenie górne Gallagera opisane w punkcie 2.1, które może być przydatne zwłaszcza dla małych wartości $p_{\max}$. Przy czym wartość prawdopodobieństwa najczęściej występującego symbolu rozszerzonego źródła jest równa $p_{\max}^k$. Poniższy przykład pokazuje stopień zwiększania liczby symboli rozszerzonego alfabetu wraz ze wzrostem k , pomimo małej wartości $n = 3$.

Przykład 2.3

Dane jest trójsymbolowe źródło o rozkładzie przedstawionym w tabeli 2.6, dla którego tworzymy binarny kod Huffmana. Średnia długość słowa kodu wynosi $L_{sr} = 0.8 + 2 \cdot (0.16 + 0.04) = 1.2$. Daje to efektywność zaledwie $E_2 = 72.19281\%$.

Tabela 2.6 Kod Huffmana z przykładu 2.3

Symbole	p_i	Kod Huffmana
c	0.80	1
b	0.16	01
a	0.04	00

Tabela 2.7 Kod Huffmana dla $k = 2$

Kod blokowy	p_i	Kod binarny
cc	0.64	0
cb	0.128	10
bc	0.128	110
ca	0.032	11100
ac	0.032	11101
bb	0.0256	11110
ba	0.0064	111110
ab	0.0064	1111110
aa	0.0016	1111111

Tabela 2.8 Kod Huffmana dla $k = 3$

Kod blokowy	p_i	Kod binarny
ccc	0.512	0
bcc	0.1024	100
cbc	0.1024	101
ccb	0.1024	110
cca	0.0256	111100
cac	0.0256	111101
acc	0.0256	111110
cbb	0.02048	111000
bcb	0.02048	111001
bbc	0.02048	111010
bca	0.00512	11101100
bac	0.00512	11101101
abc	0.00512	11101110
cba	0.00512	11111100
cab	0.00512	11111101
acb	0.00512	11111110
bbb	0.004096	111111110
aac	0.00128	1111111110
caa	0.00128	1110111100
aca	0.00128	1110111101
bba	0.001024	1110111110
bab	0.001024	1110111111
abb	0.001024	11111111110
baa	0.000256	1111111111100
aba	0.000256	1111111111101
aab	0.000256	1111111111110
aaa	0.000064	1111111111111

Możemy utworzyć np. źródło $S^{(2)}$ składające się ze wszystkich możliwych par symboli (bloków symboli), dla którego budujemy kod Huffmana (rozszerzony, składający się z 9 symboli). Został on przedstawiony w tabeli 2.7. Dzięki temu uzyskujemy średnią

$L_{sr}^{(2)} = 0.9112$ bita na symbol, co daje wzrost efektywności do $E^{(2)} = 95.07394\%$. Zwiększenie długości bloku do trzech symboli poprzez utworzenie źródła $S^{(3)}$ zawierającego wszystkie trójki symboli daje średnią długość słowa kodu Huffmana $L_{sr}^{(3)} = 0.87554$. Dla tego kodu (przedstawionego w tabeli 2.8), składającego się z 27 makrosymboli efektywność wzrasta aż do $E^{(3)} = 98.94607\%$. Daje to procentową wagę zysku równą $W = 97.23464\%$.

2.6. Kod Tunstalla i jego modyfikacje

Kod Tunstalla jest kodem przedrostkowym o stałej długości m bitowych słów kodowych. Jego budowa polega na łączeniu wielu symboli pierwotnego źródła w bloki (makrosymbole). W odróżnieniu od rozszerzonego kodu Huffmana, poszczególne makrosymbole mogą się składać z różnej liczby k_i symboli pierwotnego źródła [74]. Tworzenie kodu Tunstalla rozpoczyna się od usunięcia z listy symbolu x_i o największym prawdopodobieństwie i dodania do listy n symboli będących połączeniem usuwanego symbolu x_i z każdym symbolem z pierwotnego źródła. Krok ten można powtarzać j razy, przy czym za każdym razem usuwany element łączony jest tylko z listą n pierwotnych symboli źródła. Oznacza to, że po każdym etapie na liście przybywa $n - 1$ nowych makrosymboli. Tworzenie takiego kodu wiąże się z dużym wzrostem liczby makrosymboli, czyli ze wzrostem zapotrzebowania na pamięć niezbędną do przechowywania słów kodowych i odpowiadających im ciągów symboli pierwotnego źródła. Jest to duża wada tego rozwiązania. Aby uzyskać wysoką efektywność kompresji, należy dążyć do tego, aby końcowa liczba N_j symboli po j etapach rozszerzania źródła była bliska m -tej potęgze dwójki, ale nie większa od niej [62]. Wynosi ona:

$$N_j = n + j \cdot (n - 1) \leq 2^m \quad (2.18)$$

Stosowanie słów o stałej m bitowej długości jest główną zaletą tego kodu, ponieważ zapobiega to propagacji błędów np. podczas transmisji danych. Dla makrosymboli o zróżnicowanych prawdopodobieństwach nie jest to jednak dobrym rozwiązaniem pod względem efektywności. Rezygnacja z tej zalety dzięki dodatkowemu zastosowaniu kodu Huffmana dla makrosymboli źródła Tunstalla prowadzi do zwiększenia efektywności takiego kodu, który możemy określać mianem kodu Tunstalla-Huffmana [23]. Dla kodów, w których poszczególne symbole rozszerzonego źródła mogą składać się z różnej liczby k_i symboli pierwotnego źródła, wzór na średnią długość słowa kodowego L_{sr} (a raczej średnią liczbę

bitów potrzebną do zakodowania jednego symbolu pierwotnego źródła) rozszerza się do postaci [80]:

$$L_{sr} = \frac{\sum_{i=1}^n p_i \cdot l_i}{\sum_{i=1}^n p_i \cdot k_i} \tag{2.19}$$

Odwołując się do danych z przykładu 2.3 możemy utworzyć dla $j = 2$ siedmioelementowe kody Tunstalla oraz Tunstalla-Huffmana, które przedstawia tabela 2.9. Pierwszy krok rozszerzania źródła ($j = 1$) polega na utworzeniu makrosymboli $\{aa, ab, ac\}$ o prawdopodobieństwach odpowiednio $\{0.64, 0.128, 0.032\}$, a następnie usunięciu z listy symbolu a . Po posortowaniu najbardziej prawdopodobny jest makrosymbol aa , który w drugim kroku ($j = 2$) łączymy z symbolami pierwotnego źródła tworząc nowe makrosymbole $\{aaa, aab, aac\}$ o prawdopodobieństwach odpowiednio $\{0.512, 0.1024, 0.0256\}$. Na końcu usuwamy z listy makrosymbol aa , oraz po posortowaniu listy złożonej z siedmiu symboli rozszerzonego źródła tworzymy kod Tunstalla oraz kod Tunstalla-Huffmana (patrz tabela 2.9). Dla kodu Tunstalla otrzymujemy średnią długość kodu równą $L_{sr} = 1.22951$ bita na symbol pierwotnego źródła, co daje efektywność na poziomie jedynie $E = 70.46018\%$. Tak niska efektywność wynika z faktu zastosowania słów kodowych o tej samej długości trzech bitów niezależnie od rozkładu prawdopodobieństwa, a także ze względu na niewykorzystanie w tej sytuacji kodu „111”. Natomiast w przypadku kodu Tunstalla-Huffmana średnia długość słowa kodowego wynosi zaledwie $L_{sr} = 0.87344$, co odpowiada efektywności $E = 99.18382\%$ (waga zysku $W = 97.86359\%$). Jak widać przy użyciu jedynie 7 makrosymboli uzyskaliśmy efektywność wyższą niż przy użyciu 8 (a nawet 27) symboli rozszerzonego kodu Huffmana.

Tabela 2.9 Kody Tunstalla i Tunstalla-Huffmana z przykładu 2.3

Makrosymbol	p_i	Kod Tunstalla	Kod Tunstalla-Huffmana
aaa	0.5120	000	1
b	0.1600	001	011
ab	0.1280	010	010
aab	0.1024	011	000
c	0.0400	100	0010
ac	0.0320	101	00111
aac	0.0256	110	00110

Jest to przykład elastyczności kodu Tunstalla-Huffmana, który dobrze dostosowuje się do rozkładu prawdopodobieństwa występowania symboli źródła S . W przypadku dużych war-

tości n nadal uzyskanie wysokiej efektywności związane jest z dużym zapotrzebowaniem na pamięć niezbędną do przechowywania symboli kodowych rozszerzonego źródła.

2.7. Kod arytmetyczny i inne kody entropijne

Oprócz opisanych wcześniej podstawowych kodów przedrostkowych istnieje jeszcze wiele innych, opartych w dużym stopniu na własnościach kodu Huffmana. Należą do nich kody Golomba i Rice'a, których odmiany dostosowują się do odpowiednich rozkładów. Ze względu więc na ich specyfikę nie są w tej pracy omawiane (za wyjątkiem autorskiej modyfikacji podstawowej wersji kodu Golomba).

Osobnym typem bezpamięciowych kodów entropijnych jest kod arytmetyczny, który nie należy do klasy kodów posiadających jednoznacznie dekodowalną książkę kodów [67]. Jest zatem określany mianem kodu nieblokowego, który jest jednak kodem jednoznacznie dekodowalnym w sensie potocznym (choć nie w sensie definicji 1.2) [2]. W praktyce stosuje się całkowitoliczbową wersję kodu arytmetycznego, który koduje cały kompresowany ciąg danych do jednej liczby określonej mianem *znacznika* stanowiącego wynikowy ciąg bitów. Szczegółowe opisy algorytmu kodera i dekodera znajdują się w pracach [62, 91]. Przy założeniu precyzyjnego opisu licznosci symboli w kodowanym ciągu uzyskujemy bardzo wysoką efektywność kompresji, przewyższającą efektywność kodu Huffmana. Algorytm kodera arytmetycznego jest bardziej skomplikowany od kodera binarnego kodu Huffmana, choć wymagania pamięciowe mają zbliżone. Ponadto zachowanie wysokiej efektywności wymaga użycia liczb o wysokiej precyzji (np. rejestrów źródłowych 32 bitowych), na których dokonuje się operacji dzielenia oraz mnożenia (rejestr wynikowy powinien więc być odpowiednio 64 bitowy). Główną wadą w porównaniu do metody Huffmana jest złożoność dekodera kodu arytmetycznego, który dla każdego dekodowanego elementu wymaga przeszukiwania tabeli skumulowanych licznosci oraz większej liczby operacji arytmetycznych niż koder. Stanowi to dużą wadę, ponieważ w przypadku archiwizacji najczęściej operacja kodowania wykonywana jest raz, a dekodowanie odbywa się wielokrotnie. W pracy [7] porównano czasy wykonania dla różnych algorytmów. Czas kompresji przy użyciu kodu arytmetycznego był bardzo zbliżony do dynamicznej metody Huffmana, a ponad dwukrotnie większy niż w przypadku statycznej metody Huffmana. W przypadku dekompresji czas dynamicznej metody Huffmana był prawie dwa razy większy od swojego statycznego odpowiednika. Natomiast różnica czasów wykonania między dekodorem statycznego kodu Huffmana a dekodorem arytmetycznym była zależna od charakteru źródła. Czas dekompresji arytmetycznej był od 5 do 10 razy dłuższy.

Istnieją tzw. „miękkodeczyjne” dekodery kodów Huffmana i arytmetycznego. W ich przypadku zakłada się, że bity kodu są odbierane jako liczby rzeczywiste, bo nałożony jest szum. W ich przypadku koszt dekodera arytmetycznego (optymalnego) rośnie wykładniczo, natomiast dekodera kodu Huffmana jest o wiele rozsądniejszy [35].

Do tej pory wszystkie cechy kodów były omawiane w pracy dla przypadku pełnej znajomości rozkładu prawdopodobieństwa występowania symboli. Jednak w wielu praktycznych zastosowaniach używa się z góry zdefiniowanych przybliżonych rozkładów i dopasowanych do nich kodów (np. dla obrazów czy tekstów w danym języku). Jeśli jednak taki zdefiniowany rozkład zostanie źle dobrany do właściwości źródła, to może się okazać, że koder arytmetyczny jest bardziej podatny na negatywne skutki takiej sytuacji i uzyska mniejszą efektywność od kodu Huffmana. Przykłady takich sytuacji dla nieodpowiednio dobranych rozkładów (dla tekstu) zostały przedstawione w pracy [7].

Można uznać, że np. w przypadku realizacji sprzętowych, gdzie powinniśmy dążyć do malej złożoności obliczeniowej, jeśli zmniejszymy precyzję zapisu skumulowanego rozkładu, to możemy uzyskać efektywność jedynie na poziomie kodu Huffmana. Jednak nadal dekodery arytmetyczne będą wymagały większej mocy obliczeniowej, niż w przypadku użycia dekodera kodu Huffmana, aby uzyskać porównywalnie krótki czas pracy. Jest to istotne zwłaszcza w systemach czasu rzeczywistego takich jak transmisje multimedialne. Jeśli interesuje nas zwiększenie efektywności kompresji, musimy się więc liczyć z dużą złożonością obliczeniową urządzeń kodujących i dekodujących dla kodu arytmetycznego (w porównaniu z koderem i dekodorem kodu Huffmana [13, 36]). Drugą możliwością jest wykorzystanie rozszerzonego kodu Huffmana lub Tunstalla-Huffmana, co wiąże się ze znacznym wzrostem kosztu wykorzystania pamięci. Choć trwają prace nad zmniejszeniem zapotrzebowania zasobów pamięciowych dla takich kodów, to ich wyniki nadal nie są zadowalające [54]. Przedstawiona w kolejnych rozdziałach tej pracy autorska propozycja użycia mieszanych kodów Huffmana stanowi rozwiązanie dotychczasowego problemu, nie wymagające dużych rozmiarów pamięci, ani znacznego wzrostu mocy obliczeniowej w stosunku do binarnego kodu Huffmana.

Biorąc pod uwagę często występujące sytuacje źródeł, w których dane stanowią zbiór elementów zależnych (kolejne elementy ciągu są skorelowane), możemy zwiększyć stopień kompresji przy użyciu algorytmów słownikowych [62]. Dwie podstawowe metody kompresji LZ77 i LZ78 zostały opisane przez Jacoba Ziva i Abrahama Lempela w pracach [93, 94]. Jednak dopiero kolejne modyfikacje tych algorytmów różniące się między innymi strukturą danych implementującą słownik pozwoliły uzyskiwać wysoki stopień kompresji danych (np. algorytm LZW używany przy kompresji obrazów typu *.gif*) [21, 89]. Po-

wszechne metody kompresji takie jak ARJ, Lharc, PKZip, RAR itp. bazują na algorytmach słownikowych lecz w połączeniu z innymi jednocześnie stosowanymi algorytmami modelowania danych oraz z kodem Huffmana lub kodem arytmetycznym.

W przypadku kodów słownikowych najczęściej mamy do czynienia z dynamiczną wersją słownika, który jest aktualizowany w trakcie kodowania. Istnieją także dynamiczne kody Huffmana, które uzyskują wysoką efektywność kompresji głównie dla źródeł o lokalnie zmiennych rozkładach prawdopodobieństwa występowania poszczególnych symboli. Dokładniejsze informacje dotyczące dynamicznej wersji kodu Huffmana omówione zostaną w rozdziale 7.

2.8. Metody efektywnego zapisu parametrów kompresji

Ze względu na sposób realizacji kompresji bezpamięciowe źródła danych można podzielić na trzy typy o następujących cechach:

- brak wiedzy o długości pliku i liczności lub rozkładzie prawdopodobieństwa występowania symboli (stosujemy wówczas kody dynamiczne);
- brak wiedzy o długości pliku i przybliżona wiedza o rozkładzie prawdopodobieństwa (zastosowanie np. predefiniowanych tablic kodowych dopasowanych do rozkładu);
- pełna wiedza o liczności symboli, których suma jest automatycznie długością pliku.

W dwóch pierwszych przypadkach do oznaczenia końca pliku należy stosować dodatkowy symbol kodowy x_0 (czasem określany jako *END* lub *EOF*) o liczności 1 (czasem nadaje się mu wagę 0, jako symbolowi o najmniejszym prawdopodobieństwie). W dalszej części tego punktu rozważany będzie jedynie trzeci przypadek, gdyż wszystkie omówione wcześniej metody bezpamięciowej, statycznej kompresji danych pozwalające uzyskać wyższą efektywność niż w przypadku zastosowania binarnego kodu Huffmana, potrzebowały *pełnej wiedzy o liczności* każdego z symboli źródła. W praktycznych zastosowaniach, oprócz efektywności samego kodu, należy także brać pod uwagę długość nagłówka pliku L_H (wyrażoną w bitach), który powinien być dołączany do kodu wynikowego w możliwie jak najkrótszej formie. Przy pełnej wiedzy o liczności możemy posługując się wzorem (2.1), wyznaczyć wyrażoną w bajtach wynikową długość L_{calc} kompresowanego pliku dla binarnego kodu Huffmana:

$$L_{calc} = \left\lceil \frac{L_F + L_H}{8} \right\rceil \quad (2.20)$$

Gdzie L_F jest wyrażoną w bitach długością całego strumienia danych po kompresji (patrz wzór (2.1)). Dla rozszerzonego kodu Huffmana oraz wielu innych modyfikacji kodów Huffmana L_{calc} jest jedynie wartością przybliżoną, gdyż nie możemy zagwarantować idealnej niezależności między sąsiednimi symbolami.

Poniżej zostaną przedstawione dwie metody efektywnego zapisu liczności symboli źródła. Zostały one opracowane przez autora rozprawy na podstawie własnych badań i propozycji opisanych w [7]. Można je stosować zamiennie, przy czym wystarczy na początku pliku zapisać jeden bit określający numer używanej dalej metody. Dla *metody 1* zapisujemy bit „0”, natomiast dla *metody 2* bit „1”. Przyjmujemy, że długość kodowanego strumienia danych nie jest bezpośrednio zapisywana do pliku wynikowego, bo można ją wyliczyć jako sumę poszczególnych licznosci. Liczbę n symboli źródła ograniczamy maksymalnie do 256 (traktując elementy wejściowe jako bajty), a długość K pliku źródłowego nie powinna przekraczać 1 GB. Ograniczenie wynika z możliwości zagwarantowania prawidłowej reprezentacji licznosci skumulowanej dla potrzeb kodu arytmetycznego (dla 64 bitowych wynikowych rejestrów mnożenia w algorytmie całkowitoliczbowego kodera arytmetycznego). Po wybraniu bardziej opłacalnej metody i wysłaniu na wyjście bitu z numerem metody, wysyłamy pole złożone z 256 bitów, z których każdy jest znacznikiem występowania poszczególnych bajtów (symboli źródła) w pliku źródłowym. Jeśli bajt o wartości j wystąpił w pliku źródłowym, to na j -tej pozycji tego pola wpisujemy „1”, w przeciwnym przypadku „0” [7]. Następnie wysyłamy na wyjście pięciobitową liczbę mm , której wartość zależy od wybranej metody. Poczym zapisujemy kolejno licznosci tylko dla tych n symboli (bajtów), które wystąpiły w danym pliku. Sposób zapisu tych licznosci zależy od wybranej metody:

Metoda 1

Dla $K = n$ mamy $mm = 0$ i nie trzeba zapisywać licznosci, bo wszystkie licznosci występujących bajtów mają wartość 1. Gdzie K oznacza długość kompresowanego pliku, natomiast n jest liczbą symboli źródła (liczbą występujących bajtów). W pozostałych przypadkach gdy $K - n > 0$, to wartość mm wyliczana jest ze wzoru:

$$mm = \lfloor \log_2(K - n) \rfloor + 1 \quad (2.21)$$

Metoda może być skuteczna zwłaszcza przy dużych różnicach między minimalną a maksymalną licznoscią symboli. Poniżej znajduje się algorytm *metody 1*:

$$\text{delta} = 2^{mm};$$

Dla każdego i od 0 do 255, jeśli $\text{liczność}[i] > 0$ wykonaj:

```
{
    temp = liczność[i] - 1;
    zapisz do pliku mm bitów liczby temp;
    delta = delta - temp;
    mm = ⌈log2(delta)⌉;
    jeżeli ((mm = 0) oraz (delta > 0)) to mm = 1;
}
```

Przykład 2.4

Dane jest źródło składające się z $n = 4$ symboli $\{a, b, c, d\}$ o licznosciach odpowiednio $\{200, 7500, 100, 395\}$. Długość pliku wynosi zatem $K = 8195$. Wówczas $mm = \lfloor \log_2 8191 \rfloor + 1 = 13$. Poniżej znajdują się cztery kolejne kroki zapisu licznosci:

$$\text{delta} = 2^{13} = 8192;$$

$$\text{temp} = 199;$$

zapisz 13 bitów liczby 199 = „0000011000111”;

$$\text{delta} = 7993; mm = \lceil \log_2 7993 \rceil = 13;$$

$$\text{temp} = 7499;$$

zapisz 13 bitów liczby 7499 = „1110101001011”;

$$\text{delta} = 494; mm = \lceil \log_2 494 \rceil = 9;$$

$$\text{temp} = 99;$$

zapisz 9 bitów liczby 99 = „001100011”;

$$\text{delta} = 395; mm = \lceil \log_2 395 \rceil = 9;$$

$$\text{temp} = 394;$$

zapisz 9 bitów liczby 394 = „110001010”;

Łącznie zapis licznosci wymaga użycia $13 + 13 + 9 + 9 = 44$ bitów (dla metody 2 byłyby to aż 52 bity). A zatem cały nagłówek pliku składa się z $1 + 256 + 5 + 44 = 306$ bitów.

Metoda 2

Użycie tej metody jest opłacalne dla zbliżonych licznosci poszczególnych symboli źródła.

Dla metody 2 wartość mm dla $n_{\max} > 1$ wyrażona jest wzorem:

$$mm = \lceil \log_2 n_{\max} \rceil \quad (2.22)$$

Gdzie $n_{\max}$ jest licznością najczęściej występującego symbolu. Dla $n_{\max} = 1$, to $mm = 0$, więc nie trzeba zapisywać liczności, bo wszystkie liczności występujących bajtów mają wartość 1. Poniżej znajduje się algorytm *metody 2*:

Dla każdego i od 0 do 255, jeśli $liczność[i] > 0$ wykonaj:

```
{
    temp = liczność[i] - 1;
    zapisz do pliku mm bitów liczby temp;
}
```

Na podstawie algorytmu *metody 2* można wywnioskować, że do zapisu zestawu wszystkich występujących liczności wystarczy $n \cdot mm$ bitów. Zatem cały nagłówek składa się z $(1 + 256 + 5 + n \cdot mm)$ bitów.

Istnieje także wiele innych propozycji opisu drzew kodowych [7, 14], ale np. część z nich wymusza stosowanie suboptymalnych kodów Huffmana (gdzie liście drzewa stanowią np. zbiór złożony z 2^m elementów o zbliżonych prawdopodobieństwach). Takie kody określane są czasem jako *prefiksowe kody Huffmana* [63], gdyż jedynie przedrostek kodu jest tworzony zgodnie z zasadą konstrukcji kodu Huffmana. Przykład takiego kodu zostanie omówiony w punkcie 8.1. Uzasadnia on jednocześnie przydatność użycia kodów mieszanych, jako prefiksowych kodów Huffmana.

Procentowa efektywność wynosi wówczas:

$$E = \frac{\log_2 r}{h(r)} \cdot 100\% \quad (2.3)$$

Rozważmy najprostszy przykład dla $r = 3$. Do zakodowania jednej cyfry ternarnej potrzeba aż $h(3) = 2$ bitów, a więc procentowa efektywność wynosi zaledwie $E = 79.248139\%$. Posługując się kodem dwufazowym (patrz punkt 2.3), czyli kodem Huffmana dla symboli równoprawdopodobnych, uzyskamy średnią długość jednej cyfry r -narnej

$$h(r) = h(r) + 1 - \frac{2^{h(r)}}{r} \quad (2.4)$$

Wówczas procentowa efektywność wynosi:

$$E = \frac{\log_2 r}{h(r)} \cdot 100\% \quad (2.5)$$

3. Efektywna reprezentacja symboli wielowartościowych

W tym rozdziale zostanie omówiony sposób efektywnej reprezentacji symboli wielowartościowych w postaci liczb binarnych, a także analiza ich efektywności kompresji i sprawności konwersji. Następnie zostanie przedstawiony algorytm wyznaczania wzorcowych par parametrów (r_i, q_i) , z których tworzy się q cyfrowe r -narne liczby B , a także propozycja użycia kodu dwufazowego do kompresji liczb B . Zastosowanie obu propozycji pozwala na uzyskanie niskim kosztem efektywności konwersji przekraczającej 99.7%.

3.1. Analiza reprezentacji symboli wielowartościowych

W punkcie 2.4 wspomniano o problemie małej efektywności zapisu liczb r -narnych (wielowartościowych) np. w pamięci komputera, czy na nośnikach danych, gdyż obecnie używana technika cyfrowa oparta jest o binarną reprezentację liczb. Ponieważ zakładamy, że wszystkie symbole źródła o numerach od 0 do $r - 1$ mają równe prawdopodobieństwa występowania, to problem małej efektywności nie pojawia się jedynie dla r będących potęgą dwójki. Przypadki te więc nie będą dalej omawiane.

W ogólności do zapisu jednej liczby r -narnej potrzeba $b(r)$ bitów na jej naturalny binarny zapis, gdzie $b(r)$ jest liczbą całkowitą taką, że:

$$\log_2 r \leq b(r) < \log_2 r + 1 \quad (3.1)$$

Procentowa efektywność wynosi wówczas:

$$E = \frac{\log_2 r}{b(r)} \cdot 100\% \quad (3.2)$$

Rozważmy najprostszy przykład dla $r = 3$. Do zakodowania jednej cyfry ternarnej potrzeba aż $b(3) = 2$ bitów, a więc procentowa efektywność wynosi zaledwie $E = 79.24813\%$. Posługując się kodem dwufazowym (patrz punkt 2.3), czyli kodem Huffmana dla symboli równoprawdopodobnych, uzyskamy średnią długość jednej cyfry r -narnej:

$$\underline{b}(r) = b(r) + 1 - \frac{2^{b(r)}}{r} \quad (3.3)$$

Wówczas procentowa efektywność wynosi:

$$\underline{E} = \frac{\log_2 r}{\underline{b}(r)} \cdot 100\% \quad (3.4)$$

Co przy zakodowaniu pojedynczej cyfry ternarnej daje $\underline{E} = 95.09775\%$. Uzyskanie jeszcze większej efektywności jest możliwe dzięki użyciu *zamiany bloków cyfr r -narnych na postać binarną*. Możemy to wykonać tworząc liczbę B złożoną z q cyfr r -narnych:

$$B = \sum_{i=0}^{q-1} a_i \cdot r^i \quad (3.5)$$

Liczba B przyjmuje wartości od 0 do $r^q - 1$ i możemy przyjąć, że każda z tych wartości jest równoprawdopodobna, przy założeniu, że kolejne cyfry r -narne są także równoprawdopodobne i niezależne. Do reprezentacji liczby B wykorzystujemy kod o stałej długości. Do jej zapisu zgodnie z zależnością (3.1) wystarczy $b(r^q)$ bitów, przy czym:

$$b(r^q) \leq q \cdot b(r) \quad (3.6)$$

Np. $b(5) = 3$, natomiast $b(5^3) = 7 < 3 \cdot b(5) = 9$. Używając powyższej techniki grupowania q cyfr r -narnych możemy wyznaczyć średnią liczbę bitów $\bar{b}_q(r)$ potrzebną do reprezentacji pojedynczej cyfry r -narnej jako:

$$\bar{b}_q(r) = \frac{b(r^q)}{q} \quad (3.7)$$

Używając blokowej zamiany mamy do czynienia z kompresją, gdyż na podstawie nierówności (3.6) można zauważyć, że zmniejsza się średnia liczba bitów potrzebna do reprezentacji pojedynczej cyfry r -narnej. Wówczas wzór na procentową efektywność kompresji przyjmuje następującą postać:

$$E_{r(q)} = \hat{E}_{r(q)} \cdot 100\% = \frac{\log_2 r}{\bar{b}_q(r)} \cdot 100\% \quad (3.8)$$

Rozpatrzmy zatem liczbę B złożoną z $q = 5$ cyfr ternarnych $\{a_0, a_1, a_2, a_3, a_4\}$:

$$B = a_0 \cdot 3^0 + a_1 \cdot 3^1 + a_2 \cdot 3^2 + a_3 \cdot 3^3 + a_4 \cdot 3^4 \leq 242 < 2^8$$

Wtedy do zapisu liczby B zgodnie z zależnością (3.1) wystarczy $b(3^5) = 8$ bitów. Daje to średnią liczbę bitów potrzebną do reprezentacji pojedynczej cyfry ternarnej równą $\bar{b}_5(3) = 1.6$, co oznacza, że efektywność wzrasta do $E_{3(5)} = 99.06016\%$.

3.2. Zastosowanie kodu dwufazowego do kompresji liczb B

W poprzednim punkcie do reprezentacji liczby B był używany kod o stałej długości $b(r^q)$ bitów. Dla dalszego wzrostu efektywności kompresji (efektywności reprezentacji

cyfr r -narnych) możemy zastosować kod dwufazowy dla liczby B , która nie jest potęgą dwójki. Pozwoli to na zmniejszenie średniej długości jednej cyfry r -narnej do:

$$\underline{b}_q(r) = \frac{\underline{b}(r^q)}{q} = \frac{b(r^q) + 1 - \frac{2^{b(r^q)}}{r^q}}{q} \quad (3.9)$$

Otrzymujemy kod o zmiennej długości symboli kodowych (słowa kodu mają $b(r^q)$ lub $b(r^q) - 1$ bitów), dla którego wzór na zmodyfikowaną procentową efektywność kompresji przyjmuje postać daną wzorem:

$$\underline{E}_{r(q)} = \frac{\log_2 r}{\underline{b}_q(r)} \cdot 100\% \quad (3.10)$$

Dla liczby B złożonej z $q = 5$ cyfr ternarnych, dzięki użyciu kodu dwufazowego, średnia długość jednej cyfry ternarnej zmniejsza się z $\bar{b}_5(3) = 1.6$ do $\underline{b}_5(3) = 1.58930$ bita, natomiast efektywność wzrasta z $E_{3(5)} = 99.06016\%$ do $\underline{E}_{3(5)} = 99.72706\%$.

3.3. Problem doboru długości bloków elementów r -narnych (q)

Istotnym problemem jest dobór odpowiedniej wartości q pozwalającej uzyskać wysoką efektywność. Jeżeli przyjmiemy, że wartość $b(r^q)$ jest stała w przedziale $r^q = \overline{2^{b(r^q)-1} + 1; 2^{b(r^q)}}$ i wynosi $b(r^q) = \lceil \log_2 r^q \rceil$, to efektywność $E_{r(q)}$ jest w tym przedziale funkcją rosnącą. Widać więc, iż im wartość r^q jest bliżej potęgi dwójki (ale przy $r^q < 2^{b(r^q)}$), tym wyższa efektywność. Możemy zatem zdefiniować parametr $U_{r(q)}$ informujący o trafności doboru wartości q .

Definicja 3.1 Procentową sprawnością $U_{r(q)}$ konwersji q cyfrowej liczby r -narnej do postaci binarnej liczby B nazywamy liczbę daną wzorem:

$$U_{r(q)} = \hat{U}_{r(q)} \cdot 100\% = \frac{r^q}{2^{b(r^q)}} \cdot 100\% \quad (3.11)$$

W punkcie 3.5 zostanie przedstawiony algorytm doboru wzorcowych wartości q , w którym decyzje podejmowane są w oparciu o analizę sprawności $U_{r(q)}$. Dla pojedynczej cyfry ternarnej przy $q = 1$ sprawność konwersji wynosi zaledwie $U_{3(1)} = 75\%$, natomiast dla $q = 5$ mamy już $U_{3(5)} = 94.92188\%$. Jeżeli podwoimy liczbę cyfr ternarnych ($q = 10$) w liczbie

B , to do jej zapisu będziemy potrzebować $b(3^{10}) = 16$ bitów. Daje to średnio $\bar{b}_{10}(3) = 1.6$ bita na jedną cyfrę ternarną, więc efektywność wynosi $E_{3(10)} = \frac{\log_2 3}{\bar{b}_{10}(3)} \cdot 100\% = 99.06016\%$

i jest taka sama jak dla $q = 5$. Jednak sprawność konwersji wynosi $U_{3(10)} = \frac{3^{10}}{2^{16}} \cdot 100\% = \left(\frac{3^5}{2^8}\right)^2 \cdot 100\% = 90.10162\%$ i jest mniejsza niż $U_{3(5)}$. Wzrost q spowodował spadek sprawności przy zachowaniu tej samej efektywności. Nie ma zatem liniowej zależności między sprawnością konwersji a efektywnością kompresji.

Rozważmy kolejne porównanie dotyczące liczb ternarnych. Dla $q = 41$ mamy $b(3^{41}) = 65$, co przy sprawności $U_{3(41)} = 98.86026\%$ daje efektywność $E_{3(41)} = 99.97456\%$. Zwiększenie q do 42 cyfr powoduje spadek sprawności do $U_{3(42)} = 74.14519\%$ oraz spadek efektywności do $E_{3(42)} = 99.35586\%$. Tabela 3.1 zawiera zestaw parametrów dla liczb B złożonych z $q < 10$ cyfr ternarnych. Z tabeli tej wynika też, że wzrost q nie musi oznaczać wzrostu sprawności. Jednocześnie większa sprawność $U_{r(q)}$ nie zawsze musi oznaczać większej efektywności $E_{r(q)}$, bo np.:

$U_{3(5)} = 94.92188\% > U_{3(42)} = 74.14519\%$, ale $E_{3(5)} = 99.06016\% < E_{3(42)} = 99.35586\%$

Po dalszej analizie wyników przeprowadzonych badań jedynym wnioskiem gwarantującym wzrost efektywności jest jednocześnie wzrost wartości q i sprawności $U_{r(q)}$:

$$\text{jeżeli } q_2 > q_1 \text{ oraz } U_{r(q_2)} \geq U_{r(q_1)}, \text{ to } E_{r(q_2)} \geq E_{r(q_1)}$$

Tabela 3.1 Parametry dla liczby B złożonej z $q < 10$ cyfr ternarnych

q	$b(r^q)$	$\bar{b}_q(r)$	$U_{r(q)} [\%]$
1	2	2	75.00000
2	4	2	56.25000
3	5	$1\frac{2}{3}$	84.37500
4	7	$1\frac{3}{4}$	63.28125
5	8	$1\frac{3}{5}$	94.92188
6	10	$1\frac{2}{3}$	71.19141
7	12	$1\frac{5}{7}$	53.39356
8	13	$1\frac{5}{8}$	80.09033
9	15	$1\frac{2}{3}$	60.06775

3.4. Analiza efektywności i sprawności konwersji

Istnieje możliwość dokonania analizy sprawności konwersji w zależności od wartości q . Zgodnie z zależnością (3.1) można oszacować całkowitą liczbę $b(r^q)$:

$$\log_2 r^q \leq b(r^q) < \log_2 r^q + 1$$

$$2^{\log_2 r^q} \leq 2^{b(r^q)} < 2^{\log_2 r^q + 1}$$

$$r^q \leq 2^{b(r^q)} < 2 \cdot r^q$$

$$\frac{1}{r^q} \geq \frac{1}{2^{b(r^q)}} > \frac{1}{2 \cdot r^q}$$

Po pomnożeniu przez r^q i podstawieniu równania (3.11) otrzymujemy:

$$\frac{r^q}{r^q} \geq \hat{U}_r > \frac{r^q}{2 \cdot r^q}$$

$$\frac{1}{2} < \hat{U}_r \leq 1 \quad (3.12)$$

Z zależności (3.12) wynika, że procentowa sprawność konwersji ma stały zakres ograniczeń między 50% a 100% niezależnie od tego, jak duże będzie q [79]. Przeanalizujmy zatem zależność między wartością q a efektywnością kompresji. Zgodnie z zależnością (3.1) można oszacować całkowitą liczbę $b(r^q)$:

$$\log_2 r^q \leq b(r^q) < \log_2 r^q + 1$$

$$\frac{1}{q} \cdot \log_2 r^q \leq \bar{b}_q(r) < \frac{1}{q} \cdot (\log_2 r^q + 1)$$

$$\log_2 r \leq \bar{b}_q(r) < \log_2 r + \frac{1}{q}$$

$$\frac{1}{\log_2 r} \geq \frac{1}{\bar{b}_q(r)} > \frac{1}{\log_2 r + \frac{1}{q}}$$

Po obustronnym pomnożeniu przez $\log_2 r$ i podstawieniu równania (3.8) otrzymujemy:

$$\frac{\log_2 r}{\log_2 r} \geq \hat{E}_{r(q)} > \frac{\log_2 r}{\log_2 r + \frac{1}{q}}$$

$$\frac{\log_2 r}{\log_2 r + \frac{1}{q}} < \hat{E}_{r(q)} \leq 1 \quad (3.13)$$

Na podstawie zależności (3.13) można zauważyć, że ze wzrostem wartości q rośnie dolne ograniczenie efektywności [79]. Dla r nie będących potęgą dwójki stuprocentową efektywność uzyskać można dla q dążącego do nieskończoności:

$$\lim_{q \rightarrow \infty} \frac{\log_2 r}{\log_2 r + \frac{1}{q}} = 1 \quad (3.14)$$

Co oznacza, że średnia długość cyfry r -narnej ze wzrostem q zbliża się do wartości entropii dla r równoprawdopodobnych symboli. A zatem uzyskanie wysokiej efektywności wiąże się z dobraniem odpowiednio dużej wartości q . W praktycznych zastosowaniach powinniśmy używać możliwie małych wartości q gwarantujących założony poziom efektywności.

W podobny sposób możemy przeanalizować zależność między wartością q a efektywnością kompresji, gdy używamy kodu dwufazowego do kompresji liczb B . Zgodnie z zależnością (3.1) oraz górnym ograniczeniem Gallagera [27] można oszacować liczbę $\underline{b}(r^q)$:

$$\begin{aligned} \log_2 r^q &\leq \underline{b}(r^q) < \log_2 r^q + \frac{1}{r^q} + \sigma \\ \frac{1}{q} \cdot \log_2 r^q &\leq \underline{b}_q(r) < \frac{1}{q} \cdot (\log_2 r^q + \frac{1}{r^q} + \sigma) \\ \log_2 r &\leq \underline{b}_q(r) < \log_2 r + \frac{1}{q} \cdot (\frac{1}{r^q} + \sigma) \\ \frac{1}{\log_2 r} &\geq \frac{1}{\underline{b}_q(r)} > \frac{1}{\log_2 r + \frac{1}{q} \cdot (\frac{1}{r^q} + \sigma)} \end{aligned}$$

Po obustronnym pomnożeniu przez $\log_2 r$ i podstawieniu równania (3.10) oraz wartości $\sigma = 1 - \log_2 e + \log_2(\log_2 e) \approx 0.08607$ otrzymujemy:

$$\begin{aligned} \frac{\log_2 r}{\log_2 r} &\geq \hat{\underline{E}}_{r(q)} > \frac{\log_2 r}{\log_2 r + \frac{1}{q} \cdot (\frac{1}{r^q} + \sigma)} \\ \frac{\log_2 r}{\log_2 r + \frac{1}{q} \cdot (\frac{1}{r^q} + 0.08607)} &< \hat{\underline{E}}_{r(q)} \leq 1 \end{aligned} \quad (3.15)$$

Na podstawie zależności (3.15) można zauważyć, że ze wzrostem wartości q rośnie dolne ograniczenie efektywności. Dla r nie będących potęgą dwójki stu procentową efektywność uzyskać można dla q dążącego do nieskończoności:

$$\lim_{q \rightarrow \infty} \frac{\log_2 r}{\log_2 r + \frac{1}{q} \cdot (\frac{1}{r^q} + 0.08607)} = 1 \quad (3.16)$$

3.5. Algorytm wyznaczania wzorcowych długości bloków (q) elementów r -narnych

W punkcie 3.3 został przedstawiony przykład cyfr ternarnych, które dla $q = 5$ oraz $q = 10$ miały tę samą procentową efektywność, lecz cechowała je inna wartość sprawności.

To właśnie posługując się sprawnością konwersji możemy przedstawić poniższy *algorytm wyznaczania (optymalnych) wzorcowych wartości q* dla zadanej liczby r :

Na początku ustalamy pierwszą wzorcową wartość $q_{min} = 1$, dla której obecna maksymalna

procentowa sprawność konwersji $U_{r(1)} = \frac{r}{2^{b(r)}} \cdot 100\%$, a procentowa efektywność kompre-

sji wynosi $E_{r(1)} = \frac{\log_2 r}{b(r)} \cdot 100\%$.

Dla każdej kolejnej całkowitej wartości q_n testujemy sprawność $U_{r(q_n)}$, która może:

- być mniejsza lub równa aktualnej wzorcowej;
- być większa od wzorcowej, wtedy q_n staje się nową wartością wzorcową, gdyż efektywność $E_{r(q_n)}$ też wzrasta.

Dla wzorcowych wartości q mamy taką własność, że nie mogą być jednocześnie parzyste wartości q oraz $b(r^q)$. W sytuacji, gdy obie liczby byłyby parzyste, to istnieje mniejsza wartość q dająca tę samą efektywność. Jeśli $q = 2 \cdot k$ oraz $b(r^q) = 2 \cdot m$, to istnieją mniejsze wartości q_i o tej samej efektywności:

$$E_{r(2 \cdot k)} = \frac{\log_2 r}{\frac{b(r^{2 \cdot k})}{2 \cdot k}} = \frac{\log_2 r}{\frac{2 \cdot m}{2 \cdot k}} = \frac{\log_2 r}{\frac{m}{k}} = E_{r(k)},$$

czyli $q_i = k$ oraz $b(r^{q_i}) = m$.

Tabele 3.2, 3.3 oraz 3.4 przedstawiają kolejne uzyskiwane wzorcowe wartości q odpowiednio dla $r = 3$, $r = 5$ oraz $r = 7$ [84]. Z ostatniego wiersza tabeli 3.2 wynika, że dla liczby B , której długość wynosi 301994 bity, różnica między entropią a średnią długością liczby ternarnej jest zauważalna dopiero na jedenastym miejscu po przecinku. Podobnie wygląda sytuacja dla ostatniego wiersza dwóch kolejnych tabel. W tabeli 3.5 znajdują się parametry dla przykładowych wzorcowych par (r_i, q_i) takich, że $r < 30$ oraz $q < 87$ [75, 78].

Jeżeli do przykładu 2.2 zastosujemy odpowiednią wzorcową wartość q , to dzięki użyciu blokowej konwersji ternarny kod Huffmana uzyska lepszą efektywność od binarnego kodu Huffmana, czyli mniejszą średnią długość słowa kodowego $L_3 = 1.7 \cdot \bar{b}_q(r) < L_2 = 2.7$. Najmniejszą wzorcową wartością q spełniającą ten warunek jest $q = 29$. Wówczas efektywność kompresji wynosi $E_{3(29)} = 97.23263\%$.

Tabela 3.2 Parametry dla liczby B złożonej z $q < 200000$ cyfr, przy $r = 3$

q	$b(r^q)$	$\bar{b}_q(r)$	$U_{r(q)} [\%]$
1	2	2.000000000000000	79.248125036058
3	5	1.666666666666667	95.097750043269
5	8	1.600000000000000	99.060156295072
17	27	1.58823529411765	99.793935230591
29	46	1.58620689655172	99.921548958508
41	65	1.58536585365854	99.974557737796
94	149	1.58510638297872	99.990922864288
147	233	1.58503401360544	99.995488242923
200	317	1.585000000000000	99.997634114899
253	401	1.58498023715415	99.998880968193
306	485	1.58496732026144	99.999695921788
971	1539	1.58496395468589	99.999908265123
1636	2593	1.58496332518337	99.999947982253
2301	3647	1.58496305953933	99.999964742511
2966	4701	1.58496291301416	99.999973987214
3631	5755	1.58496282015974	99.999979845674
4296	6809	1.58496275605214	99.999983890411
4961	7863	1.58496270913122	99.999986850790
5626	8917	1.58496267330252	99.999989111329
6291	9971	1.58496264504848	99.999990893960
6956	11025	1.58496262219666	99.999992335749
7621	12079	1.58496260333290	99.999993525920
8286	13133	1.58496258749698	99.999994525055
8951	14187	1.58496257401408	99.999995375732
9616	15241	1.58496256239601	99.999996108750
10281	16295	1.58496255228091	99.999996746942
10946	17349	1.58496254339485	99.999997307590
11611	18403	1.58496253552666	99.999997804018
12276	19457	1.58496252851092	99.999998246661
12941	20511	1.58496252221621	99.999998643813
13606	21565	1.58496251653682	99.999999002143
14271	22619	1.58496251138673	99.999999327077
14936	23673	1.58496250669523	99.999999623078
15601	24727	1.58496250240369	99.999999893844
47468	75235	1.58496250105334	99.999999979041
79335	125743	1.58496250078780	99.999999995795
190537	301994	1.58496250072164	99.99999999969

Najlepszy rezultat uzyskamy, jeśli będziemy używać wzorcowych wartości q dla liczb B , a następnie stosować kod dwufazowy do kompresji liczby B . Tabela 3.5 przedstawia porównanie efektywności kodów o stałej i zmiennej długości dla przykładowych wzorcowych par (r_i, q_i) . Dzięki zastosowaniu odpowiednich wzorcowych wartości q oraz kodu dwufazowego dla liczb B łatwo uzyskujemy procentowe efektywności kompresji

$\underline{E}_{r(q)} > 99.7\%$ (patrz tabela 3.5). Zgodnie z tabelą 2.4, nawet dla przypadkowych wartości q takich, że $r^q > 512$ mamy efektywność przekraczającą 99.0988%.

Tabela 3.3 Parametry dla liczby B złożonej z $q < 100000$ cyfr, przy $r = 5$

q	$b(r^q)$	$\bar{b}_q(r)$	$U_{r(q)} [\%]$
1	3	3.000000000000000	77.397603162912
2	5	2.500000000000000	92.877123795494
3	7	2.333333333333333	99.511204066601
31	72	2.32258064516129	99.971904085428
59	137	2.32203389830508	99.995443502448
205	476	2.32195121951220	99.999004086536
351	815	2.32193732193732	99.999602614167
497	1154	2.32193158953722	99.999849493849
643	1493	2.32192846034215	99.999984260722
4647	10790	2.32192812567248	99.999998674157
8651	20087	2.32192810079760	99.999999745460
21306	49471	2.32192809537220	99.999999979119
97879	227268	2.32192809489267	99.99999999771

Tabela 3.4 Parametry dla liczby B złożonej z $q < 100000$ cyfr, przy $r = 7$

q	$b(r^q)$	$\bar{b}_q(r)$	$U_{r(q)} [\%]$
1	3	3.000000000000000	93.578497401920
6	17	2.833333333333333	99.083114896151
11	31	2.81818181818182	99.615819814947
16	45	2.812500000000000	99.817063895381
21	59	2.80952380952381	99.922802310525
26	73	2.80769230769231	99.987983525339
135	379	2.80740740740741	99.998130469070
244	685	2.80737704918033	99.999211822198
353	991	2.80736543909348	99.999625377027
462	1297	2.80735930735931	99.999843792646
571	1603	2.80735551663748	99.999978820642
2964	8321	2.80735492577598	99.99999867549
91313	256348	2.80735492208119	99.99999999160

Tabela 3.5 Parametry dla przykładowych wzorcowych q przy $r < 30$

r	q	$b(r^q)$	$\underline{b}(r)$	$\underline{E} \text{ [%]}$	$\overline{b}_q(r)$	$E_{r(q)} \text{ [%]}$	$U_{r(q)} \text{ [%]}$	$\underline{b}_q(r)$	$\underline{E}_{r(q)} \text{ [%]}$
3	5	8	$1\frac{2}{3}$	95.09775	$1\frac{3}{5}$	99.06016	94.92188	1.5893	99.72706
3	41	65	$1\frac{2}{3}$	95.09775	$1\frac{24}{41}$	99.97456	98.86026	1.5851	99.99229
5	3	7	$2\frac{2}{5}$	96.74700	$2\frac{1}{3}$	99.51120	97.65625	2.3253	99.85356
5	31	72	$2\frac{2}{5}$	96.74700	$2\frac{10}{31}$	99.97190	98.60761	2.3221	99.99151
7	11	31	$2\frac{6}{7}$	98.25742	$2\frac{9}{11}$	99.61582	92.07645	2.8104	99.89312
7	16	45	$2\frac{6}{7}$	98.25742	$2\frac{13}{16}$	99.81706	94.45367	2.8088	99.94749
9	17	54	$3\frac{2}{9}$	98.37698	$3\frac{3}{17}$	99.79394	92.57696	3.1718	99.94234
11	13	45	$3\frac{6}{11}$	97.57371	$3\frac{6}{13}$	99.93914	98.11945	3.4601	99.98172
13	17	63	$3\frac{10}{13}$	98.17493	$3\frac{12}{17}$	99.85313	93.78800	3.7012	99.95823
15	11	43	$3\frac{14}{15}$	99.32773	$3\frac{10}{11}$	99.94371	98.33634	3.9076	99.98305
17	10	41	$4\frac{2}{17}$	99.26696	$4\frac{1}{10}$	99.69422	91.67679	4.0909	99.91546
17	11	45	$4\frac{2}{17}$	99.26696	$4\frac{1}{11}$	99.91576	97.40659	4.0885	99.97491
19	4	17	$4\frac{6}{19}$	98.42759	$4\frac{1}{4}$	99.95125	99.42703	4.2486	99.98513
21	5	22	$4\frac{10}{21}$	98.12624	$4\frac{2}{5}$	99.82540	97.37256	4.3946	99.94798
23	13	59	$4\frac{14}{23}$	98.15276	$4\frac{7}{13}$	99.67170	87.43637	4.5274	99.91504
23	19	86	$4\frac{14}{23}$	98.15276	$4\frac{10}{19}$	99.93916	96.43824	4.5244	99.98210
25	3	14	$4\frac{18}{25}$	98.38678	$4\frac{2}{3}$	99.51120	95.36743	4.6508	99.85768
27	13	62	$4\frac{22}{27}$	98.75536	$4\frac{10}{13}$	99.69925	87.87578	4.7586	99.92161
29	8	39	$4\frac{26}{29}$	99.21229	$4\frac{7}{8}$	99.65089	90.99429	4.8626	99.90442

4. Mieszane kody Huffmana

W tym rozdziale zostaną omówione podstawowe własności mieszanych kodów Huffmana oraz ich porównanie z binarnym kodem Huffmana. Zdefiniowana zostanie konstrukcyjna klasa kodów mieszanych, określająca ograniczenia konstrukcyjne w postaci parametrów mieszanego kodu Huffmana oraz przedstawiona przykładowa struktura pliku wynikowego dla kodów mieszanych. W punkcie 4.5 zostanie uzasadnione ograniczenie stosowania kodów r -narnych do wartości r będących liczbą pierwszą. Zostaną także przedstawione wyniki testów, z których wynika, że dla źródeł o $n \geq 12$ symbolach przypadki gdy binarny kod Huffmana jest optymalny nie stanowią nawet jednego procenta źródeł. Punkt 4.7 zawiera trzy przykłady ilustrujące porównanie efektywności binarnego oraz mieszanego kodu Huffmana.

4.1. Podstawy konstrukcji mieszanych kodów Huffmana

Przedstawione w poprzednim rozdziale efektywne reprezentacje symboli r -narnych służą do tworzenia efektywnych realizacji *mieszanych kodów Huffmana*. Konstrukcja drzew kodowych dla tych kodów polega na łączeniu w każdym z kolejno tworzonych węzłów dowolnej odpowiednio dobranej liczby r_i najmniej prawdopodobnych potomków tak, aby zachowana została własność rodzeństwa (patrz definicja 4.4). *Skróty opis konstrukcji* takiego drzewa można przedstawić jako ciąg kroków, w których w każdym tworzymy węzeł w_i łączący r_{w_i} potomków: $r_w = \{r_{w_1}, r_{w_2}, r_{w_3}, \dots\}$. W praktyce efektywne kodowanie tą metodą można uzyskać dzięki *stosowaniu osobnych buforów* dla każdej wartości r_i , do których są zapisywane r_i -narne elementy składowe słów kodu mieszanego. Aby przybliżyć sposób funkcjonowania takiego kodu, powróćmy do przykładu 2.2. Użycie kodu ternarnego dało teoretyczną średnią długość słowa kodowego $L_3 = 1.7 \cdot \log_2 3 = 2.69444$, co odpowiada teoretycznej efektywności $E_3 = 97.30897\%$. Jeżeli użyjemy dokładnie tego samego drzewa kodowego, lecz z zastrzeżeniem, że dwa symbole o najmniejszych prawdopodobieństwach p_1 i p_2 połączymy węzłem binarnym zamiast ternarnym, to teoretyczna średnia długość słowa kodu mieszanego wyniesie:

$$\begin{aligned} L_{2,3} &= \log_2 3 + \log_2 3 \cdot (p_1 + p_2 + p_3 + p_4 + p_5 + p_6 + p_7) + 1 \cdot (p_1 + p_2) \\ L_{2,3} &= p_1 + p_2 + \log_2 3 \cdot (2 - p_8) = 2.63594 \end{aligned}$$

Co daje teoretyczną procentową efektywność $E_{mix} = E_{2,3} = 99.46843\%$, gdzie cyfry w indeksie oznaczają użyte w kodzie liczby r_i potomków w węzłach mieszanego drzewa Huffmana. W praktyce możemy uzyskać przybliżenie takiej efektywności dzięki użyciu blokowej konwersji q cyfrowej liczby B opisanej w punkcie 3.1. Zaproponowana w punkcie 3.5 wzorcowa wartość $q = 29$ pozwala na uzyskanie średniej długości kodu mieszanego (binarno-ternarnego) $L_{2,3} = p_1 + p_2 + \bar{b}_{29}(3) \cdot (2 - p_8) = 2.63793$, co odpowiada efektywności $E_{2,3} = 99.39335\%$. Kształt tego drzewa przedstawia rysunek 2.4. Kod mieszany ma tę własność, że w odróżnieniu od kodów wielowartościowych jest zawsze *kodem kompletnym* (żaden z węzłów nie może posiadać nieużywanych potomków). Mieszany kod Huffmana dla tego przykładu przedstawia tabela 4.1. Najprostszy koder działa następująco: jeśli np. chcemy zakodować symbol o prawdopodobieństwie p_2 , to zapisujemy do bufora ternarnego dwie cyfry ternarne „1₃” oraz „2₃”, a następnie do bufora binarnego bit „1”. Po zakodowaniu całego strumienia danych używamy konwersji blokowej dla cyfr znajdujących się w buforze ternarnym. *Przykładowa struktura pliku* dla takiego kodu składa się z nagłówka zawierającego zestaw licznosci, następnie znajduje się wskaźnik długości bufora binarnego, po nim jest umieszczony cały bufor binarny, a na końcu bufor ternarny w postaci ciągu liczb B . Gdy istnieją inne bufor-y r -narne, to umieszcza się je kolejno za buforem ternarnym wraz ze wskaźnikami długości każdego bufora. Istnieje też możliwość użycia koder-a, który nie musi oczekiwać, aż na wejściu pojawi się informacja o końcu kodowanego pliku. Wystarczy, aby każdy z używanych buforów r_i -narnych zapełnił się q_i cyframi. Wówczas można zakodować pierwszy zestaw liczb B_i i wysłać je na wyjście. Zalety takiej wersji koder-a omówione zostaną w punkcie 6.5.

Tabela 4.1 Kody Huffmana z przykładu 2.2

p_i	Kod mieszany	Kod binarny
$p_1 = 0.05$	1 ₃ 2 ₃ 0 ₂	1010
$p_2 = 0.05$	1 ₃ 2 ₃ 1 ₂	1011
$p_3 = 0.10$	0 ₃ 0 ₃	000
$p_4 = 0.10$	0 ₃ 1 ₃	001
$p_5 = 0.10$	0 ₃ 2 ₃	010
$p_6 = 0.10$	1 ₃ 0 ₃	011
$p_7 = 0.10$	1 ₃ 1 ₃	100
$p_8 = 0.40$	2 ₃	11

Na podstawie tabeli 4.1 możemy przedstawić zasadę działania dekodera:

1° Wczytaj z bufora ternarnego pierwszą cyfrę.

2° Jeśli jest to „2₃”, to mamy zdefiniowany symbol o prawdopodobieństwie p_8 .

W przeciwnym przypadku pobierana jest kolejna cyfra ternarna.

{

Jeśli wczytane cyfry ternarne tworzą ciąg „1₃2₃”, to wczytujemy z bufora binarnego jeden bit decydujący o zdekodowaniu symbolu o prawdopodobieństwie p_1 lub p_2 odpowiednio dla bitu o wartości „0₂” lub „1₂”.

W przeciwnym wypadku możemy na podstawie tych dwóch wczytanych cyfr ternarnych {„0₃0₃”, „0₃1₃”, „0₃2₃”, „1₃0₃”, „1₃1₃”} zdekodować jeden z pięciu symboli o prawdopodobieństwie odpowiednio od p_3 do p_7 .

}

3° Jeśli są symbole na wejściu wróć do punktu 1°.

Użycie wielu buforów w jednym koderze nie jest popularną metodą wśród podstawowych metod kompresji. Podział na buforów kodowane osobno z dostosowaniem najlepszej metody kompresji dla każdego z nich został zaprezentowany np. w pracy [90], gdzie każdy z siedmiu bitów kodu ASCII zapisywany był do osobnego bufora.

Przykład 4.1

Dane jest ośmiosymbolowe źródło S o rozkładzie przedstawionym w tabeli 4.2, dla którego tworzymy binarno-ternarny kod Huffmana. Wartość entropii tego źródła wynosi $H(S) = 2.68973$ bita na symbol. Średnia długość binarnego kodu Huffmana wynosi $L_2 = 2.76$, co daje procentową efektywność $E_2 = 97.45394\%$, natomiast dla kodu binarno-ternarnego przy $q = 5$ średnia wynosi $L_{2,3} = 2.714$, co odpowiada efektywności $E_{2,3} = 99.10570\%$. Rysunek 4.1 przedstawia drzewo kodu binarno-ternarnego, które można opisać skrótowo w następujący sposób: $r_w = \{3, 2, 2, 2, 3\}$. Jeżeli dla tego samego źródła użyjemy kodu o trzech buforach, tzn. oprócz kodu binarnego ($r_0 = 2$) zastosujemy kod ternarny ($r_1 = 3$) z parametrem $q_1 = 5$ oraz pięciowartościowy ($r_2 = 5$) z parametrem $q_2 = 3$, to otrzymamy jeszcze wyższą efektywność. Drzewo kodowe o skrótowym zapisie $r_w = \{5, 2, 3\}$ przedstawia rysunek 4.2 [84]. Średnia długość takiego kodu mieszanego wynosi:

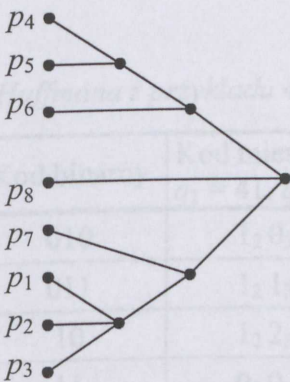
$$L_{mix} = L_{2,3,5} = \frac{8}{5} + \frac{7}{3} \cdot \frac{6+6+7+7+7}{100} + \frac{17+17}{100} = 2.71$$

Wyliczenia L_{mix} oparte na podstawie drzewa kodu mieszanego uwzględniają długości gałęzi, czyli wartości oznaczające średnią liczbę bitów $\bar{b}_q(r)$ potrzebną do zakodowania jednej cyfry r -narnej (przy drzewie binarnym długość gałęzi nie była brana pod uwagę, ponieważ jej wartość wynosiła 1). Dla takiego kodu otrzymujemy procentową efektywność równą $E_{mix} = E_{2,3,5} = 99.25198\%$. Jeżeli zwiększymy parametry wzorcowe konwersji binarnej odpowiednio do $q_1 = 41$ oraz $q_2 = 31$, to otrzymamy średnią $L_{mix} = 2.69182$, co daje wzrost efektywności do $E_{mix} = 99.92240\%$. Waga zysku wynosi wówczas $W = 97.02761\%$, co oznacza znaczne zbliżenie się średniej L_{mix} do wartości entropii w porównaniu z wartością L_2 (patrz wzór (1.9)). Tabela 4.2 zawiera symbole dla kodu binarno-ternarnego, kodu mieszanego przy $r_1 = 3, r_2 = 5$, a także binarnego kodu Huffmana.

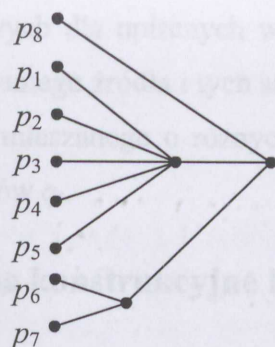
Jak widać z powyższego przykładu właściwy dobór par wzorcowych parametrów (r_i, q_i) wpływa na uzyskanie odpowiednio wysokiej efektywności E_{mix} mieszanego kodu Huffmana. Kolejny przykład wykaże, iż dobór wzorcowych wartości q ma też wpływ na sam kształt drzewa kodowego [79].

Tabela 4.2 Kody Huffmana z przykładu 4.1

p_i	Kod binarno-ternarny	Kod mieszany $r_1 = 3, r_2 = 5$	Kod binarny
$p_1 = 0.06$	$2_3 1_2 0_3$	$1_3 0_5$	1110
$p_2 = 0.06$	$2_3 1_2 1_3$	$1_3 1_5$	1111
$p_3 = 0.07$	$2_3 1_2 2_3$	$1_3 2_5$	0000
$p_4 = 0.07$	$0_3 0_2 0_2$	$1_3 3_5$	0001
$p_5 = 0.07$	$0_3 0_2 1_2$	$1_3 4_5$	110
$p_6 = 0.17$	$0_3 1_2$	$2_3 0_2$	001
$p_7 = 0.17$	$2_3 0_2$	$2_3 1_2$	10
$p_8 = 0.33$	1_3	0_3	01



Rysunek 4.1 Drzewo kodu binarno-ternarnego z przykładu 4.1



Rysunek 4.2 Drzewo mieszanego kodu Huffmana z przykładu 4.1

Przykład 4.2

Dane jest pięciosymbolowe źródło S o następującym rozkładzie prawdopodobieństw $P = \{ p_1 = p_2 = \frac{20}{121}, p_3 = p_4 = p_5 = \frac{27}{121} \}$, dla którego wartość entropii wynosi $H(S) = 2.30710$ bita na symbol. Tworzymy mieszany kod Huffmana dla parametrów $r_1 = 3, q_1 = 5$ oraz $r_2 = 5, q_2 = 3$. Najkrótsza średnia długość dla mieszanego (ale niebinarnego) kodu Huffmana $r_w = \{3, 2, 2\}$ o takich parametrach wynosi $L_{2,3} = 2.33223$ i jest dłuższa niż dla binarnego kodu Huffmana $L_2 = 2.33058$. Zatem binarny kod Huffmana jako podzbiór wszystkich kodów mieszanych staje się kodem optymalnym przy założonych wcześniej parametrach. Jeśli jednak zwiększymy wartość q_1 z 5 do 41 cyfr ternarnych, to optymalny będzie kod mieszany $r_w = \{3, 2, 2\}$, dla którego średnia wynosi $L_{2,3} = 2.32413$. Natomiast gdy zwiększymy wartość q_2 z 3 do 31 dla $r_2 = 5$, to uzyskamy optymalny kod mieszany dla $r_w = \{5\}$, dla którego $L_5 = \frac{72}{31} = 2.32258$. Dla takiego kodu procentowa efektywność wynosi $E_{mix} = 99.33359\%$.

Tabela 4.3 Kody Huffmana z przykładu 4.2 z parametrami $r_1 = 3, r_2 = 5$

p_i	Kod binarny	Kod mieszany $q_1 = 41, q_2 = 3$	Kod mieszany $q_1 = 41, q_2 = 31$
$p_1 = \frac{20}{121}$	010	$1_2 0_3$	0_5
$p_2 = \frac{20}{121}$	011	$1_2 1_3$	1_5
$p_3 = \frac{27}{121}$	10	$1_2 2_3$	2_5
$p_4 = \frac{27}{121}$	11	$0_2 0_2$	3_5
$p_5 = \frac{27}{121}$	00	$0_2 1_2$	4_5

Tablice słów kodowych dla opisanych wyżej parametrów przedstawia tabela 4.3. Wynika z niej, że dla tego samego źródła i tych samych buforów $r_1 = 3$, $r_2 = 5$ mogą istnieć drzewa optymalnego kodu mieszanego o różnych kształtach w zależności od zastosowanych wzorcowych parametrów q_i .

4.2. Ograniczenia konstrukcyjne kodu mieszanego

Przykłady z poprzedniego punktu sygnalizują potrzebę dokładniejszego opisu mieszanego kodu Huffmana. Służy do tego poniższa definicja.

Definicja 4.1 *Konstrukcyjną klasą kodów mieszanych* nazywamy zbiór kodów, dla których koder i dekoder spełnia określone ograniczenia konstrukcyjne. Do ograniczeń tych zaliczamy zestaw parametrów wykorzystywanych do utworzenia optymalnego kodu mieszanego, czyli liczbę m używanych buforów r_i -narnych, zestaw odpowiadających im parametrów $r_0, r_1, \dots, r_{m-1}$ i związanych z nimi wzorcowych wartości $q_1, q_2, \dots, q_{m-1}$ (oraz $b(r_i^{q_i})$), a także inne opcjonalne parametry potrzebne podczas dekodowania.

Wartość $b(r_i^{q_i})$ oznacza całkowitą liczbę bitów potrzebną do reprezentacji liczby B i może być wyznaczana bezpośrednio z zależności (3.1). Natomiast do parametrów opcjonalnych można zaliczyć skrótowy opis konstrukcji drzewa r_w , (co pozwala pominąć etap poszukiwania optymalnego w danej klasie drzewa podczas etapu dekompresji), a także to, czy zastosowano kod dwufazowy do kompresji liczb B . Jeżeli dana jest klasa $m = 1$, $r_0 = 2$, to mamy do czynienia z binarnym kodem Huffmana. Jeśli np. zdefiniujemy klasę kodów mieszanych dla przykładu 4.1 w następujący sposób:

„ $m = 3$, $r_0 = 2$, $r_1 = 3$, $r_2 = 5$, $q_1 = 41$, $q_2 = 31$, używany kod dwufazowy”,

to otrzymamy średnią $\underline{L}_{mix} = 2.69139$ bita na symbol oraz efektywność takiego kodu równą $\underline{E}_{mix} = 99.93843\%$. Na podstawie cech kodu opisanych w pracy [90]:

Definicja 4.2 *Idealną klasą kodu mieszanego* nazywamy klasę kodów bez ograniczeń konstrukcyjnych. Dla takiej klasy istotny jest jedynie zestaw parametrów r_i określających taki kod, w którym do reprezentacji pojedynczej cyfry r_i -narnej wystarcza $\log_2 r_i$ bitów.

Pomijając przypadki źródeł, dla których prawdopodobieństwa są postaci 2^{-l} , dla całkowitych dodatnich wartości l , klasa idealna jest pojęciem teoretycznym i stanowi granicę możliwości uzyskania wzrostu efektywności przez kod mieszany o zdefiniowanym zestawie m buforów r_i -narnych. W przypadku klasy idealnej nie uwzględnia się użycia kodu dwufazo-

wego, gdyż nie powoduje on wzrostu efektywności. Jeżeli np. zdefiniujemy idealną klasę kodów mieszanych dla przykładu 4.1 jako $m = 3$, $r_0 = 2$, $r_1 = 3$, $r_2 = 5$, to teoretyczna procentowa efektywność kodu mieszanego wyniesie $E_{mix} = 99.94538\%$.

4.3. Podstawowe własności mieszanych kodów Huffmana

Posługując się definicją klasy kodów można uszczegółowić pojęcie kodu mieszanego podając warunki, dla których kod mieszany jest optymalny:

Definicja 4.3 Dla źródła S o n symbolach *optymalnym* w danej klasie mieszanych kodem Huffmana jest kod o najkrótszej średniej długości słowa kodowego L_{mix} , dla którego każdy z węzłów drzewa kodowego nie będący liściem może łączyć dowolną określoną w danej klasie kodów liczbę $2 \leq r_i \leq n$ potomków.

W przypadku klasy bez ograniczeń (klasy idealnej) możemy mówić o *optymalnym kodzie mieszanym*. Realizacja koderów i dekodek kodu mieszanego wymaga (zgodnie z definicją klasy kodów) użycia m buforów r_i -narych. Przy czym oprócz $r_0 = 2$ pozostałe wartości r_i są liczbami nieparzystymi, gdyż każdą parzystą liczbę $r > 0$ można rozłożyć na iloczyn liczb parzystej i nieparzystej $r = 2^s \cdot r_{niep}$, gdzie s jest dodatnią liczbą całkowitą i 2^s jest maksymalną potęgą dwójki będącą dzielnikiem r , a r_{niep} jest dodatnią liczbą nieparzystą.

Przykład 4.3

Dane jest źródło złożone z $r = 12$ równoprawdopodobnych symboli, wówczas $12 = 2^2 \cdot 3$. Dla takiego źródła możemy utworzyć kod mieszany przedstawiony w tabeli 4.4, którego symbole składają się z $s = 2$ bitów i jednej cyfry ternarnej ($r_{niep} = 3$).

Tabela 4.4 Kod mieszany z przykładu 4.3

Symbol	Kod mieszany
0	0 ₂ 0 ₂ 0 ₃
1	0 ₂ 0 ₂ 1 ₃
2	0 ₂ 0 ₂ 2 ₃
3	0 ₂ 1 ₂ 0 ₃
4	0 ₂ 1 ₂ 1 ₃
5	0 ₂ 1 ₂ 2 ₃
6	1 ₂ 0 ₂ 0 ₃
7	1 ₂ 0 ₂ 1 ₃
8	1 ₂ 0 ₂ 2 ₃
9	1 ₂ 1 ₂ 0 ₃
10	1 ₂ 1 ₂ 1 ₃
11	1 ₂ 1 ₂ 2 ₃

Zdefiniujmy funkcję $g(r)$ jako sumę prawdopodobieństw p_{w_j} wszystkich n_r węzłów r_i -narnych w mieszanym drzewie kodowym:

$$g(r) = \sum_{j=1}^{n_r} p_{w_j} \quad (4.1)$$

Wówczas prawdopodobieństwo wystąpienia symbolu r_i -narnego w strumieniu wyjściowym kodu mieszanego wynosi:

$$p(r) = \frac{g(r)}{\sum_{i=0}^{m-1} g(r_i)} \quad (4.2)$$

Gdzie m jest liczbą wszystkich różnych elementów r_i -narnych występujących w danym kodzie mieszanym. Waga $w(r)$ cyfry r -narnej w danym kodzie mieszanym zależy od wartości $g(r)$ oraz liczby bitów $\bar{b}_q(r)$ potrzebnej do reprezentacji pojedynczej cyfry r -narnej i ma następującą postać:

$$w(r) = g(r) \cdot \bar{b}_q(r) \quad (4.3)$$

Suma wszystkich wag daje średnią długość kodu mieszanego L_{mix} :

$$L_{mix} = \sum_{i=0}^{m-1} w(r_i) \quad (4.4)$$

Definicja 4.4 Kompletne drzewo mieszane, którego węzły zawierają liczniki częstości, tj. liczniki zliczające, ile razy symbole źródłowe występują w kodowanym komunikacie (pliku) ma *własność rodzeństwa*, jeśli każdy węzeł oprócz liści ma r_i potomków i przechodzenie wszerek od strony prawej do lewej (począwszy od wierzchołka, poprzez r_{w_k} jego potomków, aż do ostatniego z liści) daje w wyniku listę węzłów o nierosnących licznikach [21].

Twierdzenie 4.1 (Twierdzenie Fallera-Gallagera) Drzewo mające właściwość rodzeństwa jest drzewem Huffmana.

Dowód twierdzenia znajduje się w pracy [21]. Kod mieszany określamy mianem *mieszanego kodu Huffmana* na podstawie powyższego twierdzenia, a także ze względu na duże prawdopodobieństwa własności do binarnego kodu Huffmana:

- kod mieszany jest kodem przedrostkowym, gdyż można go przedstawić w postaci drzewa kodowego, którego liście odpowiadają poszczególnym symbolom źródła;

- jeżeli $p_j > p_i$, to $l_j \leq l_i$;
- dwa symbole (lub r_i symboli) o najmniejszych prawdopodobieństwach mają słowa kodowe o tej samej, maksymalnej długości $l_{\max}$ i różnią się tylko ostatnim bitem (lub cyfrą r_i -narną, gdy węzeł drzewa nie będący liściem łączy ze sobą r_i symboli źródła o najniższym prawdopodobieństwie);
- mając dany skrótowy opis drzewa kodu mieszanego $r_w = \{r_{w_1}, r_{w_2}, r_{w_3}, \dots\}$ łączymy w każdym i -tym kroku r_{w_i} symboli lub węzłów o aktualnie najmniejszym prawdopodobieństwie według algorytmu Huffmana (dla kodu binarnego r_{w_i} było zawsze równe dwa);
- kod mieszany jest zawsze kodem kompletnym.

4.4. Dolne ograniczenie średniej długości słowa kodu Huffmana

Dla zachowania wysokiej efektywności kodu mieszanego można uznać, że im wyższa wartość p_n najbardziej prawdopodobnego symbolu, tym powinna być mniejsza liczba r symboli lub węzłów łączonych w korzeniu drzewa. Najmniejsza wartość $r = 2$ pozwala uzyskać wysoką efektywność kompresji dla największej wartości $p_n \approx 0.5$. Ze wzrostem p_n powyżej wartości 0.5 efektywność kompresji będzie spadać, choć oczywiście jest ona także zależna od prawdopodobieństw pozostałych symboli. Dla uzyskania granicznej efektywności przyjmijmy, że wszystkie pozostałe symbole tworzą poddrzewo o 100% efektywności zakodowania, którego korzeń jest połączony z symbolem x_n tworząc drzewo Huffmana. Sytuacja taka ma przykładowo miejsce dla $n - 1$ równoprawdopodobnych symboli (przy teoretycznym założeniu 100% efektywności zakodowania poddrzewa) oraz $(n - 1) \cdot p_1 \leq 0.5 \leq p_n$. Wówczas można wyznaczyć wartość progową, dla której nie można uzyskać kodu (binarnego lub mieszanego) o zadanej efektywności granicznej $\hat{E}_{gr}$. W tym celu należy rozwiązać nierówność:

$$\hat{E} = \frac{H(S)}{H(S) + R} \leq \frac{(1 - p_n) \cdot \log_2(n - 1) - p_n \cdot \log_2 p_n - (1 - p_n) \cdot \log_2 p_n}{(1 - p_n) \cdot \log_2(n - 1) + 1} < \hat{E}_{gr} \quad (4.5)$$

opartą między innymi na warunku:

$$R \geq 1 + p_n \cdot \log_r p_n + (1 - p_n) \cdot \log_r(1 - p_n) \quad \text{dla} \quad p_n \geq \frac{r}{r^2 + r - 1} \quad (4.6)$$

dotyczącym redundancji R dla r -narnych kodów Huffmana zaprezentowanym w [30], a także na podstawie wzoru entropii dla poddrzewa z pracy [12]. Przyjmijmy, że zadowala-

jącym progiem jest procentowa efektywność na poziomie $E = 95\%$. (czyli $\hat{E}_{gr} = 0.95$). Przykładowe wartości p_n , powyżej których uzyskanie takiego poziomu efektywności nie będzie możliwe przedstawiono w tabeli 4.5. Zatem propozycja użycia mieszanych kodów Huffmana nie rozwiązuje problemu niskiej efektywności kompresji dla $p_n > 0.5$. Stanowi to wadę w porównaniu np. z kodowaniem arytmetycznym nie posiadającym ograniczenia dolnego na średnią długość słowa kodowego, które może być ułamkową częścią bitu, co gwarantuje zachowanie wysokiej efektywności niezależnie od wartości p_n . Natomiast łatwo uzasadnić, że mieszany kod Huffmana ma dolne ograniczenie na średnią długość słowa kodu $L_{mix} = 1$, gdy np. źródło składa się z dwóch symboli.

Tabela 4.5 Wartości p_n , powyżej których efektywność kodu Huffmana jest mniejsza od 95%

Liczba $n - 1$ równoprawdopodobnych symboli	Wartość progowa p_n
2	0.6510
3	0.6617
4	0.6682
10	0.6860
20	0.6973
50	0.7104
100	0.7191
255	0.7297

4.5. Zastosowanie liczb pierwszych do konstrukcji buforów r -narnych

Tworząc kod mieszany powinniśmy dążyć do definiowania klasy kodów, która zagwarantuje nam odpowiednią efektywność E_{mix} jak najniższym kosztem, co jest ważne np. podczas sprzętowej realizacji kodera lub dekodera. Minimalizacja kosztu uzyskania z góry określonej efektywności dla znanego źródła S może być rozpatrywana na kilka sposobów:

- minimalizacja liczby m buforów r_i -narnych, która powoduje prostszą organizację pamięci buforowych i oznacza małą liczbę sprzętowych konwerterów r_i -narnych;
- minimalizacja rozmiaru bitowego poszczególnych buforów, czyli liczb B (prosty koder i dekodek);
- podjęcie decyzji, czy użycie kompresji liczb B kodem dwufazowym jest konieczne (wymaga to dodatkowo użycia kodera i dekodera kodu dwufazowego);

- ograniczenie od góry maksymalnego opóźnienia wprowadzanego przez koder/dekoder.

Każdą nieparzystą liczbę złożoną możemy przedstawić jako iloczyn liczb pierwszych. Pozwala to usunąć bufor r_i -nary będące liczbami złożonymi. Jeśli przeanalizujemy wartość $r = 15 = 3 \cdot 5$, to można ją rozłożyć na dwie wartości $r_1 = 3$ oraz $r_2 = 5$. Jeśli źródło wymaga użycia tylko wartości $r = 15$, to użyjemy tylko takiego bufora. Jednak jeśli źródło wymaga użycia wartości $r_i = \{3, 5, 15\}$, to możemy zastąpić kod cyfry piętnastowartościowej przez połączenie w mieszany symbol kodowy jednej cyfry ternarnej i jednej pięciowartościowej. Zakładamy, że kod dwufazowy nie jest używany. Jeżeli dla $r = 15$ użyjemy wartości $q = 11$, to wtedy liczba B składa się z $b(15^{11}) = 43$ bitów, a efektywność wynosi $E_{15(11)} = 99.94371\%$. Poszukując dla $r_1 = 3$ oraz $r_2 = 5$ zastępczych wzorcowych wartości q_1 i q_2 , musimy brać pod uwagę następującą zależność:

$$b(15^{11}) = b(3^{11} \cdot 5^{11}) \leq b(3^{11}) + b(5^{11})$$

Wówczas uzyskanie efektywności nie gorszej od $E_{15(11)}$ może wymagać doboru wyższych wartości wzorcowych q_1 i q_2 , niż wynika to z poniższych ograniczeń (nie stanowią więc one wystarczającego warunku uzyskania odpowiedniej efektywności):

$$3^5 \leq 3^{11} \leq 3^{17}$$

$$5^3 \leq 5^{11} \leq 5^{31}$$

Jeśli z tych zależności wybierzemy ograniczenia górne, czyli wzorcowe wartości $q_1 = 17$ oraz $q_2 = 31$, to otrzymamy średnią długość ścieżki złożonej z symbolu ternarnego i pięciowartościowego równą:

$$L_{3,5} = \bar{b}_{17}(3) + \bar{b}_{31}(5) = \frac{27}{17} + \frac{72}{31} = 3.91082 > \bar{b}_{11}(15) = \frac{43}{11} = 3.90909$$

Dopiero wzrost q_1 do kolejnej wartości wzorcowej $q_1 = 29$ pozwala uzyskać krótszą od zakładanej średnią długość kodu mieszanego:

$$L_{3,5} = \bar{b}_{29}(3) + \bar{b}_{31}(5) = \frac{46}{29} + \frac{72}{31} = 3.90879 < \bar{b}_{11}(15) = 3.90909$$

Daje to procentową efektywność takiego kodu równą $E_{3,5} = 99.95147\%$. Wiąże się to jednak z użyciem buforów o większych rozmiarach liczb B . Dla $r_1 = 3$ liczba B składa się z 46 bitów, a dla $r_2 = 5$ z 72 bitów, podczas gdy początkowo proponowany bufor dla $r = 15$ wymagał liczb B o rozmiarze jedynie 43 bitów. Zatem podejmowanie decyzji dotyczących zdefiniowania klasy kodów mieszanych nie jest sprawą łatwą i zależy od zastosowań.

Np. dla koderów i dekodeków projektowanych w języku wysokiego poziomu, działających na procesorach arytmetyczno-logicznych ogólnego przeznaczenia rozmiar liczb B nie stanowi dużego problemu (pod warunkiem, że zadbamy o odpowiednią prędkość pracy kodera i dekodera). Inaczej wygląda sytuacja, gdy interesuje nas zaprojektowanie sprzętowej realizacji algorytmu, który powinien cechować się jak największą prostotą działania (operacje na małych porcjach danych, małe rozmiary buforów itp.).

Rozpatrując wyłącznie przypadki dla idealnej klasy kodu mieszanego, możemy taką klasę opierać wyłącznie na wartościach r będących liczbami pierwszymi. Wynika to z prostej zależności, iż dowolna nieparzysta liczba złożona r może być przedstawiona jako iloczyn j liczb pierwszych $r = r_1 \cdot r_2 \cdot \dots \cdot r_j$. Na podstawie tych liczb pierwszych można utworzyć słowo kodu mieszanego składające się z j cyfr r_i -narych mające tę samą średnią długość słowa kodowego L_{mix} , co średnia L_r dla liczby złożonej, ponieważ:

$$L_{mix} = \log_2 r_1 + \log_2 r_2 + \dots + \log_2 r_j = \log_2 (r_1 \cdot r_2 \cdot \dots \cdot r_j) = \log_2 r = L_r \quad (4.7)$$

W przypadku pozostałych klas można się także ograniczyć do zbioru liczb pierwszych r_i , dzięki którym przy użyciu kodu mieszanego o odpowiednio dużych wartościach wzorcowych q_i możemy uzyskać efektywność E_{mix} nie gorszą, niż w przypadku użycia jako $r_i > 2$ nieparzystych liczb złożonych. Pozwala to na zmniejszenie złożoności obliczeniowej algorytmu poszukiwania optymalnego w danej klasie mieszanego kodu Huffmana. Szczegóły dotyczące tego algorytmu zostaną opisane w następnym rozdziale.

4.6. Testy efektywności

Dla klasy zdefiniowanej w tabeli 4.6 (bez użycia kodu dwufazowego), a także dla klasy idealnej dokonano pomiarów wpływu liczby n elementów źródła na procentowy udział binarnego kodu Huffmana we wszystkich testach, które polegały na znalezieniu optymalnego w danej klasie mieszanego kodu Huffmana. Każde badanie wykonano dla $2^{17} = 131072$ testowych źródeł o n symbolach. Rozkład prawdopodobieństwa dla testowych źródeł był generowany w sposób pseudolosowy (generator *Rand* języka C o w przybliżeniu jednostajnym rozkładzie). Rysunek 4.3 przedstawia posortowane średnie wartości prawdopodobieństw p_k symboli dla źródła o $n = 24$ symbolach (średnia na podstawie 131072 testów). Wartości p_k są zgodne z teoretyczną średnią wartością prawdopodobieństw dla takiego rozkładu liniowego:

$$p_k = \frac{2 \cdot k}{n \cdot (n + 1)}$$

(4.8)

Tabela 4.6

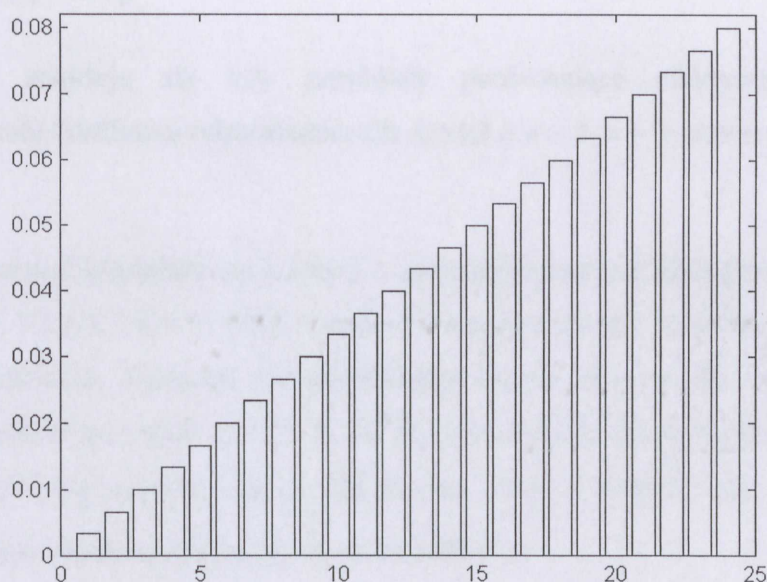
Testowa klasa kodów mieszanych

<i>r</i>	<i>q</i>	<i>b</i> (<i>r</i> ^{<i>q</i>})
2	1	1
3	41	65
5	31	72
7	16	45
11	13	45
13	17	63
17	11	45
19	4	17

Tabela 4.7

Wyniki testów kodów mieszanych

Liczba <i>n</i> symboli źródła	Dla klasy idealnej [%]	Dla klasy testowej [%]
3	82.615662	82.667542
4	47.846985	47.922516
5	39.065552	39.154816
6	25.956726	26.042938
7	13.333893	13.404083
8	5.573273	5.610651
9	2.482605	2.515411
10	1.639557	1.671600
11	1.264954	1.302338
12	0.832367	0.855255
13	0.453186	0.462341
14	0.193787	0.198364
15	0.062561	0.063324



Rysunek 4.3 Posortowane średnie wartości prawdopodobieństw wystąpienia symboli dla źródła o $n = 24$ symbolach

Procentowe wyniki testów dla $n \leq 15$, w których binarny kod Huffmana uzyskał najkrótszą średnią długość słowa kodowego L_{mix} wśród wszystkich kodów mieszanych przedstawia tabela 4.7. Na podstawie wyników widać, że wartości dla klasy kodów mieszanych z tabeli 4.6 różnią się w niewielkim stopniu od wyników zastosowania klasy idealnej. Uzasadnia to, iż ograniczenie klasy kodów do zbioru wartości r_i będących liczbami pierwszymi jest dobrą propozycją uproszczenia konstrukcji kodu mieszanego. Dla $n \geq 12$ przypadki uzasadnionego użycia binarnego kodu Huffmana (jako optymalnego kodu mieszanego) nie stanowią nawet jednego procenta źródeł testowych. Dla $n = 15$ na 131072 przeprowadzone testy kodów mieszanych tylko 83 (82 dla klasy idealnej) próby nie dały lepszego rezultatu niż sam binarny kod Huffmana. Dla wyższych wartości n pomiar nie jest już wystarczająco precyzyjny. Mimo zwiększenia liczby testów do 300000 dla $n = 20$ nie uzyskano ani jednego przypadku, w którym użycie kodu binarnego byłoby uzasadnione.

Na podstawie tych wyników możemy wywnioskować, że wzrost efektywności uzyskany dla kodu mieszanego (względem binarnego kodu Huffmana) jest zjawiskiem powszechnym zwłaszcza w sytuacjach praktycznych, w których liczba n symboli źródła często znacznie przekracza wartość 20.

4.7. Przykłady

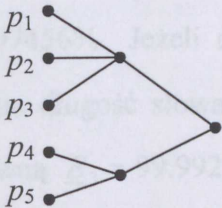
Poniżej znajdują się trzy przykłady porównujące efektywności binarnego i mieszanego kodu Huffmana odpowiednio dla źródeł o $n = 5$, $n = 7$ oraz $n = 20$ symbolach.

Przykład 4.4

Dane jest pięciosymbolowe źródło S o następującym rozkładzie prawdopodobieństw $P = \{0.13, 0.15, 0.17, 0.25, 0.3\}$ [68]. Tabela 4.8 zawiera zestaw słów binarnego i mieszanego kodu Huffmana. Rysunek 4.4 przedstawia kształt drzewa dla mieszanego kodu Huffmana o skrótowym opisie $r_w = \{3, 2, 2\}$. Średnia długość słowa kodu binarnego wynosi $L_2 = 2.28$, co przy entropii tego źródła równej $H(S) = 2.24887$ bita na symbol daje procentową efektywność na poziomie $E_2 = 98.63445\%$.

Tabela 4.8 Kody Huffmana z przykładu 4.4

p_i	Kod mieszany	Kod binarny
$p_1 = 0.13$	$0_2 0_3$	100
$p_2 = 0.15$	$0_2 1_3$	101
$p_3 = 0.17$	$0_2 2_3$	00
$p_4 = 0.25$	$1_2 0_2$	01
$p_5 = 0.30$	$1_2 1_2$	11



Rysunek 4.4 Drzewo kodu binarno-ternarnego z przykładu 4.4

Jeżeli oprócz kodu binarnego użyjemy także kodu ternarnego ($r_1 = 3$), to otrzymamy średnią długość dla kodu binarno-ternarnego równą:

$$L_{mix} = L_{2,3} = \frac{100 + 55}{100} + \frac{45}{100} \cdot \bar{b}_q(3)$$

Dla $q_1 = 5$ daje to $L_{mix} = 2.27$, co oznacza efektywność $E_{mix} = 99.06897\%$. Zwiększenie długości bufora cyfr ternarnych do $q_1 = 41$ pozwala uzyskać średnią $L_{mix} = 2.26342$, co daje efektywność $E_{mix} = 99.35721\%$. Waga zysku wynosi $W = 53.27010\%$ (patrz wzór (1.9)). Jeżeli dodatkowo użyjemy kodu dwufazowego do kompresji liczb B , to średnia długość słowa kodowego zmniejszy się do $\underline{L}_{mix} = 2.26329$ i uzyskamy efektywność równą

$\underline{E}_{mix} = 99.36276\%$. Dalszy wzrost efektywności mieszanego kodu Huffmana jest możliwy jedynie do wartości granicznej $E_{mix} = 99.36517\%$ wyznaczonej przez klasę idealną.

Przykład 4.5

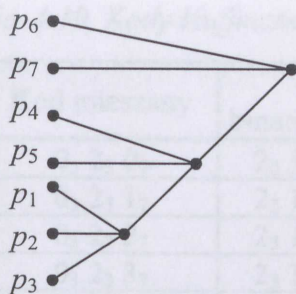
Dane jest siedmiosymbolowe źródło S o rozkładzie przedstawionym w tabeli 4.9, w której znajdują się także zestawy słów binarnego i mieszanego kodu Huffmana. Rysunek 4.5 przedstawia kształt drzewa dla mieszanego kodu Huffmana o skrótowym opisie $r_w = \{3, 3, 3\}$. Jak wynika z rysunku 4.5, jest to kod wyłącznie ternarny (ze względu na specyficzne wartości rozkładu źródła) [2, 83]. Średnia długość słowa kodu binarnego wynosi $L_2 = \frac{65}{27} = 2.40741$, co przy entropii tego źródła równej $H(S) = 2.28939$ bita na symbol daje procentową efektywność na poziomie $E_2 = 95.09775\%$. Jeżeli zamiast kodu binarnego użyjemy kodu ternarnego ($r_1 = 3$), to otrzymamy średnią długość kodu równą:

$$L_3 = \left(1 + \frac{1}{3} + \frac{1}{9}\right) \cdot \bar{b}_q(3) = \frac{13}{9} \cdot \bar{b}_q(3)$$

Dla $q_1 = 5$ daje to $L_3 = 2.31313$, co oznacza efektywność $E_3 = 99.06156\%$. Zwiększenie liczby cyfr ternarnych w liczbie B do $q_1 = 41$ pozwala uzyskać średnią $L_3 = 2.28997$, co daje efektywność $E_3 = 99.97456\%$. Jeżeli dodatkowo użyjemy kodu dwufazowego do kompresji liczb B , to średnia długość słowa kodowego zmniejszy się do $\underline{L}_3 = 2.28957$ i uzyskamy efektywność równą $\underline{E}_3 = 99.99229\%$. Kod ten cechuje wysoka waga zysku $W = 99.85048\%$. Rozpatrując klasę idealną dla kodu ternarnego uzyskamy teoretyczną efektywność $E_3 = 100\%$.

Tabela 4.9 Kody Huffmana z przykładu 4.5

p_i	Kod ternarny	Kod binarny
$p_1 = \frac{1}{27}$	220	0010
$p_2 = \frac{1}{27}$	221	0011
$p_3 = \frac{1}{27}$	222	000
$p_4 = \frac{1}{9}$	20	010
$p_5 = \frac{1}{9}$	21	011
$p_6 = \frac{1}{3}$	0	10
$p_7 = \frac{1}{3}$	1	11



Rysunek 4.5 Drzewo kodu mieszane (ternarnego) z przykładu 4.5

Przykład 4.6

Dane jest źródło S składające się z $n = 20$ symboli o rozkładzie przedstawionym w tabeli 4.10, w której znajdują się także zestawy słów binarnego, binarno-ternarnego i mieszane kodu Huffmana. Rysunek 4.6 przedstawia kształt drzewa dla mieszane kodu Huffmana o skrótowym opisie $r_w = \{7, 5, 3, 3, 3, 2, 3\}$.

Średnia długość słowa kodu binarnego wynosi $L_2 = 3.762$, co przy entropii tego źródła równej $H(S) = 3.72255$ bita na symbol daje procentową efektywność na poziomie $E_2 = 98.95126\%$. Jeżeli zamiast kodu binarnego użyjemy kodu mieszane z daną klasą: $r_0 = 2, r_1 = 3, r_2 = 5, r_3 = 7, q_1 = 41, q_2 = 31, q_3 = 16$, to zgodnie ze wzorem (4.4) otrzymamy średnią długość kodu mieszane równą:

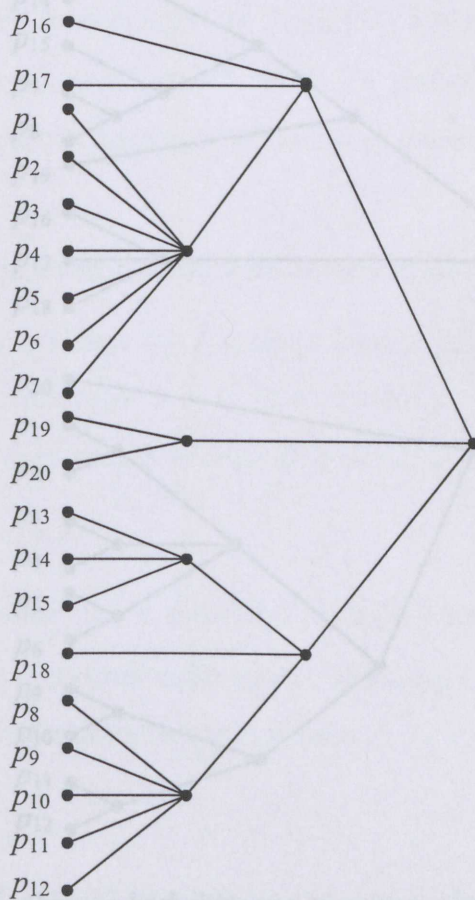
$$L_{mix} = \frac{333}{1000} \cdot 1 + \frac{1000 + 332 + 335 + 111}{1000} \cdot \bar{b}_q(3) + \frac{112}{1000} \cdot \bar{b}_q(5) + \frac{111}{1000} \cdot \bar{b}_q(7)$$

$$L_{mix} = 0.333 + 1.778 \cdot \bar{b}_q(3) + 0.112 \cdot \bar{b}_q(5) + 0.111 \cdot \bar{b}_q(7)$$

Dla klasy zdefiniowanej w przykładzie daje to średnią $L_{mix} = L_{2,3,5,7} = 3.72410$, co odpowiada efektywności $E_{mix} = 99.95836\%$ (waga zysku $W = 96.06975\%$). Jeżeli dodatkowo użyjemy kodu dwufazowego do kompresji liczb B , to średnia długość słowa kodowego zmniejszy się do $\underline{L}_{mix} = 3.72314$ i uzyskamy wówczas efektywność równą $\underline{E}_{mix} = 99.98409\%$. W przypadku klasy idealnej dla kodu mieszane uzyskamy średnią długość słowa kodowego równą $L_{mix} = 3.72274$ oraz teoretyczną efektywność $E_{mix} = 99.99492\%$. Prawdopodobieństwa $p(r_i)$ występowania cyfr r_i -narych wynoszą odpowiednio $p(2) = 0.14267, p(3) = 0.76178, p(5) = 0.04799$ oraz $p(7) = 0.04756$.

Tabela 4.10 Kody Huffmana z przykładu 4.6

p_i	Kod mieszany	Kod binarno-ternarny	Kod binarny
$p_1 = 0.015$	$0_3 2_3 0_7$	$2_3 1_2 0_2 0_3 0_2$	001100
$p_2 = 0.015$	$0_3 2_3 1_7$	$2_3 1_2 0_2 0_3 1_2$	001101
$p_3 = 0.016$	$0_3 2_3 2_7$	$2_3 1_2 0_2 1_3 0_2$	001110
$p_4 = 0.016$	$0_3 2_3 3_7$	$2_3 1_2 0_2 1_3 1_2$	001111
$p_5 = 0.016$	$0_3 2_3 4_7$	$2_3 1_2 0_2 2_3 0_2$	101000
$p_6 = 0.016$	$0_3 2_3 5_7$	$2_3 1_2 0_2 3_3 1_2$	101001
$p_7 = 0.017$	$0_3 2_3 6_7$	$0_3 0_2 1_2 1_2 0_2$	000000
$p_8 = 0.022$	$2_3 2_3 0_5$	$0_3 0_2 1_2 1_2 1_2$	000001
$p_9 = 0.022$	$2_3 2_3 1_5$	$2_3 1_2 1_2 0_2 0_2$	000010
$p_{10} = 0.022$	$2_3 2_3 2_5$	$2_3 1_2 1_2 0_2 1_2$	000011
$p_{11} = 0.023$	$2_3 2_3 3_5$	$2_3 1_2 1_2 1_2 0_2$	001000
$p_{12} = 0.023$	$2_3 2_3 4_5$	$2_3 1_2 1_2 1_2 1_2$	001001
$p_{13} = 0.037$	$2_3 0_3 0_3$	$0_3 0_2 0_2 0_2$	101010
$p_{14} = 0.037$	$2_3 0_3 1_3$	$0_3 0_2 0_2 1_2$	101010
$p_{15} = 0.037$	$2_3 0_3 2_3$	$0_3 0_2 1_2 0_2$	101011
$p_{16} = 0.110$	$0_3 0_3$	$1_3 0_3$	010
$p_{17} = 0.111$	$0_3 1_3$	$1_3 1_3$	011
$p_{18} = 0.112$	$2_3 1_3$	$1_3 2_3$	100
$p_{19} = 0.166$	$1_3 0_2$	$0_3 1_2$	110
$p_{20} = 0.167$	$1_3 1_2$	$2_3 0_2$	111

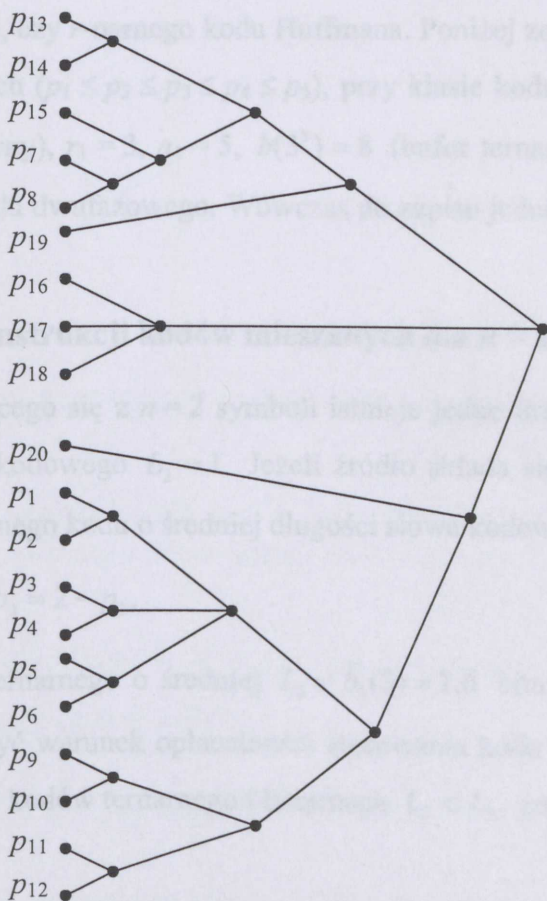


Rysunek 4.6 Drzewo mieszanego kodu Huffmana z przykładu 4.6

Ograniczenie kodu mieszanego do binarno-ternarnego kodu Huffmana z daną klasą: $r_0 = 2$, $r_1 = 3$, $q_1 = 41$ spowoduje tylko nieznaczny wzrost średniej długości kodu mieszanego równej:

$$L_{2,3} = 1.464 + 1.427 \cdot \bar{b}_q(3)$$

Dla zdefiniowanej wyżej klasy daje to średnią $L_{2,3} = 3.72632$, co odpowiada efektywności $E_{2,3} = 99.89881\%$. Jeżeli dodatkowo użyjemy kodu dwufazowego do kompresji liczb B , to średnia długość słowa kodowego zmniejszy się do $\underline{L}_{2,3} = 3.72592$ i uzyskamy wówczas efektywność równą $\underline{E}_{2,3} = 99.90957\%$. W przypadku klasy idealnej dla kodu binarno-ternarnego uzyskamy średnią długość słowa kodowego równą $L_{2,3} = 3.72574$ oraz teoretyczną efektywność $E_{2,3} = 99.91424\%$. Rysunek 4.7 przedstawia kształt drzewa dla binarno-ternarnego kodu Huffmana o skrótowym opisie $r_w = \{2, 2, 2, 2, 2, 2, 3, 2, 2, 2, 2, 2, 3, 2, 2, 3\}$. Prawdopodobieństwa $p(r_i)$ występowania cyfr r_i -narnych dla kodu binarno-ternarnego są zrównoważone i wynoszą odpowiednio $p(2) = 0.50640$, $p(3) = 0.49360$.



Rysunek 4.7 Drzewo kodu binarno-ternarnego z przykładu 4.6

5. Konstrukcja optymalnych mieszanych kodów Huffmana

Rozdział ten rozpoczyna analiza konstrukcji optymalnych mieszanych kodów Huffmana na podstawie analizy źródeł o małej liczbie symboli ($n \leq 5$) wraz z podaniem warunków opłacalności zastosowania kodu mieszanego. Algorytm uzyskania optymalnego w danej klasie mieszanego kodu Huffmana *metodą porównań* ma wykładniczą złożoność obliczeniową w funkcji liczby symboli źródła (co zostanie uzasadnione w punkcie 5.2). W dwóch kolejnych punktach zostaną przedstawione, wraz z przykładami, dwie metody zmniejszenia złożoności obliczeniowej poszukiwania optymalnego kodu Huffmana, które polegają na tworzeniu integralnych poddrzew rozpatrywanych jako mniejsze źródła o $k < n$ symbolach.

5.1. Podstawy

Przedstawiona w tym punkcie analiza cech źródeł o małej liczbie symboli pokazuje, że znalezienie dla źródła S optymalnego w danej klasie kodu mieszanego nie jest tak proste jak w przypadku binarnego, czy r -narnego kodu Huffmana. Poniżej zostaną przeanalizowane źródła o $n \leq 5$ symbolach ($p_1 \leq p_2 \leq p_3 \leq p_4 \leq p_5$), przy klasie kodu mieszanego o parametrach $r_0 = 2$ (bufor binarny), $r_1 = 3$, $q_1 = 5$, $b(3^5) = 8$ (bufor ternarny z ośmiobitowymi liczbami B), bez użycia kodu dwufazowego. Wówczas do zapisu jednej cyfry ternarnej wystarczy 1.6 bita.

5.1.1. Analiza konstrukcji kodów mieszanych dla $n = 2$ oraz $n = 3$

Dla źródła składającego się z $n = 2$ symboli istnieje jedno drzewo binarnego kodu o średniej długości słowa kodowego $L_2 = 1$. Jeżeli źródło składa się z $n = 3$ symboli, to istnieje jedno drzewo binarnego kodu o średniej długości słowa kodowego równej:

$$L_2 = 2 \cdot (p_1 + p_2) + p_3 = 2 - p_3,$$

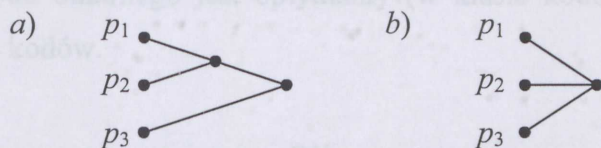
oraz jedno drzewo kodu ternarnego o średniej $L_3 = \bar{b}_5(3) = 1.6$ bita (patrz rysunek 5.1 *a* oraz *b*). Możemy wyznaczyć warunek opłacalności stosowania kodu ternarnego przez porównanie średnich długości kodów ternarnego i binarnego $L_3 < L_2$, co daje:

$$p_1 + p_2 > 1.5 \cdot p_3 \tag{5.1}$$

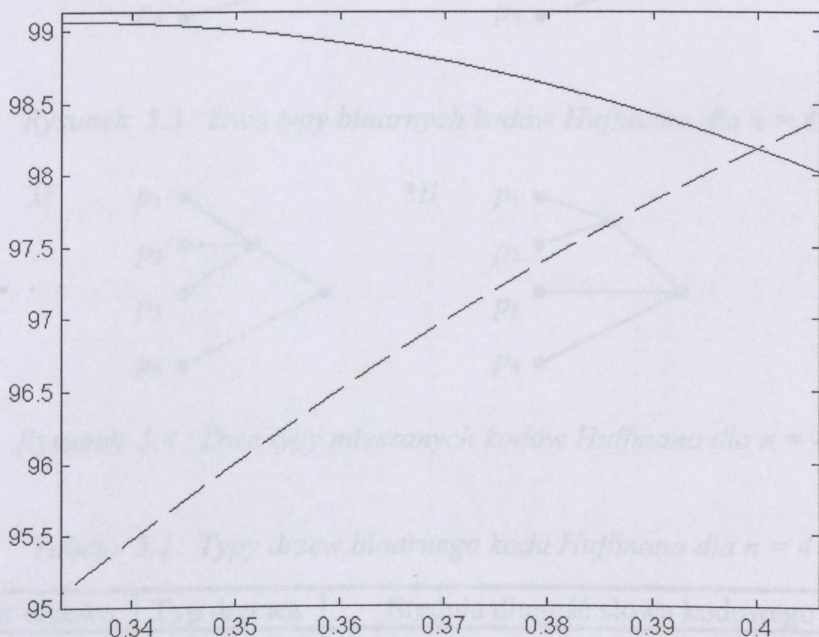
Oznacza to, że $p_3 < 0.4$. Jednocześnie możemy oszacować dolną granicę dla najbardziej prawdopodobnego symbolu x_n jako:

$$p_n \geq \frac{1}{n} \quad (5.2)$$

Czyli dla trójsymbolowego źródła przy $q_1 = 5$ użycie kodu mieszanego (ternarnego) jest opłacalne w przedziale $\frac{1}{3} \leq p_3 < 0.4$.



Rysunek 5.1 Binarny(a) i ternarny(b) kod Huffmana dla $n = 3$



Rysunek 5.2 Procentowa efektywność binarnego (linia przerywana) i ternarnego (linia ciągła) kodu Huffmana w funkcji p_3

Jeśli przyjmujemy najprostszą sytuację, że $p_1 = p_2$, to funkcję entropii takiego źródła można przedstawić w postaci funkcji jednej zmiennej p_3 :

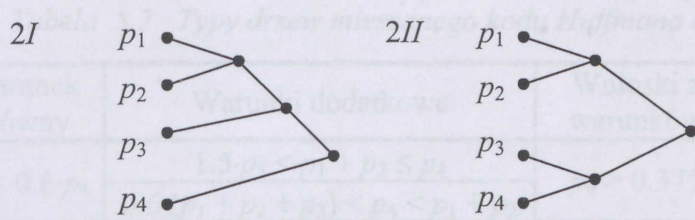
$$H(S) = -p_3 \cdot \log_2 p_3 - (1 - p_3) \cdot \log_2 \frac{1 - p_3}{2}$$

Wówczas można porównać procentowe efektywności dla kodu binarnego i ternarnego w funkcji p_3 najbardziej prawdopodobnego symbolu x_3 . Wykres takiej zależności przedsta-

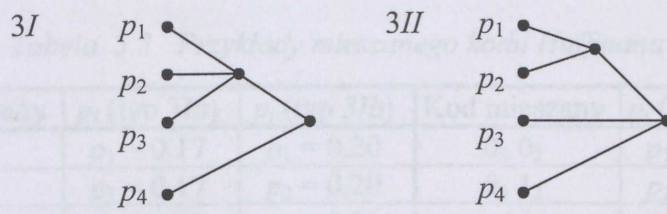
wia rysunek 5.2, gdzie linia ciągła oznacza procentową efektywność kodu ternarnego, natomiast linia przerywana procentową efektywność kodu binarnego.

5.1.2. Analiza konstrukcji kodów mieszanych dla $n = 4$

Gdy źródło składa się z $n = 4$ symboli, to wówczas istnieją dwa typy drzew binarnych (2I oraz 2II) przedstawionych na rysunku 5.3. Tabela 5.1 zawiera warunki, dla których dany typ kodu binarnego jest optymalny (w klasie kodów binarnych), a także średnie długości tych kodów.



Rysunek 5.3 Dwa typy binarnych kodów Huffmana dla $n = 4$



Rysunek 5.4 Dwa typy mieszanych kodów Huffmana dla $n = 4$

Tabela 5.1 Typy drzew binarnego kodu Huffmana dla $n = 4$

Nr drzewa	Typ drzewa	Średnia długość słowa kodowego
2I	$p_1 + p_2 \leq p_4$	$L_{2I} = 3 \cdot (p_1 + p_2) + 2 \cdot p_3 + p_4 = 3 - p_3 - 2 \cdot p_4$
2II	$p_1 + p_2 > p_4$	$L_{2II} = 2 \cdot (p_1 + p_2 + p_3 + p_4) = 2$

Dla $n = 4$ istnieje jedno drzewo mieszane (typ 3I) łączące trzy najmniej prawdopodobne symbole przy użyciu kodu ternarnego, dla którego średnia wynosi:

$$L_{mix} = L_{3I} = 2.6 \cdot (p_1 + p_2 + p_3) + p_4 = 2.6 - 1.6 \cdot p_4,$$

a także jedno drzewo mieszane (typ 3II) łączące dwa najmniej prawdopodobne symbole w jeden węzeł w_{12} , który jest następnie łączony z dwoma kolejnymi symbolami przy użyciu kodu ternarnego (patrz rysunek 5.4). Dla tego drzewa średnia długość słowa kodowego wynosi:

$$L_{mix} = L_{3II} = 1.6 + p_1 + p_2$$

Tabela 5.2 określa warunki opłacalności użycia kodu mieszanego dla $n = 4$, które dzielą wszystkie kody binarno-ternarne na cztery typy. Pierwszy warunek główny dotyczy drzew typu 3Ia oraz 3Ib (różnią się warunkami dodatkowymi). Natomiast drugi warunek główny dotyczy drzew typu 3IIa oraz 3IIb (także różnią się warunkami dodatkowymi). Przykłady dla tych czterech typów warunków znajdują się w tabeli 5.3, która została opublikowana przez autora w pracy [78].

Tabela 5.2 Typy drzew mieszanego kodu Huffmana dla $n = 4$

Nr drzewa	Warunek główny	Warunki dodatkowe	Wnioski z warunków	Średnia długość słowa kodowego
3Ia	$p_3 < 0.6 \cdot p_4$	$1.5 \cdot p_3 < p_1 + p_2 \leq p_4$	$p_4 > 0.375$	$L_{3I} = 2.6 - 1.6 \cdot p_4$
3Ib		$0.6 \cdot (p_1 + p_2 + p_3) < p_4 < p_1 + p_2$		
3IIa	$p_3 \geq 0.6 \cdot p_4$	$p_1 + p_2 \leq p_4 < 0.4$	$p_4 < 0.4$ oraz $p_1 + p_2 < 0.4$	$L_{3II} = 1.6 + p_1 + p_2$
3IIb		$p_4 < p_1 + p_2 < 0.4$		

Tabela 5.3 Przykłady mieszanego kodu Huffmana dla $n = 4$

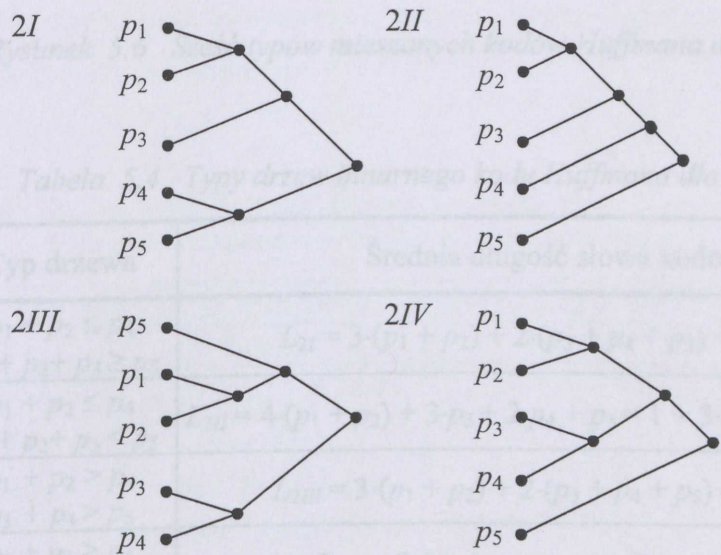
Kod mieszany	p_i (typ 3Ia)	p_i (typ 3Ib)	Kod mieszany	p_i (typ 3IIa)	p_i (typ 3IIb)
0 ₂ 0 ₃	$p_1 = 0.17$	$p_1 = 0.20$	0 ₃ 0 ₂	$p_1 = 0.17$	$p_1 = 0.19$
0 ₂ 1 ₃	$p_2 = 0.17$	$p_2 = 0.20$	0 ₃ 1 ₂	$p_2 = 0.20$	$p_2 = 0.20$
0 ₂ 2 ₃	$p_3 = 0.17$	$p_3 = 0.21$	1 ₃	$p_3 = 0.24$	$p_3 = 0.24$
1 ₂	$p_4 = 0.49$	$p_4 = 0.39$	2 ₃	$p_4 = 0.39$	$p_4 = 0.37$
$L_{mix} = 2.6 - 1.6 \cdot p_4$	$L_2 = 1.85$ $L_{mix} = 1.816$	$L_2 = 2.00$ $L_{mix} = 1.976$	$L_{mix} = 1.6 + p_1 + p_2$	$L_2 = 1.98$ $L_{mix} = 1.97$	$L_2 = 2.00$ $L_{mix} = 1.99$

5.1.3. Analiza konstrukcji kodów mieszanych dla $n = 5$

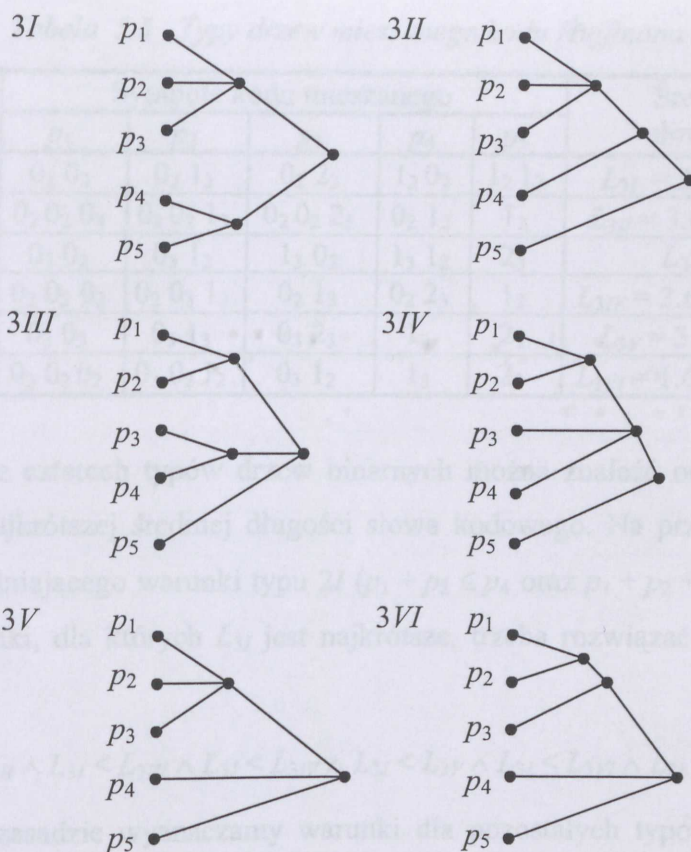
Ze wzrostem liczby n symboli źródła pojawia się problem odnajdywania kodu o minimalnej wartości L_{mix} oraz warunków opłacalności stosowania kodów mieszanych, gdyż należy przeanalizować coraz więcej warunków, aby wyprowadzić zależności dla opłacalności użycia konkretnego drzewa kodu mieszanego względem każdego z typów drzew binarnych. Ponadto pojawia się problem, jak w prosty sposób tworzyć optymalne drzewa mieszane. Jeżeli rozbudujemy klasyczny algorytm binarnego kodu Huffmana o istnienie symboli ternarnych, to możemy stworzyć propozycję następującego *algorytmu konstrukcji optymalnego kodu binarno-ternarnego*:

Połącz trzy najmniej prawdopodobne symbole. Jeśli kod ternarny o średniej długości $L_3 = \bar{b}_q(3)$ jest krótszy od średniej długości L_2 dla kodu binarnego powstałego z połączenia tych trzech symboli, to łączymy te trzy symbole w nowy węzeł w_{123} o trzech potomkach używając kodu ternarnego. Usuwamy z listy te trzy symbole zastępując je węzłem w_{123} i po posortowaniu prawdopodobieństw rozpoczynamy algorytm od początku.

Możemy uzyskać wzrost efektywności, ale niestety *nie jest to właściwe podejście* do uzyskania optymalnego kodu, ponieważ kolejne symbole o wyższych prawdopodobieństwach także mają wpływ na podjęcie decyzji o tym, czy trzy najmniej prawdopodobne symbole połączyć używając kodu ternarnego, czy też połączyć jedynie dwa najmniej prawdopodobne symbole, a dopiero dla kolejnych symboli użyć kodu ternarnego. Można to zaobserwować już dla źródła składającego się z $n > 4$ symboli. Rozważmy zatem trudniejszą sytuację, w której mamy źródło o $n = 5$ symbolach [76]. Istnieją wówczas cztery typy binarnych drzew Huffmana (2I, 2II, 2III, 2IV – patrz rysunek 5.5) [19], których cechy przedstawia tabela 5.4, a także sześć typów drzew mieszanych (3I, 3II, 3III, 3IV, 3V, 3VI – patrz rysunek 5.6), których cechy zostały przedstawione w tabeli 5.5.



Rysunek 5.5 Cztery typy binarnych kodów Huffmana dla $n = 5$



Rysunek 5.6 Sześć typów mieszanych kodów Huffmana dla $n = 5$

Tabela 5.4 Typy drzew binarnego kodu Huffmana dla $n = 5$

Nr drzewa	Typ drzewa	Średnia długość słowa kodowego
2I	$p_1 + p_2 \leq p_4$ $p_1 + p_2 + p_3 \geq p_5$	$L_{2I} = 3 \cdot (p_1 + p_2) + 2 \cdot (p_3 + p_4 + p_5) = 2 + p_1 + p_2$
2II	$p_1 + p_2 \leq p_4$ $p_1 + p_2 + p_3 < p_5$	$L_{2II} = 4 \cdot (p_1 + p_2) + 3 \cdot p_3 + 2 \cdot p_4 + p_5 = 1 + 3 \cdot (p_1 + p_2) + 2 \cdot p_3 + p_4$
2III	$p_1 + p_2 > p_4$ $p_3 + p_4 > p_5$	$L_{2III} = 3 \cdot (p_1 + p_2) + 2 \cdot (p_3 + p_4 + p_5) = 2 + p_1 + p_2$
2IV	$p_1 + p_2 > p_4$ $p_3 + p_4 \leq p_5$	$L_{2IV} = 3 \cdot (p_1 + p_2 + p_3 + p_4) + p_5 = 3 - 2 \cdot p_5$

Tabela 5.5 Typy drzew mieszanego kodu Huffmana dla $n = 5$

Nr drzewa	Symbole kodu mieszanego					Średnia długość słowa kodowego
	p_1	p_2	p_3	p_4	p_5	
3I	0 ₂ 0 ₃	0 ₂ 1 ₃	0 ₂ 2 ₃	1 ₂ 0 ₂	1 ₂ 1 ₂	$L_{3I} = 2.6 - 0.6 \cdot (p_4 + p_5)$
3II	0 ₂ 0 ₂ 0 ₃	0 ₂ 0 ₂ 1 ₃	0 ₂ 0 ₂ 2 ₃	0 ₂ 1 ₂	1 ₂	$L_{3II} = 3.6 - 1.6 \cdot p_4 - 2.6 \cdot p_5$
3III	0 ₃ 0 ₂	0 ₃ 1 ₂	1 ₃ 0 ₂	1 ₃ 1 ₂	2 ₃	$L_{3III} = 2.6 - p_5$
3IV	0 ₂ 0 ₃ 0 ₂	0 ₂ 0 ₃ 1 ₂	0 ₂ 1 ₃	0 ₂ 2 ₃	1 ₂	$L_{3IV} = 2.6 + p_1 + p_2 - 1.6 \cdot p_5$
3V	0 ₃ 0 ₃	0 ₃ 1 ₃	0 ₃ 2 ₃	1 ₃	2 ₃	$L_{3V} = 3.2 - 1.6 \cdot (p_4 + p_5)$
3VI	0 ₃ 0 ₂ 0 ₂	0 ₃ 0 ₂ 1 ₂	0 ₃ 1 ₂	1 ₃	2 ₃	$L_{3VI} = 1.6 + 2 \cdot (p_1 + p_2) + p_3$

Dla każdego z czterech typów drzew binarnych można znaleźć odpowiadające im kody mieszane o najkrótszej średniej długości słowa kodowego. Na przykład aby dla drzewa binarnego spełniającego warunki typu 2I ($p_1 + p_2 \leq p_4$ oraz $p_1 + p_2 + p_3 \geq p_5$) dobrać odpowiednie warunki, dla których L_{3I} jest najkrótsze, trzeba rozwiązać zestaw następujących nierówności:

$$L_{3I} < L_{3II} \wedge L_{3I} < L_{3III} \wedge L_{3I} < L_{3IV} \wedge L_{3I} < L_{3V} \wedge L_{3I} < L_{3VI} \wedge L_{3I} < L_{2I}$$

Na podobnej zasadzie wyznaczamy warunki dla pozostałych typów drzew mieszanych. Niektóre układy nierówności są sprzeczne, więc nie zawsze można znaleźć takie warunki, dla których kod typu 3x jest najlepszy.

Tabela 5.6 Zestaw warunków dla drzewa binarnego typu 2I, przy których dany typ drzewa mieszanego daje najkrótszą średnią długość słowa kodowego

Dla spełnionych warunków $p_1 + p_2 \leq p_4$ oraz $p_1 + p_2 + p_3 \geq p_5$			
Typ 3I	Typ 3IV	Typ 3V	Typ 3VI
$p_4 + 2 \cdot p_5 < 1$ $p_4 > \frac{2}{3} \cdot p_5$ $p_3 + 0.4 \cdot p_4 + 2 \cdot p_5 < 1$ $p_4 + p_5 < 0.6$ $p_1 + p_2 + \frac{2}{7} \cdot p_3 > \frac{2}{7}$ $p_1 + p_2 > 1.5 \cdot p_3$	$p_3 + 0.4 \cdot p_4 + 2 \cdot p_5 > 1$ $p_3 > 0.6 \cdot p_4$ $p_1 + p_2 < 0.6 \cdot p_5$ $p_1 + p_2 + 1.6 \cdot p_4 < 0.6$ $p_4 < 0.6 \cdot p_5$ $p_5 > 0.375$	$p_4 + p_5 > 0.6$ $p_5 < 0.4$ $\frac{8}{3} \cdot p_4 + p_5 > 1$ $p_1 + p_2 + 1.6 \cdot p_4 > 0.6$ $p_1 + p_2 > 1.5 \cdot p_3$ $p_1 + p_2 + \frac{8}{3} \cdot p_3 < \frac{2}{3}$	$p_1 + p_2 + \frac{2}{7} \cdot p_3 < \frac{2}{7}$ $p_3 + 0.4 \cdot p_4 > 0.6 \cdot p_5$ $p_1 + p_2 < p_4$ $p_4 > 0.6 \cdot p_5$ $p_1 + p_2 < 1.5 \cdot p_3$ $p_1 + p_2 + p_3 < 0.4$

Tabela 5.6 przedstawia zestaw warunków dla drzewa binarnego typu 2I, przy których dany typ drzewa mieszanego daje najkrótszą średnią długość słowa kodowego, natomiast w tabeli 5.7 znajdują się przykłady mieszanego kodu Huffmana spełniające warunki typu 2I. Następne pary tabel odpowiadają kolejno typowi 2II, 2III oraz 2IV.

Tabela 5.7 Przykłady mieszanego kodu Huffmana dla $n = 5$ spełniające warunki 2I

Typ 3I	Typ 3IV	Typ 3V	Typ 3VI
$p_1 = 0.14$	$p_1 = 0.1$	$p_1 = 0.13$	$p_1 = 0.11$
$p_2 = 0.14$	$p_2 = 0.1$	$p_2 = 0.13$	$p_2 = 0.11$
$p_3 = 0.14$	$p_3 = 0.2$	$p_3 = 0.13$	$p_3 = 0.16$
$p_4 = 0.28$	$p_4 = 0.2$	$p_4 = 0.28$	$p_4 = 0.26$
$p_5 = 0.30$	$p_5 = 0.4$	$p_5 = 0.33$	$p_5 = 0.36$
$L_2 = 2.28$	$L_2 = 2.20$	$L_2 = 2.26$	$L_2 = 2.22$
$L_{mix} = 2.252$	$L_{mix} = 2.16$	$L_{mix} = 2.224$	$L_{mix} = 2.20$

Tabela 5.8 Zestaw warunków dla drzewa binarnego typu 2II, przy których dany typ drzewa mieszanego daje najkrótszą średnią długość słowa kodowego

Dla spełnionych warunków $p_1 + p_2 \leq p_4$ oraz $p_1 + p_2 + p_3 < p_5$			
Typ 3II	Typ 3IV	Typ 3V	Typ 3VI
$p_4 + 2 \cdot p_5 > 1$ $p_4 + p_5 > 0.625$ $p_3 < 0.6 \cdot p_4$ $p_5 > 0.4$ $p_3 + 0.4 \cdot p_4 < 0.6 \cdot p_5$ $p_1 + p_2 > 1.5 \cdot p_3$	$p_3 + 0.4 \cdot p_4 + 2 \cdot p_5 > 1$ $p_3 > 0.6 \cdot p_4$ $p_1 + p_2 < 0.6 \cdot p_5$ $p_1 + p_2 + 1.6 \cdot p_4 < 0.6$ $p_4 < 0.6 \cdot p_5$ $2.5 \cdot p_4 + p_5 < 1$	$p_4 + p_5 > 0.6$ $p_5 < 0.4$ $\frac{8}{3} \cdot p_4 + p_5 > 1$ $p_1 + p_2 + 1.6 \cdot p_4 > 0.6$ $p_1 + p_2 > 1.5 \cdot p_3$ $p_3 + 0.4 \cdot p_4 + 1.4 \cdot p_5 < 0.8$	$p_1 + p_2 + \frac{2}{7} \cdot p_3 < \frac{2}{7}$ $p_3 + 0.4 \cdot p_4 > 0.6 \cdot p_5$ $p_1 + p_2 < p_4$ $p_4 > 0.6 \cdot p_5$ $p_1 + p_2 < 1.5 \cdot p_3$ $p_5 < 0.4$

Tabela 5.9 Przykłady mieszanego kodu Huffmana dla $n = 5$ spełniające warunki 2II

Typ 3II	Typ 3IV	Typ 3V	Typ 3VI
$p_1 = 0.1$	$p_1 = 0.10$	$p_1 = 0.12$	$p_1 = 0.05$
$p_2 = 0.1$	$p_2 = 0.10$	$p_2 = 0.12$	$p_2 = 0.05$
$p_3 = 0.1$	$p_3 = 0.15$	$p_3 = 0.12$	$p_3 = 0.20$
$p_4 = 0.2$	$p_4 = 0.20$	$p_4 = 0.25$	$p_4 = 0.35$
$p_5 = 0.5$	$p_5 = 0.45$	$p_5 = 0.39$	$p_5 = 0.35$
$L_2 = 2.00$	$L_2 = 2.10$	$L_2 = 2.21$	$L_2 = 2.05$
$L_{mix} = 1.98$	$L_{mix} = 2.08$	$L_{mix} = 2.176$	$L_{mix} = 2.00$

Tabela 5.10 Zestaw warunków dla drzewa binarnego typu 2III, przy których dany typ drzewa mieszanego daje najkrótszą średnią długość słowa kodowego

Dla spełnionych warunków $p_1 + p_2 > p_4$ oraz $p_3 + p_4 > p_5$			
Typ 3I	Typ 3III	Typ 3IV	Typ 3V
$p_4 + 2 \cdot p_5 < 1$ $p_4 > \frac{2}{3} \cdot p_5$ $p_3 + 0.4 \cdot p_4 + 2 \cdot p_5 < 1$ $p_4 + p_5 < 0.6$ $p_1 + p_2 + \frac{2}{7} \cdot p_3 > \frac{2}{7}$ $p_1 + p_2 > 1.5 \cdot p_3$	$p_4 < \frac{2}{3} \cdot p_5$ $p_4 + p_5 < 0.625$ $p_1 + p_2 > 0.6 \cdot p_5$ $\frac{8}{3} \cdot p_4 + p_5 < 1$ $p_1 + p_2 + p_5 > 0.6$	$p_3 + 0.4 \cdot p_4 + 2 \cdot p_5 > 1$ $p_3 > 0.6 \cdot p_4$ $p_1 + p_2 < 0.6 \cdot p_5$ $p_1 + p_2 + 1.6 \cdot p_4 < 0.6$ $p_4 < 0.6 \cdot p_5$ $p_5 > 0.375$	$p_4 + p_5 > 0.6$ $p_5 < 0.4$ $\frac{8}{3} \cdot p_4 + p_5 > 1$ $p_1 + p_2 + 1.6 \cdot p_4 > 0.6$ $p_1 + p_2 > 1.5 \cdot p_3$ $p_1 + p_2 + \frac{8}{3} \cdot p_3 < \frac{2}{3}$

Tabela 5.11 Przykłady mieszanego kodu Huffmana dla $n = 5$ spełniające warunki 2III

Typ 3I	Typ 3III	Typ 3IV	Typ 3V
$p_1 = 0.18$	$p_1 = 0.17$	$p_1 = 0.11$	$p_1 = 0.13$
$p_2 = 0.18$	$p_2 = 0.17$	$p_2 = 0.11$	$p_2 = 0.13$
$p_3 = 0.18$	$p_3 = 0.17$	$p_3 = 0.18$	$p_3 = 0.13$
$p_4 = 0.20$	$p_4 = 0.17$	$p_4 = 0.21$	$p_4 = 0.24$
$p_5 = 0.26$	$p_5 = 0.32$	$p_5 = 0.39$	$p_5 = 0.37$
$L_2 = 2.36$	$L_2 = 2.34$	$L_2 = 2.22$	$L_2 = 2.26$
$L_{mix} = 2.324$	$L_{mix} = 2.28$	$L_{mix} = 2.196$	$L_{mix} = 2.224$

Tabela 5.12 Zestaw warunków dla drzewa binarnego typu 2IV, przy których dany typ drzewa mieszanego daje najkrótszą średnią długość słowa kodowego

Dla spełnionych warunków $p_1 + p_2 > p_4$ oraz $p_3 + p_4 \leq p_5$			
Typ 3II	Typ 3III	Typ 3IV	Typ 3V
$p_4 + 2 \cdot p_5 > 1$ $p_4 + p_5 > 0.625$ $p_3 < 0.6 \cdot p_4$ $p_5 > 0.4$ $p_3 + 0.4 \cdot p_4 < 0.6 \cdot p_5$ $p_5 + \frac{8}{3} \cdot p_4 > 1$	$p_4 < \frac{2}{3} \cdot p_5$ $p_4 + p_5 < 0.625$ $p_1 + p_2 > 0.6 \cdot p_5$ $\frac{8}{3} \cdot p_4 + p_5 < 1$ $p_5 < 0.4$	$p_3 + 0.4 \cdot p_4 + 2 \cdot p_5 > 1$ $p_3 > 0.6 \cdot p_4$ $p_1 + p_2 < 0.6 \cdot p_5$ $p_1 + p_2 + 1.6 \cdot p_4 < 0.6$ $p_1 + p_2 + 0.4 \cdot p_5 < 0.4$ $p_4 < 0.6 \cdot p_5$	$p_4 + p_5 > 0.6$ $p_5 < 0.4$ $\frac{8}{3} \cdot p_4 + p_5 > 1$ $p_1 + p_2 + 1.6 \cdot p_4 > 0.6$ $p_4 > 0.125 + 0.25 \cdot p_5$ $p_1 + p_2 > 1.5 \cdot p_3$

Tabela 5.13 Przykłady mieszanego kodu Huffmana dla $n = 5$ spełniające warunki 2IV

Typ 3II	Typ 3III	Typ 3IV	Typ 3V
$p_1 = 0.10$	$p_1 = 0.13$	$p_1 = 0.11$	$p_1 = 0.12$
$p_2 = 0.10$	$p_2 = 0.14$	$p_2 = 0.11$	$p_2 = 0.12$
$p_3 = 0.10$	$p_3 = 0.15$	$p_3 = 0.18$	$p_3 = 0.14$
$p_4 = 0.19$	$p_4 = 0.20$	$p_4 = 0.18$	$p_4 = 0.23$
$p_5 = 0.51$	$p_5 = 0.38$	$p_5 = 0.42$	$p_5 = 0.39$
$L_2 = 1.98$	$L_2 = 2.24$	$L_2 = 2.16$	$L_2 = 2.22$
$L_{mix} = 1.97$	$L_{mix} = 2.22$	$L_{mix} = 2.148$	$L_{mix} = 2.208$

Ponieważ $n = 5$, to istnieje także możliwość użycia symboli pięciowartościowych. Gdy poszerzymy klasę kodów mieszanych o parametry $r_2 = 5$, $q_2 = 3$, $b(5^3) = 7$, to opłacalność użycia kodu pięciowartościowego otrzymujemy jedynie dla typu 2III, czyli dla następujących warunków [82]:

$$p_1 + p_2 > p_4 \wedge p_3 + p_4 > p_5 \wedge p_1 + p_2 > \frac{1}{3}$$

Na przykład dla źródła o rozkładzie $P = \{0.18, 0.19, 0.20, 0.21, 0.22\}$ mamy średnią długość słowa kodu binarnego $L_2 = 2.37$. Dla kodu binarno-ternarnego średnia wynosi

$L_{mix} = 2.342$ (typ 3I), natomiast dla kodu pięciowartościowego uzyskujemy najkrótszą średnią $L_5 = \bar{b}_3(5) = \frac{7}{3} = 2.33333$.

Już dla $n = 5$ opis warunków doboru odpowiedniego mieszanego kodu Huffmana jest dość rozbudowany. Ze wzrostem liczby symboli źródła liczba warunków staje się coraz większa, a ponadto dany zestaw warunków jest związany ze z góry zdefiniowaną klasą kodów mieszanych. Te cechy uzasadniają, iż dla $n > 5$ warunkowy opis opłacalności jest niepraktyczny.

5.2. Konstruowanie optymalnych kodów mieszanych

Z wniosków przedstawionych w punkcie 5.1 wynika, że algorytm tworzenia optymalnego drzewa kodu mieszanego w danej klasie nie jest tak prosty jak w przypadku binarnego kodu Huffmana, gdyż także elementy o wyższych prawdopodobieństwach ($p_{r+1}, \dots, p_n$) decydują o tym, czy opłaca się połączenie elementów $p_1, \dots, p_r$ jednym węzłem o r potomkach. Ponadto mamy do czynienia z wieloma różnymi wartościami r_i , które mogą być brane pod uwagę, co także zwiększa liczbę wszystkich możliwych do uzyskania mieszanych drzew Huffmana.

Algorytm znalezienia optymalnego w danej klasie mieszanego kodu Huffmana (czyli dla zdefiniowanych par (r_i, q_i) , gdzie r_i jest liczbą pierwszą taką, że $2 \leq r_i \leq n$), wymaga porównania średniej długości słowa kodowego wszystkich drzew mieszanych możliwych do utworzenia z n elementów (*metoda porównań*). Przy czym tworząc nowy węzeł w należy zachować zasadę sortowania prawdopodobieństw (podobnie jak w algorytmie binarnego kodu Huffmana) i łączyć r_i najmniej prawdopodobnych symboli lub węzłów. Użycie wyłącznie liczb pierwszych jako wartości r_i pozwala na zmniejszenie złożoności algorytmu poszukiwania kodu o najkrótszej średniej długości słowa kodowego L_{mix} . Możemy obliczyć liczbę D wszystkich takich drzew dla źródła o n symbolach. Wyznaczamy wszystkie kolejne liczby pierwsze: $r_0 = 2, r_1 = 3, \dots, r_{m-1} \leq n$, a następnie poszukujemy wszystkich rozwiązań równania:

$$\sum_{i=0}^{m-1} k_i \cdot (r_i - 1) = n - 1 \quad (5.3)$$

z m niewiadomymi k_i , gdzie k_i jest nieujemną liczbą całkowitą, taką że:

$$k_i \leq \left\lfloor \frac{n-1}{r_i-1} \right\rfloor \quad (5.4)$$

oraz k_i oznacza liczbę węzłów r_i -narnych możliwych do uzyskania przy konstrukcji drzewa mieszanego. Jeśli równanie (5.3) ma h rozwiązań, to liczba D wszystkich drzew mieszanych wynosi:

$$D = \sum_{j=1}^h \frac{\left(\sum_{i=0}^{m-1} k_{j_i}\right)!}{k_{j_0}! \cdot k_{j_1}! \cdot \dots \cdot k_{j_{m-1}}!} \tag{5.5}$$

gdzie indeksy k_{j_i} oznaczają odpowiednio: i – numer niewiadomej z j -tego zestawu rozwiązań równania (5.3). Opisany w dalszej części tego punktu przykład 5.1 przedstawia wszystkie kroki wyznaczania liczby D dla źródła o $n = 8$ symbolach.

Ze wzrostem n liczba D szybko rośnie, co można zauważyć na podstawie tabeli 5.14. Jest to w przybliżeniu zależność wykładnicza o współczynniku około 1.794 wyznaczona dla $n \leq 25$. Poniższy wzór na przybliżoną wartość D został wyznaczony na podstawie proporcji dla kolejnych argumentów n (pomijając kilka pierwszych wyrazów, podobnie jak ma to miejsce w przypadku ciągu Fibonacciego):

$$D \approx 33 \cdot (1.794)^{n-8} \tag{5.6}$$

Tabela 5.14 Liczba wszystkich drzew mieszanych dla $n \leq 25$

n	$D_{(n)}$	n	$D_{(n)}$
2	1	14	1101
3	2	15	1977
4	3	16	3539
5	6	17	6350
6	10	18	11376
7	19	19	20401
8	33	20	36560
9	60	21	65548
10	106	22	117484
11	192	23	210617
12	342	24	377524
13	616	25	676764

Praktyczna realizacja algorytmu poszukiwania optymalnego mieszanego kodu Huffmana nie wymaga tworzenia od początku każdego drzewa, tak aby wyliczyć średnią długość słowa kodowego. Jest to autorska propozycja tzw. szybkiej rekurencji z parametrami globalnymi, dzięki czemu nie trzeba przekazywać parametrów do wywoływanej rekurencyjnie funkcji. W ten sposób aktualnie wyznaczany wynik L_{mix} częściowo zależy od poprzednio

wyliczonych wartości składowych. Na początku algorytm buduje binarne drzewo Huffmana, zapamiętując na każdym kroku wartości związane z poszczególnymi tworzonymi węzłami, aż do momentu połączenia dwóch ostatnich elementów tworzących korzeń drzewa. W drugim kroku cofamy się rekurencyjnie o jeden poziom tworzenia drzewa binarnego. Mamy wówczas trzy symbole, które można połączyć korzeniem ternarnym. Następnie cofając się o jeszcze jeden poziom, mamy do dyspozycji już cztery elementy na liście symboli. Można z nich utworzyć jedno drzewo łączące w pierwszym kroku $r = 3$ symbole, a w drugim dwa pozostałe (zgodnie z zasadą, że łącząc r symboli, ich liczba z listy zmniejsza się o $r - 1$). Innych drzew dla czterech symboli nie da się utworzyć, gdyż pierwszy (na tym poziomie) tworzony węzeł musi mieć $r > 2$ potomków, gdzie r jest liczbą pierwszą. Zatem algorytm cofa się rekurencyjnie o kolejny poziom. Mamy już pięć elementów na liście, więc łączymy najpierw $r = 3$ najmniej prawdopodobne, a dla pozostałych wykonujemy algorytm od początku począwszy od bieżącego poziomu rekurencji. Są to dwie możliwości, kod binarny o dwóch węzłach lub jeden węzeł ternarny. Oprócz tych możliwości będąc na poziomie o pięciu symbolach można zamiast pierwszego węzła ternarnego użyć jeszcze korzenia o $r = 5$ potomkach. I znów należy się cofnąć rekurencyjnie o jeden poziom, tak aby uzyskać sześć symboli, z których możemy tworzyć kolejne drzewa, itd. Cofamy się do momentu, gdy pozostanie nam dokładnie n symboli źródła, czyli do zerowego poziomu rekurencji. Wówczas łączymy $r > 2$ najmniej prawdopodobnych symboli tworząc pierwszy niebinarny węzeł, przy czym $r \leq n$. Na każdym ostatnim poziomie rekurencji (w momencie utworzenia korzenia drzewa mieszanego) wyznacza się średnią długość takiego kodu. Jeśli ta średnia jest najkrótsza z dotychczas utworzonych kodów, to kod ten staje się nowym proponowanym kodem mieszanym. Po przeanalizowaniu wszystkich D iteracji otrzymujemy optymalny kod mieszany w danej klasie kodów.

Program napisany w języku C testowany na komputerze klasy Pentium III 500 MHz wykonywał 406222 porównania na sekundę (pomiar wykonany dla $n = 20$). Co oznacza, że na podstawie zależności (5.6) już dla $n = 38$ symboli potrzebowalibyśmy około godziny, aby znaleźć optymalne drzewo kodowe. Jest to główna wada tej propozycji poszukiwania optymalnych mieszanych kodów Huffmana (w odniesieniu do wydajności obecnie stosowanych procesorów przy $n \leq 256$). Kod źródłowy głównych funkcji tego programu znajduje się w dodatku B. W tabeli 5.15 zostały przedstawione wyniki działania programu testującego średnią długość słowa L_{mix} dla wszystkich $D = 33$ drzew o skrótowym zapisie r_w dla ośmiosymbolowego źródła z przykładu 4.1.

Tabela 5.15 Kolejne kroki poszukiwania optymalnego mieszanego kodu Huffmana dla danych z przykładu 4.1 ($n = 8$)

L_{mix}	r_w
2.7600	2 2 2 2 2 2 2
2.7054	2 2 2 2 2 3
2.8222	2 2 2 2 3 2
2.7310	2 2 2 3 2 2
2.7963	2 2 2 3 3
2.7726	2 2 2 5
2.7832	2 2 3 2 2 2
2.7085	2 2 3 2 3
2.8454	2 2 3 3 2
2.8161	2 2 5 2
2.7429	2 3 2 2 2 2
2.7083	2 3 2 2 3
2.8051	2 3 2 3 2
2.7222	2 3 3 2 2
2.7676	2 3 3 3
2.7755	2 3 5
2.7813	2 5 2 2
2.8667	2 5 3
2.9325	2 7
2.7512	3 2 2 2 2 2
2.6966	3 2 2 2 3
2.8134	3 2 2 3 2
2.7222	3 2 3 2 2
2.7876	3 2 3 3
2.7638	3 2 5
2.7927	3 3 2 2 2
2.7380	3 3 2 3
2.8549	3 3 3 2
2.8573	3 5 2
2.7665	5 2 2 2
2.6918	5 2 3
2.8286	5 3 2
2.8844	7 2

Przykład 5.1

Dane jest ośmiosymbolowe źródło S . Dla $n = 8$ mamy $r_0 = 2, r_1 = 3, r_2 = 5, r_3 = 7$, czyli klasa kodu mieszanego może wymagać maksymalnie $m = 4$ buforów r_i -narnych. Podstawiając do (5.3) otrzymujemy równanie postaci:

$$k_0 + 2 \cdot k_1 + 4 \cdot k_2 + 6 \cdot k_3 = 7 \tag{5.7}$$

Wówczas na podstawie zależności (5.4) mamy $k_1 \leq 7, k_2 \leq 3, k_3 \leq 1, k_4 \leq 1$, a z tego wynika siedem zestawów rozwiązań równania (5.7) zawartych w tabeli 5.16:

Tabela 5.16 Zestaw rozwiązań równania (5.7)

k_0	k_1	k_2	k_3	$\sum_{i=0}^3 k_i$
7	0	0	0	7
5	1	0	0	6
3	2	0	0	5
1	3	0	0	4
3	0	1	0	4
1	1	1	0	3
1	0	0	1	2

Zatem na podstawie wzoru (5.5) otrzymujemy liczbę D wszystkich drzew mieszanych równą:

$$D = \frac{7!}{7!} + \frac{6!}{5! \cdot 1!} + \frac{5!}{3! \cdot 2!} + \frac{4!}{1! \cdot 3!} + \frac{4!}{3! \cdot 1!} + \frac{3!}{1! \cdot 1! \cdot 1!} + \frac{2!}{1! \cdot 1!} = 1 + 6 + 10 + 4 + 4 + 6 + 2 = 33$$

5.3. Metoda integralnych poddrzew

W tym punkcie zostanie zaprezentowana pierwsza metoda zmniejszania liczby drzew, dla których należy porównać średnie długości słów kodu mieszanego L_{mix} , aby wyznaczyć optymalny w danej klasie mieszany kod Huffmana. W tym celu należy poszukiwać tzw. *integralnego poddrzewa* składającego się z k najmniej prawdopodobnych symboli spełniających warunek:

$$p_1' = \sum_{i=1}^k p_i \leq p_{k+1} \tag{5.8}$$

Dla takiego poddrzewa należy znaleźć optymalny kod mieszany. Następnie oznaczamy korzeń tego integralnego poddrzewa jako nowy symbol o prawdopodobieństwie p_1' , co daje nam nowy zestaw symboli o niemalejących prawdopodobieństwach $p_1', p_{k+1}, p_{k+2}, \dots, p_n$. Dla tych symboli ponownie próbujemy zastosować metodę integralnych poddrzew. Jeśli nie znajdziemy więcej integralnych poddrzew, to z pozostałych na liście symboli tworzymy optymalny mieszany kod Huffmana, który następnie łączymy z integralnymi poddrzewami.

Twierdzenie 5.1 Jeżeli suma prawdopodobieństw k najmniej prawdopodobnych symboli jest nie większa od prawdopodobieństwa każdego z pozostałych symboli, to tych k symboli może być traktowane jako osobne *integralne poddrzewo*, dla którego rozkład prawdopodobieństw występujących w nim symboli nie ma wpływu na kształt pozostałej części mieszanego kodu Huffmana.

Uzasadnienie powyższego twierdzenia jest proste, gdyż zgodnie ze wzorem (5.8) wartość p_1' stanowi węzeł o najmniejszym prawdopodobieństwie, więc przy tworzeniu drzewa Huffmana jest on bezwarunkowo brany pod uwagę w pierwszej kolejności. Aby metoda integralnych poddrzew dawała najlepsze rezultaty, należy znaleźć najmniejszą z wartości k z przedziału $2 \leq k < n$ spełniającą nierówność (5.8). Przykład 5.2 pokazuje skuteczność działania tej metody zmniejszającej liczbę D .

Przykład 5.2

Dane jest źródło o $n = 14$ symbolach, którego rozkład przedstawia tabela 5.17. Zgodnie z tabelą 5.14 potrzebujemy aż $D_{(14)} = 1101$ iteracji poszukiwania optymalnego drzewa. Jeżeli zastosujemy metodę integralnych poddrzew, to z $n_1 = 10$ najmniej prawdopodobnych symboli możemy uzyskać integralne poddrzewo przedstawione na rysunku 5.7, które wymaga tylko $D_1 = 106$ iteracji poszukiwania optymalnego drzewa (ponieważ $p_1' = \sum_{i=1}^{10} p_i = 0.1 \leq p_{11} = 0.1$). Następnie otrzymujemy drugie integralne poddrzewo złożone z $n_2 = 3$ elementów, bo $p_2' = p_1' + p_{11} + p_{12} = 0.3 \leq p_{13} = 0.31$. Zatem $D_2 = 2$, także dla pozostałych na liście $n_3 = 3$ symboli (p_2', p_{13}, p_{14}) potrzeba jedynie $D_3 = 2$ porównań. Rysunek 5.8 przedstawia nadrzędną część drzewa mieszanego. Łącznie daje to do przetestowania zaledwie:

$$D = D_1 + D_2 + D_3 = 110 \text{ poddrzew.}$$

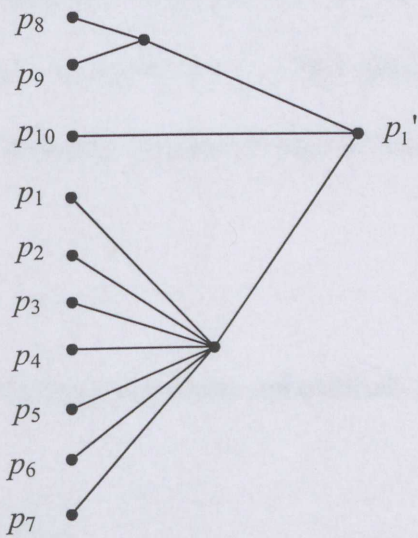
Dla źródeł o większej liczbie symboli zysk jest znacznie większy. Na przykład znalezienie tylko jednego integralnego poddrzewa o $k = 10$ symbolach dla źródła o $n = 25$ symbolach powoduje zmniejszenie się liczby testowanych drzew z $D_{(25)} = 676764$ do $D_{(10)} + D_{(16)} = 106 + 3539 = 3645$ (ponad 185 razy mniej drzew).

Po znalezieniu optymalnego kodu mieszanego dla przykładu 5.2 ustalane są wartości r_i użyte w tym kodzie jako informacja o wynikowej klasie kodu. Do zakodowania źródła wystarczy klasa kodów mieszanych składająca się z dwóch kodów r -narnych, czyli $r_1 = 3, r_2 = 7$ ($q_1 = 41, q_2 = 16$) i z kodu binarnego $r_0 = 2$. Średnia długość tego kodu wynosi

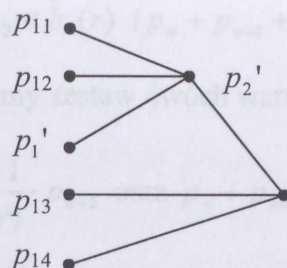
$L_{mix} = L_{2,3,7} = 2.34795$, co odpowiada procentowej efektywności $E_{mix} = 99.51476\%$. Natomiast użycie binarnego kodu Huffmana gwarantuje efektywność na poziomie jedynie $E_2 = 97.15412\%$ przy średniej długości słowa kodowego $L_2 = 2.405$.

Tabela 5.17 Optymalny kod mieszany z przykładu 5.2 w klasie $r_0 = 2, r_1 = 3, r_2 = 7, q_1 = 41, q_2 = 16$

p_i	Kod mieszany
$p_1 = 0.005$	$0_3 2_3 2_3 0_7$
$p_2 = 0.005$	$0_3 2_3 2_3 1_7$
$p_3 = 0.005$	$0_3 2_3 2_3 2_7$
$p_4 = 0.005$	$0_3 2_3 2_3 3_7$
$p_5 = 0.005$	$0_3 0_3 0_3 4_7$
$p_6 = 0.005$	$0_3 2_3 2_3 5_7$
$p_7 = 0.005$	$0_3 2_3 2_3 6_7$
$p_8 = 0.015$	$0_3 2_3 0_3 0_2$
$p_9 = 0.015$	$0_3 2_3 0_3 1_2$
$p_{10} = 0.035$	$0_3 2_3 1_3$
$p_{11} = 0.10$	$0_3 0_3$
$p_{12} = 0.10$	$0_3 1_3$
$p_{13} = 0.31$	1_3
$p_{14} = 0.39$	2_3



Rysunek 5.7 Pierwsze integralne poddrzewo z przykładu 5.2



Rysunek 5.8 Nadrzędna część drzewa z przykładu 5.2

5.4. Poddzewa integralne binarnie

Druga propozycja zmniejszania liczby drzew, dla których należy wyznaczyć średnie długości słów kodu mieszanego L_{mix} , polega na stworzeniu binarnego drzewa Huffmana i dzieleniu go na *poddzewa integralne binarnie*. Możemy wyznaczyć warunek bezwzględnej opłacalności użycia kodu binarnego w danym węźle w_j binarnego drzewa Huffmana. Opłacalność użycia węzła r -narnego uzyskujemy łącząc r elementów o zbliżonych prawdopodobieństwach. W takiej sytuacji należy porównać średnie długości kodu mieszanego $L_r = \bar{b}_q(r)$ i kodu binarnego L_2 , który w takiej sytuacji jest kodem dwufazowym. Największe odchylenie $2\cdot\varepsilon$ między prawdopodobieństwami uzyskamy wówczas dla źródła o rozkładzie $P = \{p - \varepsilon, p, p, \dots, p, p + \varepsilon\}$, gdzie $p = \frac{1}{r}$. Posługując się własnościami kodu dwufazowego (patrz punkt 2.3) możemy wyznaczyć jego średnią długość słowa kodowego dla powyższego rozkładu P :

$$L_2 = b(r) + 1 - \frac{2^{b(r)}}{r} - \varepsilon \tag{5.9}$$

Wówczas odchylenie, dla którego uzyskamy opłacalność użycia węzła r -narnego, spełnia nierówność:

$$\varepsilon < b(r) + 1 - \frac{2^{b(r)}}{r} - \bar{b}_q(r) \tag{5.10}$$

Dla powyższej nierówności ze wzrostem r wartość ε maleje. Oznacza to, iż w przypadku opłacalności użycia kodów r -narnych dla $r \geq 3$ największy rozrzut między prawdopodobieństwami łączonych symboli lub węzłów występuje dla $r = 3$. Wystarczy zatem wyznaczyć warunek, dla którego połączenie trzech elementów o prawdopodobieństwach p_w, p_{w+1}, p_{w+2} przy użyciu kodu binarnego da lepszy rezultat niż użycie kodu ternarnego:

$$2 \cdot (p_w + p_{w+1}) + p_{w+2} < \bar{b}_q(r) \cdot (p_w + p_{w+1} + p_{w+2}) \text{ oraz } p_w + p_{w+1} \leq p_{w+3} \quad (5.11)$$

Po uproszczeniu otrzymujemy zestaw dwóch warunków:

$$p_w + p_{w+1} < \frac{\bar{b}_q(r) - 1}{2 - \bar{b}_q(r)} \cdot p_{w+2} \text{ oraz } p_w + p_{w+1} \leq p_{w+3} \quad (5.12)$$

Algorytm polega na utworzeniu drzewa binarnego, dla którego sprawdzamy każdy węzeł w_j (w kolejności niemalejących prawdopodobieństw) łączący elementy p_w i p_{w+1} . Jeśli warunki (5.12) są spełnione, to z symboli występujących w poddrzewie binarnym, którego korzeniem jest węzeł w o prawdopodobieństwie $p_w + p_{w+1}$, można utworzyć *poddrzewo integralne binarnie*. Dodatkowo wymagane jest, aby prawdopodobieństwa wszystkich k pierwotnych symboli poddrzewa binarnego stanowiły niemalejący ciąg k najmniej prawdopodobnych symboli źródła (tzn. żaden węzeł nie należący do tego poddrzewa nie może tworzyć poddrzewa zawierającego symbole o mniejszym prawdopodobieństwie, niż p_k). Poszukiwanie kolejnych węzłów jest kontynuowane do momentu dotarcia do korzenia głównego drzewa binarnego kodu Huffmana. W praktyce należy łączyć obie metody podziału na poddrzewa integralne (patrz punkt 5.3) oraz na poddrzewa integralne binarnie. Najpierw dzielimy źródło na zestaw integralnych poddrzew zgodnie z twierdzeniem 5.1. Dla znalezionych symboli każdego integralnego poddrzewa tworzymy drzewo binarne Huffmana, w którym poszukujemy punktów podziału na poddrzewa integralne binarnie zgodnie z warunkami (5.12) celem utworzenia jeszcze mniejszych poddrzew. Jeśli znajdziemy poddrzewo integralne binarnie, to dopiero dla niego wyszukujemy optymalnego drzewa mieszane-go. Poniżej znajdują się dwa przykłady, które obrazują sposób łączenia obu opisanych wcześniej metod podziału na poddrzewa.

Przykład 5.3

Dane jest źródło o $n = 10$ symbolach i rozkładzie przedstawionym w tabeli 5.18. Przyjmujemy klasę kodu mieszane-go: $r_0 = 2, r_1 = 3, r_2 = 5, q_1 = 5, q_2 = 3$ (bez użycia kodu dwufazowego). Pozostałe kody r -narne nie wystąpiły w optymalnym mieszanym kodzie Huffmana. Wówczas pierwszy z warunków (5.12) przyjmuje postać:

$$p_w + p_{w+1} < 1.5 \cdot p_{w+2}$$

Zgodnie z tabelą 5.14 potrzebujemy aż $D_{(10)} = 106$ iteracji poszukiwania optymalnego drzewa. Jednak dla $n_0 = 7$ najmniej prawdopodobnych elementów możemy uzyskać inte-

gralne poddrzewo, ponieważ $p_1' = \sum_{i=1}^7 p_i = 0.15 \leq p_8 = 0.18$. Następnie tworzymy z tych siedmiu symboli binarne drzewo Huffmana przedstawione na rysunku 5.9. Zawiera ono w sobie binarnie integralne poddrzewo złożone z $n_1 = 5$ najmniej prawdopodobnych elementów, dla których poszukujemy optymalnego poddrzewa mieszanego (wówczas $D_1 = 6$), bo dla węzłów binarnych mamy prawdopodobieństwa $p_{34} = p_3 + p_4 = 0.02$ oraz $p_{125} = p_1 + p_2 + p_5 = 0.03$. Zatem spełnione są zależności poddrzewa integralnego binarnie:

$$p_2' = p_{34} + p_{125} = 0.05 \leq p_7 = 0.06 \text{ oraz } p_{34} + p_{125} = 0.05 \leq 1.5 \cdot p_6 = 0.06$$

W integralnym poddrzewie mamy już tylko symbole p_2', p_6, p_7 . Dla tych $n_2 = 3$ symboli mamy do sprawdzenia $D_2 = 2$ drzewa, a następnie uzyskujemy optymalne integralne poddrzewo o prawdopodobieństwie korzenia $p_1' = p_2' + p_6 + p_7 = 0.15$. Pozostały więc cztery symbole o prawdopodobieństwach p_1', p_8, p_9, p_{10} , dla których jest spełniony warunek poddrzewa integralnego binarnie:

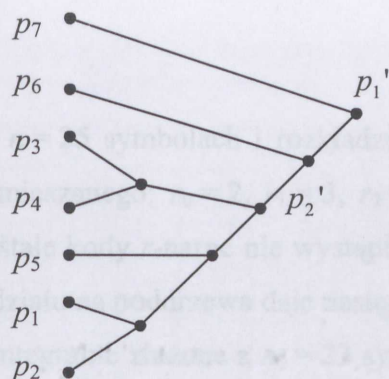
$$p_3' = p_1' + p_8 = 0.33 \leq p_{10} = 0.35 \text{ oraz } p_1' + p_8 = 0.33 \leq 1.5 \cdot p_9 = 0.48,$$

więc łączymy węzłem w_3 elementy o prawdopodobieństwach p_1' oraz p_8 . Ponieważ to poddrzewo zawiera tylko $n_3 = 2$ elementy, to nie potrzeba poszukiwać optymalnego drzewa mieszanego. Dla pozostałych $n_4 = 3$ elementów o prawdopodobieństwach p_3', p_9, p_{10} poszukujemy optymalnego drzewa mieszanego spośród $D_4 = 2$ drzew. Zatem łączna liczba iteracji przy poszukiwaniu optymalnych drzew mieszanych wynosi jedynie:

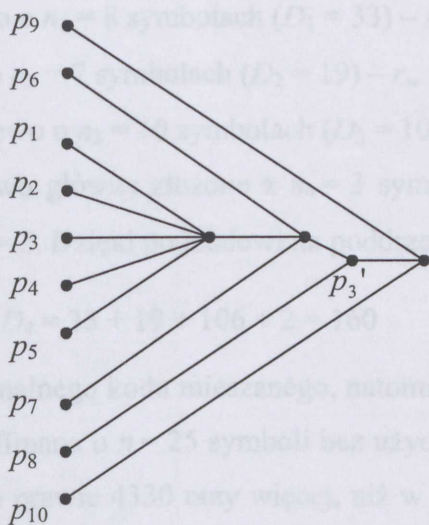
$$D = D_1 + D_2 + D_4 = 6 + 2 + 2 = 10.$$

Tabela 5.18 Optymalny kod mieszany z przykładu 5.3 w klasie $r_0 = 2, r_1 = 3, r_2 = 5, q_1 = 5, q_2 = 3$

p_i	Kod mieszany
$p_1 = 0.01$	1 ₃ 0 ₂ 1 ₃ 0 ₅
$p_2 = 0.01$	1 ₃ 0 ₂ 1 ₃ 1 ₅
$p_3 = 0.01$	1 ₃ 0 ₂ 1 ₃ 2 ₅
$p_4 = 0.01$	1 ₃ 0 ₂ 1 ₃ 3 ₅
$p_5 = 0.01$	1 ₃ 0 ₂ 1 ₃ 4 ₅
$p_6 = 0.04$	1 ₃ 0 ₂ 0 ₃
$p_7 = 0.06$	1 ₃ 0 ₂ 2 ₃
$p_8 = 0.18$	1 ₃ 1 ₂
$p_9 = 0.32$	0 ₃
$p_{10} = 0.35$	2 ₃



Rysunek 5.9 Poddrzewo binarne dla $n_0 = 7$ integralnych symboli z przykładu 5.3



Rysunek 5.10 Drzewo optymalnego kodu mieszanego z przykładu 5.3

Średnia długość takiego kodu wynosi:

$$L_{mix} = 0.33 + 1.15 \cdot \bar{b}_q(3) + 0.05 \cdot \bar{b}_q(5)$$

Kończącą postać optymalnego drzewa mieszanego kodu Huffmana przedstawia rysunek 5.10. Dla zdefiniowanej w przykładzie klasy mamy $L_{mix} = L_{2,3,7} = 2.28667$, co odpowiada procentowej efektywności $E_{mix} = 98.96164\%$. Natomiast użycie binarnego kodu Huffmana gwarantuje efektywność na poziomie jedynie $E_2 = 96.70611\%$ przy średniej długości słowa kodu $L_2 = 2.34$.

Przykład 5.4

Dane jest źródło o $n = 25$ symbolach i rozkładzie przedstawionym w tabeli 5.19. Przyjmujemy klasę kodu mieszanego: $r_0 = 2$, $r_1 = 3$, $r_2 = 5$, $q_1 = 41$, $q_2 = 31$ (bez użycia kodu dwufazowego). Pozostałe kody r -narne nie wystąpiły w optymalnym mieszanym kodzie Huffmana. Użycie podziału na poddrzewa daje następujący rezultat: Otrzymujemy poddrzewo integralne złożone z $n_0 = 23$ symboli i prawdopodobieństwie ko-rzenia $p_0' = 0.33$, które jest następnie dzielone na trzy poddrzewa (w tym dwa pierwsze są integralne binarnie):

- pierwsze poddrzewo o $n_1 = 8$ symbolach ($D_1 = 33$) – $r_w = \{5, 3, 2\}$;
- drugie poddrzewo o $n_2 = 7$ symbolach ($D_2 = 19$) – $r_w = \{3, 2, 2, 2, 2\}$;
- oraz trzecie poddrzewo o $n_3 = 10$ symbolach ($D_3 = 106$) – $r_w = \{5, 3, 2, 2, 2\}$.

Pozostaje jeszcze poddrzewo główne złożone z $n_4 = 3$ symboli o prawdopodobieństwach p_0', p_{24}, p_{25} , wówczas $D_4 = 2$. Dzięki podziałowi na poddrzewa potrzeba łącznie:

$$D = D_1 + D_2 + D_3 + D_4 = 33 + 19 + 106 + 2 = 160$$

iteracji poszukiwania optymalnego kodu mieszanego, natomiast dla znalezienia optymalnego mieszanego drzewa Huffmana o $n = 25$ symboli bez użycia metod podziału potrzeba aż $D = 676764$ iteracji. Jest to prawie 4330 razy więcej, niż w przypadku użycia podziału na poddrzewa. Dzięki użyciu optymalnego mieszanego kodu Huffmana (patrz tabela 5.19 oraz rysunek 5.11) o skróconym zapisie $r_w = \{5, 3, 2, 3, 2, 2, 2, 2, 5, 3, 2, 2, 2, 3\}$ uzyskujemy średnią długość słowa kodowego:

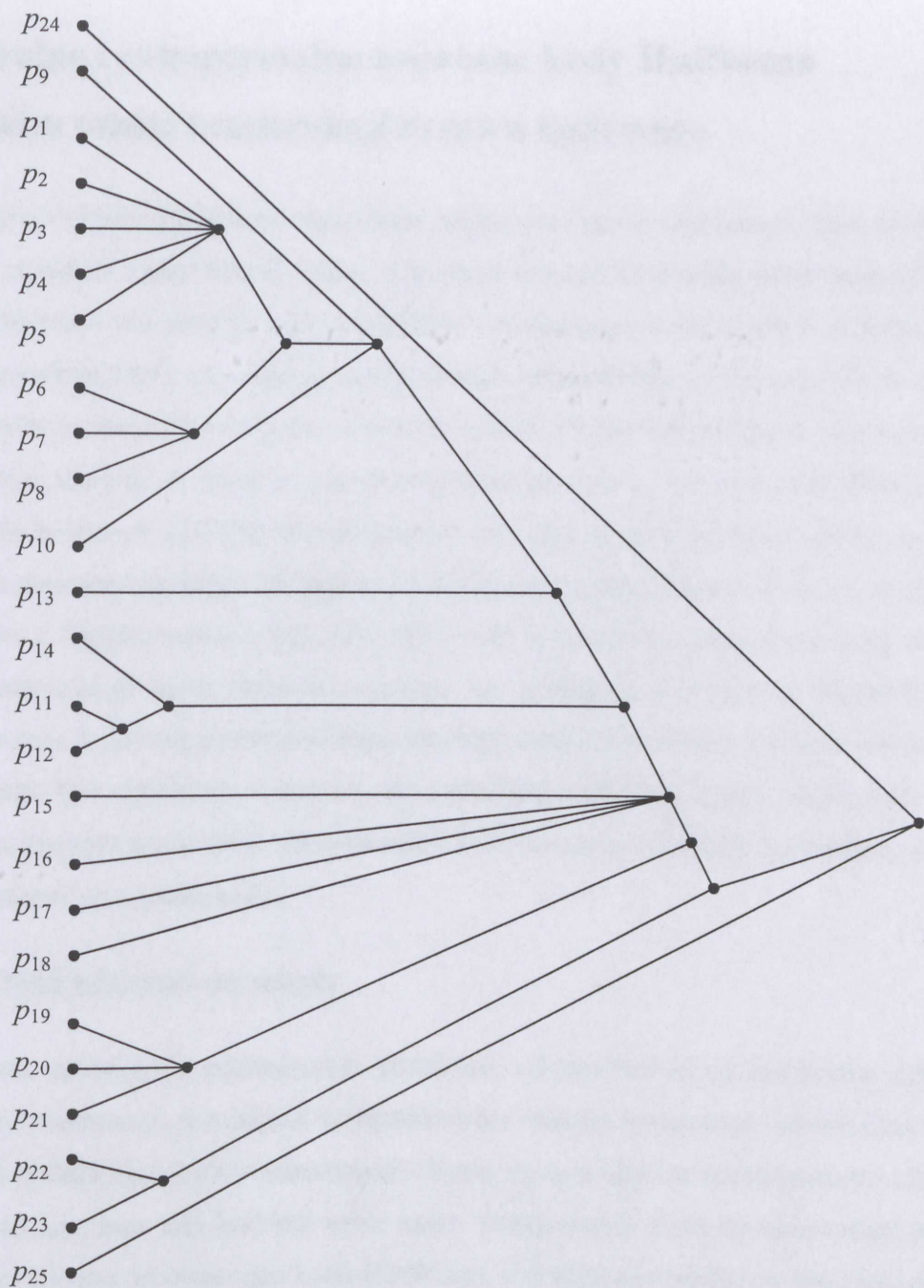
$$L_{mix} = 0.6955 + 1.0819 \cdot \bar{b}_q(3) + 0.077 \cdot \bar{b}_q(5)$$

Dla klasy przedstawionej w przykładzie otrzymujemy średnią długość $L_{mix} = 2.58955$, dla której efektywność wynosi $E_{mix} = 99.93013\%$, a waga zysku $W = 97.85545\%$. Jeśli dodatkowo użyjemy kodu dwufazowego do kompresji liczb B , to efektywność wzrośnie do $\underline{E}_{mix} = 99.94323\%$. Możemy to porównać z klasą kodu binarno-ternarnego, dla której optymalny kod ma strukturę $r_w = \{2, 3, 2, 2, 3, 3, 2, 2, 2, 2, 2, 3, 2, 2, 3, 2, 2, 3\}$, natomiast średnia długość kodu wynosi $L_{2,3} = 2.5905$, co daje efektywność $E_{2,3} = 99.89333\%$. Obie klasy uzyskały znacznie lepszy rezultat w porównaniu do binarnego kodu Huffmana, dla którego średnia długość wynosi $L_2 = 2.6721$, a efektywność zaledwie $E_2 = 96.84281\%$.

Tabela 5.19 Optymalny kod mieszany z przykładu 5.4 w klasie
 $r_0 = 2, r_1 = 3, r_2 = 5, q_1 = 41, q_2 = 31$

p_i	Kod mieszany
$p_1 = 0.0001$	$1_3\ 0_2\ 0_2\ 0_5\ 0_2\ 0_2\ 1_3\ 0_2\ 0_5$
$p_2 = 0.0001$	$1_3\ 0_2\ 0_2\ 0_5\ 0_2\ 0_2\ 1_3\ 0_2\ 1_5$
$p_3 = 0.0001$	$1_3\ 0_2\ 0_2\ 0_5\ 0_2\ 0_2\ 1_3\ 0_2\ 2_5$
$p_4 = 0.0001$	$1_3\ 0_2\ 0_2\ 0_5\ 0_2\ 0_2\ 1_3\ 0_2\ 3_5$
$p_5 = 0.0001$	$1_3\ 0_2\ 0_2\ 0_5\ 0_2\ 0_2\ 1_3\ 0_2\ 4_5$
$p_6 = 0.0002$	$1_3\ 0_2\ 0_2\ 0_5\ 0_2\ 0_2\ 1_3\ 1_2\ 0_3$
$p_7 = 0.0002$	$1_3\ 0_2\ 0_2\ 0_5\ 0_2\ 0_2\ 1_3\ 1_2\ 1_3$
$p_8 = 0.0002$	$1_3\ 0_2\ 0_2\ 0_5\ 0_2\ 0_2\ 1_3\ 1_2\ 2_3$
$p_9 = 0.0010$	$1_3\ 0_2\ 0_2\ 0_5\ 0_2\ 0_2\ 0_3$
$p_{10} = 0.0012$	$1_3\ 0_2\ 0_2\ 0_5\ 0_2\ 0_2\ 2_3$
$p_{11} = 0.0021$	$1_3\ 0_2\ 0_2\ 0_5\ 1_2\ 1_2\ 0_2$
$p_{12} = 0.0021$	$1_3\ 0_2\ 0_2\ 0_5\ 1_2\ 1_2\ 1_2$
$p_{13} = 0.0036$	$1_3\ 0_2\ 0_2\ 0_5\ 0_2\ 1_2$
$p_{14} = 0.0040$	$1_3\ 0_2\ 0_2\ 0_5\ 1_2\ 0_2$
$p_{15} = 0.0152$	$1_3\ 0_2\ 0_2\ 1_5$
$p_{16} = 0.0153$	$1_3\ 0_2\ 0_2\ 2_5$
$p_{17} = 0.0154$	$1_3\ 0_2\ 0_2\ 3_5$
$p_{18} = 0.0155$	$1_3\ 0_2\ 0_2\ 4_5$
$p_{19} = 0.0258$	$1_3\ 0_2\ 1_2\ 0_3$
$p_{20} = 0.0260$	$1_3\ 0_2\ 1_2\ 1_3$
$p_{21} = 0.0262$	$1_3\ 0_2\ 1_2\ 2_3$
$p_{22} = 0.0855$	$1_3\ 1_2\ 0_2$
$p_{23} = 0.09$	$1_3\ 1_2\ 1_2$
$p_{24} = 0.33$	0_3
$p_{25} = 0.34$	2_3

Rysunek 5.11 Drzewo optymalnego kodu mieszanego z przykładu 5.4



Rysunek 5.11 Drzewo optymalnego kodu mieszanego z przykładu 5.4

6. Optymalne i suboptymalne mieszane kody Huffmana o krótkim czasie konstrukcji drzewa kodowego

W tym rozdziale zostaną przedstawione propozycje uzyskania kompromisu między uzyskaniem wysokiej efektywności kodu, a krótkim czasem tworzenia mieszanego kodu Huffmana. Pierwszą propozycją jest utworzenie optymalnego kodu binarno-ternarnego. Następnie omówione będą trzy metody poszukiwania suboptymalnych mieszanych kodów Huffmana, które cechują się krótszym czasem tworzenia drzewa kodowego w porównaniu do podstawowej metody opisanej w poprzednim rozdziale. Są to kolejno: kod dwupoziomowy, metoda kolejnych uściśleń oraz połączenie obu tych propozycji pod nazwą rozszerzonego kodu dwupoziomowego. W punkcie 6.4 zostaną zaprezentowane testy dla rzeczywistych plików z danymi różnego typu, których wyniki wykazują wysoką procentową efektywność rozszerzonego kodu dwupoziomowego na przeciętnym poziomie 99.84558%. Punkt 6.5 zawiera propozycję użycia suboptymalnego kodu mieszanego o stałym opóźnieniu, który może być wykorzystywany np. do transmisji danych w czasie rzeczywistym. Podobne zastosowanie może mieć suboptymalny kod mieszany o małym opóźnieniu, którego cechy opisane są w punkcie 6.6.

6.1. Kod binarno-ternarny

Pierwszą propozycją zmniejszenia złożoności obliczeniowej poszukiwania optymalnego kodu mieszanego jest użycie wyłącznie kodu binarno-ternarnego (dwubuforowego), który jest podzbiorem kodów mieszanych. Może się to wiązać ze zmniejszeniem efektywności kompresji, lecz taki kod ma wiele zalet. Efektywność kodu binarno-ternarnego jest najczęściej wyższa od binarnego kodu Huffmana, a praktyczna realizacja wymaga użycia tylko dwóch buforów: binarnego i ternarnego. Procentowe wyniki testów dla $10 \leq n \leq 15$, w których binarny kod Huffmana uzyskał najkrótszą średnią długość słowa kodowego L_{mix} wśród wszystkich kodów mieszanych (binarnego i binarno-ternarnych) przedstawia tabela 4.7 (klasa testowa: $r_0 = 2$, $r_1 = 3$, $q_1 = 41$). Wyniki testów kodu binarno-ternarnego dla $n < 10$ są prawie identyczne jak dla kodu mieszanego wielobuforowego (patrz tabela 4.7), więc nie umieszczono ich w tabeli 6.1. Dopiero dla $n \geq 10$ wyniki różnią się (i to w niewielkim stopniu) w porównaniu do testów dla wszystkich kodów mieszanych (dla r_i będących liczbą pierwszą).

W przykładzie 4.6 porównano kod mieszany o efektywności $E_{mix} = 99.95836\%$ z kodem binarno-ternarnym, dla którego po pozbyciu się buforów $r_2 = 5$ oraz $r_3 = 7$ uzyskano jedynie niewielki spadek efektywności do $E_{2,3} = 99.89881\%$. Podobnie w przykładzie 5.4 porównano kod mieszany o efektywności $E_{mix} = 99.93013\%$ z kodem binarno-ternarnym, dla którego po pozbyciu się bufora o $r_2 = 5$ uzyskano także niewielki spadek efektywności do $E_{2,3} = 99.89333\%$. Przykłady praktycznego zastosowania kodu binarno-ternarnego do kompresji obrazów zostaną przedstawione w rozdziale 8.

Tabela 6.1 Wyniki testów kodów binarno-ternarnych

Liczba n symboli kodu	Dla klasy idealnej [%]	Dla klasy testowej [%]
10	1.639557	1.671600
11	1.266479	1.303101
12	0.834656	0.856781
13	0.458527	0.467682
14	0.199127	0.204468
15	0.068665	0.068665

Podobnie jak w punkcie 5.2 możemy wyznaczyć liczbę D wszystkich drzew binarno-ternarnych dla źródła o n symbolach. Ponieważ dla rozważanego przypadku istnieją tylko dwie liczby pierwsze $r_0 = 2$ oraz $r_1 = 3$, poszukujemy wszystkich rozwiązań równania:

$$k_0 + 2 \cdot k_1 = n - 1 \tag{6.1}$$

z dwiema niewiadomymi k_0 oraz k_1 , będącymi nieujemnymi liczbami całkowitymi takimi, że $k_0 \leq n - 1$ oraz $k_1 \leq \left\lfloor \frac{n-1}{2} \right\rfloor$. Przy czym k_i oznacza liczbę węzłów r_i -narnych możliwych do uzyskania w konstrukcji drzewa mieszanego. Równanie (6.1) ma $h + 1$ par rozwiązań $\{k_0, k_1\}$ spełniających powyższe ograniczenia, gdzie h dane jest wzorem:

$$h = \left\lfloor \frac{n-1}{2} \right\rfloor \tag{6.2}$$

Wówczas liczba D wszystkich drzew mieszanych wynosi:

$$D = \sum_{j=0}^h \frac{(n-1-j)!}{j! \cdot (n-1-2 \cdot j)!} = \sum_{j=0}^h \binom{n-1-j}{j} \tag{6.3}$$

Na podstawie wartości wyznaczonych ze wzoru (6.3) możemy dojść do wniosku, że liczba $D = F(n)$ dla kodu binarno-ternarnego jest wartością ciągu Fibonacciego, który zdefiniowany jest w następujący sposób:

$$F(0) = 0, F(1) = 1, F(n) = F(n-1) + F(n-2) \text{ dla } n \geq 2 \quad (6.4)$$

Wraz ze wzrostem n stosunek dwóch kolejnych wartości tego ciągu staje się zależnością wykładniczą o ilorazie równym $\frac{\sqrt{5}+1}{2} = 1.61803$ (tzw. złota proporcja). Stąd dla dużych wartości n można wyznaczyć przybliżoną wartość $F(n)$ dla $t \leq n$ na podstawie wzoru:

$$F(n) \approx F(t) \cdot \left(\frac{\sqrt{5}+1}{2} \right)^{n-t} \quad (6.5)$$

Np. dla $t = 10$ mamy $F(t) = 55$, wtedy:

$$F(n) \approx 55 \cdot \left(\frac{\sqrt{5}+1}{2} \right)^{n-10} \quad (6.6)$$

Wartość ciągu można wyznaczyć także ze wzoru [88]:

$$F(n) = \frac{\left(\frac{1+\sqrt{5}}{2} \right)^n - \left(\frac{1-\sqrt{5}}{2} \right)^n}{\sqrt{5}} \quad (6.7)$$

Ciąg Fibonacciego ma wiele interesujących własności występujących także przy innych zagadnieniach związanych z kodem Huffmana. Wydawane jest nawet pismo *Fibonacci Quartely*, w którym przedstawiane są zagadnienia dotyczące tego ciągu [56].

Omawiany w poprzednim rozdziale program potrzebował około godziny (na komputerze klasy Pentium III 500 MHz), aby wyznaczyć optymalny kod mieszany dla źródła o $n = 38$ symbolach. W przypadku kodu binarno-ternarnego jest to niecałe 97 sekund. Ale już dla $n = 44$ symboli czas zwiększa się do prawie 30 minut. Zatem dla źródeł o dużej liczbie symboli nawet algorytm poszukiwania optymalnego kodu binarno-ternarnego okazuje się mało praktyczny. W takich przypadkach należy zastosować przedstawione w kolejnych punktach suboptymalne mieszane kody Huffmana.

Można też poszukiwać innych dwubuforowych kodów mieszanych, np. binarno-pięciowartościowych. W takich przypadkach liczba D wszystkich drzew mieszanych wynosi:

$$D = \sum_{j=0}^h \frac{(n-1-j \cdot (r-2))!}{j! \cdot (n-1-j \cdot (r-1))!} = \sum_{j=0}^h \binom{n-1-j \cdot (r-2)}{j} \quad (6.8)$$

Przy czym równanie:

$$k_0 + k_1 \cdot (r-1) = n-1 \quad (6.9)$$

ma $h+1$ par rozwiązań $\{k_0, k_1\}$ będących nieujemnymi liczbami całkowitymi takimi, że

$k_0 \leq n-1$ oraz $k_1 \leq \left\lfloor \frac{n-1}{r-1} \right\rfloor$, gdzie h dane jest wzorem:

$$h = \left\lfloor \frac{n-1}{r-1} \right\rfloor \quad (6.10)$$

6.2. Suboptymalny kod dwupoziomowy

Ze względu na dużą liczbę iteracji potrzebnych do utworzenia optymalnego drzewa kodu mieszanego dla większych n może pojawić się problem, jeśli użycie metod podziału na poddrzewa integralne i poddrzewa integralne binarnie nie daje podziału ma wystarczająco małe poddrzewa o np. $n_i < 20$ elementach. Można wówczas poszukiwać suboptymalnych drzew. Pierwszą propozycją jest tzw. *kod dwupoziomowy*. Tworzymy binarny kod Huffmana do momentu, aż pozostanie tylko $N < n$ elementów (węzłów lub symboli źródła), które stają się zbiorem symboli źródła $S_0 = \{w_1, w_2, \dots, w_N\}$ o prawdopodobieństwach $p_{w_1}, p_{w_2}, \dots, p_{w_N}$. Z elementów w_i tworzy się tzw. *poddrzewo główne* (poddrzewo pierwszego poziomu), czyli optymalny w danej klasie kod mieszany o N symbolach. Jeśli dany symbol w_i posiada potomków (w utworzonym wcześniej binarnym kodzie Huffmana), to jest on korzeniem *poddrzewa drugiego poziomu*, które jednak nie może posiadać więcej niż N liści, czyli symboli pierwotnego źródła. Dla każdego z tych $k \leq N$ poddrzew (wraz z poddrzewem pierwszego poziomu jest ich maksymalnie $N+1$) należy znaleźć optymalny mieszany kod Huffmana. Średnią długość słowa kodowego całego drzewa kodu mieszanego L_{mix} można wyznaczyć ze wzoru:

$$L_{mix} = L(S_0) + \sum_{i=1}^k p_{w_i} \cdot L(S_i) \quad (6.11)$$

Przy czym średnie długości słowa kodowego wynoszą: $L(S_0)$ dla poddrzewa głównego oraz $L(S_i)$ dla poddrzew drugiego poziomu. Jeśli w_i jest symbolem pierwotnego źródła, to $L(S_i) = 0$, dlatego we wzorze (6.11) rozpatruje się jedynie k poddrzew. Dzięki rozbiciu na

dwa poziomy drzew możliwe jest szybkie uzyskanie suboptymalnego kodu. Minimalnie może on kodować $n_{min} = 2 \cdot N - 1$ symboli dla drzewa Huffmana będącego odpowiednikiem kodu unarnego $\{1, 01, 001, \dots, 0_{(n-1)} \text{ razy } 1, 0_n \text{ razy}\}$ zmodyfikowanego dla najdłuższego słowa kodowego, które zgodnie z własnościami kodu Huffmana powinno mieć długość taką jak przedostatnie słowo kodu. Ma to miejsce, gdy rozkład prawdopodobieństwa symboli źródła jest wykładniczy o proporcji $\lambda = \frac{p_{j-1}}{p_j} \leq \frac{\sqrt{5}-1}{2}$, co wynika z założenia, że łączymy elementy o prawdopodobieństwie $p(S_{n-2})$ i $p_{n-1} = \lambda \cdot p_n$ (ogólnie $p_{n-j} = \lambda^j \cdot p_n$), przy czym spełniony jest warunek zbieżności szeregu geometrycznego $|\lambda| < 1$. Można przyjąć, iż $p(S_{n-2}) = p_n \cdot \frac{\lambda^2}{1-\lambda}$ jest nieskończoną sumą prawdopodobieństw wszystkich najmniej prawdopodobnych elementów, a także $p(S_{n-2}) \leq p_n$ więc $p_n \cdot \frac{\lambda^2}{1-\lambda} \leq p_n$. Rozwiązaniem tej nierówności jest właśnie $0 < \lambda \leq \frac{\sqrt{5}-1}{2}$. Maksymalnie kod dwupoziomowy może kodować $n_{max} = N^2$ symboli, gdy są one równoprawdopodobne i jednocześnie N jest potęgą dwójki. Jeśli np. $N = 13$, to wystarczy maksymalnie $D' = (N+1) \cdot D_{(N)} = 14 \cdot 616 = 8624$ iteracji poszukiwania suboptymalnego drzewa mieszanego. Jednocześnie $n_{min} = 2 \cdot 13 - 1 = 25$, zatem w przypadku poszukiwania optymalnego kodu mieszanego potrzebowalibyśmy co najmniej $D_{(25)} = 676764$ iteracji. Drugą ważną zaletą kodu dwupoziomowego jest mała liczba buforów r -narych, ponieważ $r \leq N$.

Przykład 6.1

Dane jest źródło S zawierające $n = 19$ równoprawdopodobnych symboli ($p_1 = \dots = p_{19} = \frac{1}{19}$). Tworzymy dwupoziomowy suboptymalny kod dla $N = 16$. Powstają $k = 3$ poddrzewa drugiego poziomu o prawdopodobieństwach ich korzeni $p_{w_1} = p_{w_2} = p_{w_3} = \frac{2}{19}$, a także poddrzewo główne składające się z korzeni tych trzech poddrzew $\{w_1, w_2, w_3\}$ oraz z trzynastu równoprawdopodobnych symboli pierwotnego źródła o prawdopodobieństwach $p_{w_4} = \dots = p_{w_{16}} = \frac{1}{19}$. Poszukujemy optymalnego kodu mieszanego dla poddrzewa głównego, a następnie łączymy z kodami symboli w_1, w_2 oraz w_3 tworząc

suboptymalny kod mieszany przedstawiony w tabeli 6.2, którego skrótowy opis $r_w = \{2, 2, 2, 2, 2, 3, 3, 3, 5, 3, 2\}$.

Tabela 6.2 Mieszany kod Huffmana dla przykładu 6.1

p_i	Kod mieszany
$p_1 = \frac{1}{19}$	0 ₂ 2 ₅ 0 ₂
$p_2 = \frac{1}{19}$	0 ₂ 2 ₅ 1 ₂
$p_3 = \frac{1}{19}$	0 ₂ 3 ₅ 0 ₂
$p_4 = \frac{1}{19}$	0 ₂ 3 ₅ 1 ₂
$p_5 = \frac{1}{19}$	0 ₂ 4 ₅ 0 ₂
$p_6 = \frac{1}{19}$	0 ₂ 4 ₅ 1 ₂
$p_7 = \frac{1}{19}$	0 ₂ 0 ₅ 0 ₂
$p_8 = \frac{1}{19}$	0 ₂ 0 ₅ 1 ₂
$p_9 = \frac{1}{19}$	0 ₂ 1 ₅ 0 ₂
$p_{10} = \frac{1}{19}$	0 ₂ 1 ₅ 1 ₂
$p_{11} = \frac{1}{19}$	1 ₂ 0 ₃ 0 ₃
$p_{12} = \frac{1}{19}$	1 ₂ 0 ₃ 1 ₃
$p_{13} = \frac{1}{19}$	1 ₂ 0 ₃ 2 ₃
$p_{14} = \frac{1}{19}$	1 ₂ 1 ₃ 0 ₃
$p_{15} = \frac{1}{19}$	1 ₂ 1 ₃ 1 ₃
$p_{16} = \frac{1}{19}$	1 ₂ 1 ₃ 2 ₃
$p_{17} = \frac{1}{19}$	1 ₂ 2 ₃ 0 ₃
$p_{18} = \frac{1}{19}$	1 ₂ 2 ₃ 1 ₃
$p_{19} = \frac{1}{19}$	1 ₂ 2 ₃ 2 ₃

Kształt drzewa tego kodu przedstawia rysunek 4.6, przy czym korzenie poddrzew drugiego poziomu znajdują się wewnątrz elipsy. Ponieważ $r \leq 16$, to dla liczb pierwszych $r \leq 13$. Po znalezieniu optymalnych poddrzew klasa kodu suboptymalnego staje się kodem wymagającym użycia jedynie trzech buforów [79]:

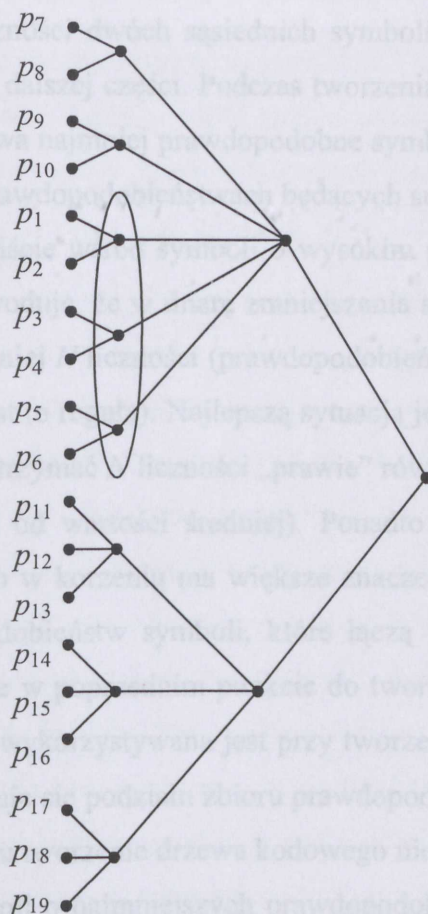
$$,,r_0 = 2, r_1 = 3, r_2 = 5, q_1 = 41, q_2 = 31, \text{ bez użycia kodu dwufazowego}”$$

Średnia długość słowa kodu dla poddrzewa głównego wynosi:

$$L(S_0) = 1 + 2 \cdot \frac{2}{19} + \frac{10}{19} \cdot \frac{72}{31} + 2 \cdot \frac{9}{19} \cdot \frac{65}{41} = 3.93486$$

Natomiast całkowita średnia długość słowa kodu suboptymalnego wynosi:

$$L_{mix} = L(S_0) + 3 \cdot \frac{2}{19} = 4.25065$$



Rysunek 6.1 Drzewo mieszanego kodu Huffmana z przykładu 6.1

Daje to efektywność $E_{mix} = 99.93590\%$ oraz wagę zysku $W = 95.98495\%$, podczas gdy dla binarnego kodu Huffmana mamy $L_2 = 4.31579$, co odpowiada efektywności $E_2 = 98.42759\%$. Gdyby użyć klasy $r_1 = 19$, $q_1 = 4$, to uzyskalibyśmy optymalny mieszany kod Huffmana, dla którego średnia długość słowa kodowego wyniosłaby $L_{19} = 4.25$, a procentowa efektywność $E_{19} = 99.95124\%$ (procentowa waga zysku $W = 96.94603\%$). Jak widać na tym przykładzie, suboptymalny kod dwupoziomowy charakteryzuje się efektywnością niewiele odbiegającą od efektywności dla optymalnego mieszanego kodu Huffmana (waga zysku gorsza o niecały 1%), choć nie jest to regułą dla każdego źródła. Dla kodów suboptymalnych także skrótowy opis drzewa kodowego $r_w = \{\dots\}$ nie zawsze jest możliwy. Dzieje się tak w sytuacji, gdy nie wszystkie spośród r łączonych symboli mają aktualnie najmniejsze prawdopodobieństwa.

6.3. Metoda kolejnych uściśleń

W przypadku posortowanej niemalejąco listy licznosci n symboli źródła S w wielu przypadkach stosunek licznosci dwóch sąsiednich symboli jest znacznie większy na początku tej listy niż w jego dalszej części. Podczas tworzenia binarnego kodu Huffmana łączymy w każdym kroku dwa najmniej prawdopodobne symbole. Po wielu krokach powstają w ten sposób węzły o prawdopodobieństwach będących sumą wielu prawdopodobieństw, które umieszczane są na liście wśród symboli o wysokim prawdopodobieństwie. Ta własność kodu Huffmana powoduje, że w miarę zmniejszania się wielkości listy (po utworzeniu k węzłów), zawarte w niej N licznosci (prawdopodobieństw) staje się bardziej zrównoważoną grupą (choć nie jest to regułą). Najlepszą sytuacją jest znalezienie takiej naturalnej wartości $N = n - k$, aby otrzymać N licznosci „prawie” równoprawdopodobnych (zakładając niewielkie odchylenie od wartości średniej). Ponadto zrównoważenie prawdopodobieństw symboli łączonych w korzeniu ma większe znaczenie dla efektywności kodu niż zrównoważenie prawdopodobieństw symboli, które łączą węzły potomne. Własności te zostały wykorzystane także w poprzednim punkcie do tworzenia kodu dwupoziomowego. Podobna zasada dla $N = 2$ wykorzystywana jest przy tworzeniu kodu Shannona-Fano [67], gdzie rekurencyjnie dokonuje się podziału zbioru prawdopodobieństw na dwie prawie równe części. W tym przypadku tworzenie drzewa kodowego nie rozpoczyna się od najbardziej zagłębionego węzła (symboli o najmniejszych prawdopodobieństwach), lecz od korzenia. Ponieważ propozycja kodów mieszanych pozwala na użycie dowolnej naturalnej wartości $N \geq 2$ (a w szczególności będącej liczbą pierwszą), to pojawia się problem znalezienia najlepszej wartości N . Ponieważ kod Shannona-Fano nie jest optymalny, to propozycja jego rozszerzenia także nie musi być optymalna. Możemy problem optymalnego podziału zastąpić następującym algorytmem, który częściowo wykorzystuje własności binarnego kodu Huffmana:

$N = n$, wówczas początkowa wartość $L_{mix} = \bar{b}_q(n)$;

1° Oblicz średnią długość kodu mieszanego zgodnie ze wzorem (6.11), przy czym

$$L(S_0) = \bar{b}_q(N);$$

2° Zgodnie z zasadą tworzenia binarnego kodu Huffmana połącz dwa najmniej prawdopodobne elementy (symbole lub węzły), tworząc w ten sposób poddrzewo binarne (poddrzewo drugiego poziomu), wtedy $N = N - 1$;

3° Jeśli $N > 1$, to wróć do punktu 1°.

Algorytm kończy działanie, gdy $N = 2$ i otrzymujemy wówczas pełne drzewo binarnego kodu Huffmana. Wówczas wybieramy tę wartość N , dla której uzyskano najkrótszą średnią długość $L_{2,N}$ mieszanego kodu Huffmana wyznaczoną zgodnie ze wzorem (6.11). Przy czym nadrzędne drzewo kodu mieszanego w punkcie 1° składa się z głównego korzenia posiadającego N potomków, z których część (lub wszystkie) to korzenie binarnych poddrzew Huffmana. Ponieważ tych N symboli łączonych jest głównym korzeniem o $r = N$ potomkach przy użyciu kodu r -narnego, wówczas $L(S_0) = \bar{b}_q(N)$. Powstałe w ten sposób binarne poddrzewa (grupy symboli pierwotnego źródła) dzielimy rekurencyjnie w identyczny sposób do momentu, aż dana grupa będzie się składać z pojedynczego symbolu źródła. Jest to tzw. *metoda kolejnych uściśleń*. Podobnie jak w przypadku kodu dwupoziomowego przedstawionego w punkcie 6.2 mamy do czynienia z kodem suboptymalnym. Jednak podstawowa zasada, że efektywność kodu suboptymalnego nie jest mniejsza niż dla binarnego kodu Huffmana, jest zachowana zarówno dla metody kolejnych uściśleń jak i dla kodu dwupoziomowego. Główną zaletą metody kolejnych uściśleń jest bardzo szybkie tworzenie drzewa kodowego. To najszybsza z dotychczas przedstawianych metod uzyskania kodu mieszanego i nawet dla źródeł o 256 symbolach czas utworzenia drzewa jest mierzony w milisekundach (dla procesora Intel Pentium III 500 MHz). Niestety najczęściej wiąże się to z jedynie niewielkim wzrostem efektywności w porównaniu do binarnego kodu Huffmana. W punkcie 6.4 zostaną przedstawione testy porównujące efektywność obu metod.

Używając metody kolejnych uściśleń zostały wykonane także testy dla przykładów przedstawionych w dwóch poprzednich rozdziałach. Dla źródła z przykładu 4.1 uzyskano kod binarno-ternarny zgodny z kolumną drugą tabeli 4.2. Dla pięcioelementowego źródła z przykładu 4.4 optymalnym okazał się jedynie binarny kod Huffmana. Natomiast dla przykładów 4.2, 4.5, 5.2 oraz 5.3 uzyskano kod o identycznej efektywności z optymalnym mieszanym kodem Huffmana. Dla źródła z przykładu 5.4 uzyskano suboptymalny binarno-ternarny kod Huffmana (z węzłem ternarnym między innymi w korzeniu drzewa) o średniej długości słowa kodowego $L_{2,3} = 2.59741$, co daje efektywność $E_{2,3} = 99.62759\%$ (przy efektywności kodu optymalnego $E_{mix} = E_{2,3,5} = 99.93013\%$). Kod suboptymalny dla źródła z przykładu 4.6 przedstawia tabela 6.3 oraz rysunek 6.2. Średnia długość tego kodu wynosi $L_{2,5} = 3.75119$, co daje procentową efektywność $E_{2,5} = 99.23632\%$ (przy efektywności kodu optymalnego $E_{mix} = E_{2,3,5,7} = 99.95836\%$).

Rysunek 6.2 Drzewo wewnętrznego mieszanego kodu Huffmana z przykładu 4.6

6.4. Kod dwupoziomowy z wykorzystaniem metody kolejnych uściśleń

Propozycja użycia kodu dwupoziomowego przedstawiona w punkcie 6.2 może okazać się nieskuteczna, gdy źródło ma zbyt dużą liczbę symboli n i wówczas część poddrzew drugiego poziomu może zawierać większą liczbę symboli niż dopuszczalne N . W najgorszym przypadku źródło może się składać maksymalnie z $2 \cdot N - 1$ symboli (patrz punkt 6.2). W takiej sytuacji można zaproponować połączenie dwóch przedstawionych wcześniej metod. Najpierw tworzymy binarny kod Huffmana do momentu, aż pozostanie tylko $N < n$ elementów (węzłów lub symboli źródła). Stają się one zbiorem symboli źródła $S_0 = \{w_1, w_2, \dots, w_N\}$ o prawdopodobieństwach $p_{w_1}, p_{w_2}, \dots, p_{w_N}$, z których tworzymy poddrzewo główne (kod optymalny w danej klasie kodów mieszanych). Podobnie jak w przypadku kodu dwupoziomowego symbole $w_1, w_2, \dots, w_N$ stanowią zestaw N elementów, na które mogą składać się korzenie poddrzew drugiego poziomu lub symbole pierwotnego źródła o odpowiednio wysokich prawdopodobieństwach. Dla każdego z tych poddrzew budujemy suboptymalny kod w oparciu o metodę kolejnych uściśleń. Połączenie tych propozycji możemy określić mianem *rozszerzonego kodu dwupoziomowego*. Autor rozprawy proponuje, aby ustalić wartość $N = 23$ z buforami r -narymymi ze zbioru $r_i = \{2, 3, 5, 7, 11, 13, 17, 19, 23\}$. Dla takiej wartości N do utworzenia suboptymalnego drzewa kodu mieszanego dla źródła o $n = 256$ symbolach wystarczy niecała sekunda (w przypadku procesora Pentium III 500 MHz). Dzięki użyciu rozszerzonego kodu dwupoziomowego dla przykładu 5.4 uzyskujemy średnią długość słowa kodowego $L_{mix} = 2.58918$, co daje efektywność $E_{mix} = 99.92892\%$. Jest to znacznie lepszy rezultat, niż w przypadku użycia tylko metody kolejnych uściśleń (patrz punkt 6.3).

W tabeli 6.4 znajduje się porównanie średnich długości słów kodowych dla binarnego kodu Huffmana L_2 , metody kolejnych uściśleń L_{mku} oraz dla rozszerzonego kodu dwupoziomowego L_{rkd} . Wyniki uzyskano na podstawie badań dwunastu plików z danymi różnego typu. Średnia długość słowa kodowego nie uwzględnia nagłówka pliku, tak aby wielkość kompresowanego pliku nie miała wpływu na porównywanie efektywności. Na początku testowane były pliki multimedialne zarówno te bez wcześniejszej kompresji (BMP, WAV) jak i skompresowane (JPG, MP3). Druga część testów to różne odmiany dokumentów tekstowych: DOC – dokument Worda, PS – publikacja naukowa w formacie PostScript, DJVU – artykuł Gallagera [27] w formacie DeJaVu (format wykorzystuje kod entropijny do kompresji danych, stąd zastosowanie podstawowego kodu Huffmana nie

spowodowało zmniejszenia średniej długości słowa kodowego), PDF – publikacja Shannona [64], CPP – źródło programu w języku C++. W ostatniej części testów znalazły się: EXE – program wykonywalny realizujący kompresję metodą Huffmana, PPT – prezentacja PowerPoint, a także XLS – arkusz Excela, dla którego niska efektywność kompresji wynika z dużej wartości najbardziej prawdopodobnego symbolu $p_{\max} = 0.62469$. Metoda kolejnych uściśleń okazała się lepsza od binarnego kodu Huffmana jedynie w czterech przypadkach. Natomiast dla wszystkich dwunastu testów najlepszy rezultat uzyskała metoda rozszerzonego kodu dwupoziomowego, dla której uzyskano średni wzrost procentowej efektywności o 0.30868%. Oznacza to średni zysk 2552 bajtów na każdy 1MB skompresowanych danych testowych.

Tabela 6.4 Wyniki testów dla suboptymalnych kodów mieszanych

Typ pliku	Rozmiar [B]	L_2	E_2 [%]	L_{mku}	E_{mku} [%]	L_{rkd}	E_{rkd} [%]
BMP	263222	7.47832	99.70188	7.47832	99.70188	7.45944	99.95415
JPG	896708	7.99791	99.78994	7.99791	99.78994	7.98449	99.95762
MP3	1489609	7.94294	99.78752	7.94294	99.78752	7.92852	99.96896
WAV	572118	7.67210	99.64351	7.67210	99.64351	7.64720	99.96802
PDF	366296	7.92118	99.53184	7.91032	99.66852	7.88599	99.97603
PS	1257608	5.26470	99.32028	5.25391	99.52423	5.23260	99.92957
DJVU	103525	8.00000	99.97188	8.00000	99.97188	7.99908	99.98334
CPP	12328	5.33396	99.31449	5.32028	99.56980	5.30119	99.92832
DOC	505856	4.38683	99.64436	4.38683	99.64436	4.37557	99.90065
XLS	28672	3.45710	98.38101	3.45710	98.38101	3.44751	98.65461
EXE	229376	6.54894	99.59602	6.54894	99.59602	6.52536	99.95584
PPT	1261056	7.91335	99.76010	7.90816	99.82546	7.89674	99.96982
Średnia:	—————	6.65978	99.53690	6.65640	99.59201	6.64031	99.84558

6.5. Suboptymalny kod mieszany ze stałym opóźnieniem

W tej pracy kod mieszany był dotychczas rozważany jako kod z podziałem na wiele buforów r -narych. Najpierw poddawany był kompresji cały ciąg danych, a następnie w pliku wynikowym po nagłówku umieszczano kolejne zakodowane bufory r -narne. Istnieje możliwość przeplatania danych binarnych i r -narych w strumieniu wynikowym, co może być przydatne w przypadku transmisji danych czasu rzeczywistego. Użycie liczb B wprowadza opóźnienie transmisji, gdyż dekodery już w momencie pojawienia się pierwszego słowa kodu zawierającego choć jedną cyfrę r_i -narną (w ogólności f cyfr r_i -narych), powinien mieć dostęp do pierwszej liczby B_i (czyli także do przyszłych $q - f$ cyfr r_i -narych), tak aby móc odczytać (po zdekodowaniu liczby B_i) niezbędnych f cyfr r_i -narych.

Ponieważ wraz z każdym słowem kodowym pojawia się średnio $g(r)$ cyfr r -narnych, to w sytuacji gdy $q > g(r)$ koder wprowadza opóźnienie, gdyż musi zapisać q cyframi najwolniej zapełniający się bufor r -narny i dopiero po zakodowaniu tych cyfr w liczbę B , wysła ją do odbiorcy (podobnie dzieje się z innymi wcześniej zakodowanymi liczbami B_i o większej częstości występowania cyfr r_i -narnych). Opóźnienie będące stosunkiem $(q - g(r))$ cyfr r -narnych do średniej liczby $g(r)$ cyfr r -narnych pojawiających się w każdym kolejnym słowie kodu oznacza średnią liczbę symboli wejściowych, które należy wczytać, aby zapełnić r_i -narny bufor liczby B_i :

$$T(r) = \frac{q}{g(r)} - 1 \quad (6.12)$$

Aby uprościć zasadę opisaną powyżej transmisji i zmniejszyć wprowadzane średnie opóźnienie można utworzyć suboptymalny kod dwubuforowy, gdzie oprócz strumienia binarnego będzie tylko jeden dodatkowy bufor r -narny. Przykładem takiego kodu jest kod binarno-ternarny opisywany w punkcie 6.1. Jeśli jednak cyfry r -narne pojawiają się dość rzadko, czyli gdy wartość $g(r)$ (patrz wzór (4.1)) jest bardzo mała, to i tak opóźnienie średnie jest odwrotnie proporcjonalne do $g(r)$ i może być zbyt duże. Aby temu zapobiec, można przedstawić zasadę dla kodu suboptymalnego zgodnie z poniższą definicją:

Definicja 6.1 Kod mieszany pierwszego rzędu to taki kod, w którym korzeń drzewa kodowego łączy ze sobą $r > 2$ potomków (*korzeń niebinarny*), natomiast pozostałe węzły drzewa kodowego posiadają wyłącznie dwóch potomków. Wówczas $g(r) = 1$.

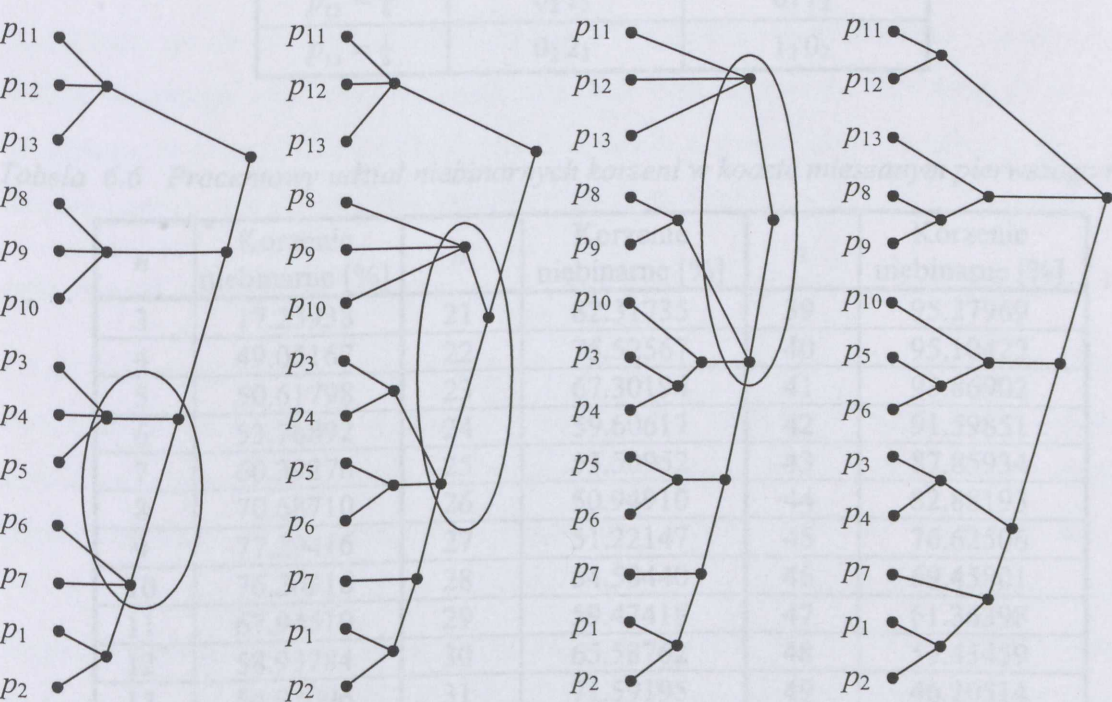
Kod mieszany pierwszego rzędu charakteryzuje stałe opóźnienie wynoszące $q - 1$ symboli wejściowych. Wynika to z faktu, iż każde słowo kodowe zawiera na początku cyfrę r -narną, a do wysłania liczby B potrzeba właśnie q cyfr r -narnych. Dowolny kod mieszany, w którym każde słowo kodu zawiera cyfrę r -narną, można przekształcić do postaci kodu, w którym korzeń drzewa kodowego łączy ze sobą r potomków. Wystarczy iteracyjnie stosować następującą zasadę do momentu uzyskania niebinarnego korzenia:

Jeżeli węzeł r_1 -narny w drzewie Huffmana ma wszystkich potomków r_2 -narnych, to możemy zamienić węzły w ten sposób, że na miejscu r_1 -narnego umieszczamy r_2 -narny i wszystkich jego potomków robimy r_1 -narnych.

Poniżej znajduje się przykład dwóch kodów mieszanych dla źródła S o identycznej efektywności. Wykorzystując trzykrotnie powyższą zasadę zamiany węzłów możemy z kodu mieszanego I otrzymać kod mieszany II (patrz rysunek 6.3 – elipsy zawierają węzły poddawane zamianie).

Przykład 6.2

Dane jest źródło S o $n = 13$ symbolach, których rozkład prawdopodobieństwa przedstawia tabela 6.5. Dla tego źródła możliwe jest utworzenie optymalnego kodu mieszanego (który jest kodem binarno-ternarnym o średniej długości słowa kodowego $L_{2,3} = 3.37703$ i efektywności $E_{2,3} = 99.98806\%$, dla $r_1 = 3$, $q_1 = 41$) na kilka sposobów. Dwa przykładowe kody zawarte są w drugiej i trzeciej kolumnie tabeli 6.5. Pierwszy z tych kodów mieszanych posiada węzły ternarne na różnych poziomach drzewa. Wartość $g(3) = 1$ jest taka sama zarówno w przypadku pierwszego jak i drugiego kodu. Kod mieszany II jest optymalnym (w sensie algorytmu przedstawionego w punkcie 5.2) mieszanym kodem Huffmana pierwszego rzędu.



Rysunek 6.3 Kolejne etapy zamiany drzewa kodu mieszanego I na kod mieszany II

Tabela 6.5 Dwa optymalne kody mieszane dla źródła z przykładu 6.2

p_i	Kod mieszany I	Kod mieszany II
$p_1 = \frac{1}{48}$	$1_2 1_2 1_2 2_3 0_2$	$2_3 1_2 1_2 1_2 0_2$
$p_2 = \frac{1}{48}$	$1_2 1_2 1_2 2_3 1_2$	$2_3 1_2 1_2 1_2 1_2$
$p_3 = \frac{1}{24}$	$1_2 1_2 0_2 0_3$	$2_3 1_2 0_2 0_2$
$p_4 = \frac{1}{24}$	$1_2 1_2 0_2 1_3$	$2_3 1_2 0_2 1_2$
$p_5 = \frac{1}{24}$	$1_2 1_2 0_2 2_3$	$2_3 0_2 1_2 0_2$
$p_6 = \frac{1}{24}$	$1_2 1_2 1_2 0_3$	$2_3 0_2 1_2 1_2$
$p_7 = \frac{1}{24}$	$1_2 1_2 1_2 1_3$	$2_3 1_2 1_2 0_2$
$p_8 = \frac{1}{12}$	$1_2 0_2 0_3$	$1_3 1_2 0_2$
$p_9 = \frac{1}{12}$	$1_2 0_2 1_3$	$1_3 1_2 1_2$
$p_{10} = \frac{1}{12}$	$1_2 0_2 2_3$	$2_3 0_2 0_2$
$p_{11} = \frac{1}{6}$	$0_2 0_3$	$0_3 0_2$
$p_{12} = \frac{1}{6}$	$0_2 1_3$	$0_3 1_2$
$p_{13} = \frac{1}{6}$	$0_2 2_3$	$1_3 0_2$

Tabela 6.6 Procentowy udział niebinarnych korzeni w kodzie mieszanym pierwszego rzędu

n	Korzenie niebinarne [%]	n	Korzenie niebinarne [%]	n	Korzenie niebinarne [%]
3	17.23938	21	82.31735	39	95.27969
4	49.05167	22	75.52567	40	95.10422
5	50.61798	23	67.30194	41	93.86902
6	53.76892	24	59.60617	42	91.59851
7	60.39276	25	53.50952	43	87.85934
8	70.68710	26	50.94910	44	82.88193
9	77.29416	27	51.22147	45	76.62506
10	76.20316	28	54.50440	46	69.45801
11	67.94510	29	59.47418	47	61.36398
12	58.93784	30	65.58762	48	53.45459
13	54.92096	31	71.59195	49	46.20514
14	57.04804	32	77.34985	50	39.65836
15	63.39874	33	82.35474	51	34.89151
16	71.52634	34	86.43646	52	31.75888
17	79.04663	35	89.75449	53	30.40619
18	84.20334	36	92.24167	54	31.01196
19	86.82709	37	93.79044	80	99.07837
20	86.25488	38	94.97757	107	12.09717

Pojawia się pytanie, jak często zastosowanie kodu mieszanego pierwszego rzędu pozwala uzyskać wzrost efektywności w porównaniu do binarnego kodu Huffmana. Tabela

6.6 przedstawia procentowy udział kodów mieszanych pierwszego rzędu, które uzyskały wyższą efektywność od binarnego kodu Huffmana. Do każdego badania wykorzystano $2^{17} = 131072$ testowych źródeł o n symbolach. Rozkład prawdopodobieństwa dla testowych źródeł był generowany w sposób pseudolosowy (generator *Rand* języka C o w przybliżeniu jednostajnym rozkładzie) uzyskując źródła S o rozkładzie liniowym (o średnich wartościach prawdopodobieństw zgodnych ze wzorem (4.8)). Do badania efektywności wykorzystano parametry kodów r -narnych zgodne z tabelą 4.6. Opłacalność użycia kodu mieszanego pierwszego rzędu jest zależna od liczby n symboli źródła. Analizując wyniki z tabeli 6.6 należy odpowiedzieć na pytanie, skąd biorą się takie zmiany wartości procentowych wraz ze wzrostem liczby n . Zależy to od dwóch czynników: rozkładu prawdopodobieństwa (źródło o rozkładzie liniowym) oraz zasady tworzenia drzewa kodowego według algorytmu Huffmana. Rozpatrzmy najprostszy i najczęściej występujący przypadek, czyli kod mieszany (suboptymalny) pierwszego rzędu z węzłem ternarnym w korzeniu, porównując go do binarnego kodu Huffmana. Dla źródła o n symbolach przy konstrukcji drzewa kodowego stosujemy algorytm tworzenia binarnego kodu Huffmana do momentu, aż zostaną trzy elementy. Dla tych trzech symboli o prawdopodobieństwach $\{p_1, p_2, p_3\}$ możemy wyznaczyć średnią długość binarnego kodu Huffmana jako $L_2 = 2 \cdot (p_1 + p_2) + p_3$ oraz długość kodu ternarnego przy $q = 41$ równą $L_3 = \bar{b}_{41}(3) = \frac{65}{41} = 1.58366$ (patrz punkt

5.1.1). Generator źródła S daje losowe wyniki, ale oscylujące wokół wartości średnich. Zatem im wyższa średnia długość L_2 , tym większa szansa, że użycie kodu ternarnego będzie opłacalne (czyli $L_3 < L_2$). Procentowe wyniki opłacalności użycia w korzeniu trzech potomków przedstawia tabela 6.7. Najdłuższe średnie $L_2 = 1.6$ występują dla $n = \{5, 10, 20, 40, 80, \dots\}$ przy jednoczesnym lokalnym maksimum procentowego użycia kodów ternarnych, co wykazuje pewną przewidywalność dla tego typu źródeł. Wraz ze wzrostem n odchylenia generowanych prawdopodobieństw od wartości średniej (zgodnie ze wzorem (4.8)) są proporcjonalnie coraz mniejsze, dlatego bardziej uwidacznia się zależność między średnimi L_2 i L_3 w każdym kolejnym maksimum procentowego użycia kodów ternarnych. Jeśli przeanalizujemy w taki sposób połączenie kodów binarnego i pięciowartościowego, zasada będzie podobna, lecz uzyskamy inne miejsca lokalnych wartości maksymalnych, itd. Użycie dowolnej liczby pierwszej r jako liczby potomków korzenia jest złożeniem takich analiz dla poszczególnych wartości r (patrz tabela 6.6).

Tabela 6.7 Procentowy udział ternarnych korzeni w kodzie binarno-ternarnym pierwszego rzędu dla $q = 41$

n	Klasa rzeczywista [%]	L_2 dla trzech potomków korzenia [bit]
3	17.33938	1.50000
4	49.05167	1.60000
5	48.79456	1.60000
6	43.45932	1.57143
7	43.59894	1.57143
8	51.32599	1.58333
9	59.65500	1.60000
10	59.71146	1.60000
11	51.38626	1.59091
12	40.83633	1.57692
13	33.35342	1.57143
14	31.89392	1.57143
15	36.83319	1.57500
16	46.36307	1.58088
17	57.37000	1.58824
18	66.51688	1.59649
19	71.97647	1.60000
20	72.85690	1.60000
21	68.81485	1.79740
22	60.83221	1.59289
23	50.97199	1.58696
24	40.63263	1.58000
25	31.42319	1.57539
26	24.62235	1.57265
27	19.87991	1.57143
28	18.15567	1.57143
29	19.28635	1.57241
30	22.83936	1.57419
31	28.82385	1.57661
32	36.45477	1.57955
33	45.87479	1.58289
34	55.01862	1.58656
35	63.95340	1.59048
36	71.84677	1.59460
37	78.16544	1.59744
38	83.01544	1.59919
39	85.85968	1.60000
40	87.00867	1.60000
41	83.36258	1.59930

6.6. Optymalny i suboptymalny kod mieszany z małym opóźnieniem

Dla kodów optymalnych (w danej klasie kodów dwubuforowych) procentowy udział niebinarnego korzenia jest mniejszy niż dla kodów mieszanych pierwszego rzędu, gdyż suma opłacalności (suma zysków między zastosowaniem elementów r -narnych zamiast binarnych) użycia wielu kodów r -narnych w różnych węzłach drzewa kodu mieszanego może sprawiać, że w korzeniu nie tak często będzie występowało więcej niż dwóch potomków. Można to stwierdzić na podstawie tabeli 6.8 przedstawiającej procentowy udział optymalnych kodów mieszanych posiadających niebinarny korzeń, które uzyskały wyższą efektywność od binarnego kodu Huffmana (zarówno dla klasy rzeczywistej jak dla klasy idealnej). Test przeprowadzony został dla źródeł o rozkładzie liniowym (wyniki z tabel 6.6, 6.7, 6.8 uzyskano na podstawie tego samego generatora losowego).

Tabela 6.8 Procentowy udział niebinarnych korzeni w optymalnym kodzie mieszanym

n	Klasa idealna [%]	Klasa rzeczywista [%]
3	17.38434	17.23938
4	48.65952	48.38257
5	45.89462	45.55969
6	35.55756	34.98459
7	29.67682	28.66440
8	29.88968	28.06778
9	31.02036	28.39813
10	29.75617	26.71051
11	26.93558	23.59161
12	23.84949	20.37048
13	21.91544	18.47000
14	22.12982	18.73627
15	23.88229	20.39642
16	26.31378	23.32153
17	28.75977	26.65863
18	30.95703	30.46722
19	32.99027	34.70001
20	33.97446	38.63297
21	35.47974	42.47742
22	35.98633	45.92285
23	37.32300	47.45483
24	38.59253	47.51587
25	39.98718	45.18127

Tworząc subobtymalny kod mieszany pierwszego rzędu o r potomkach w korzeniu można także sprawdzić, czy użycie elementu r -narnego dla pozostałych węzłów pozwoli na dalsze skrócenie średniej długości słowa kodowego. Można w tym celu skorzystać z uproszczenia metody kolejnych uściśleń opisanej w punkcie 6.3 ograniczając test do dwóch możliwości: węzeł binarny lub r -narny (gdzie r zostało wcześniej ustalone dla korzenia). Można także rozważać optymalny kod binarno- r -narny, np. binarno-ternarny opisany w punkcie 6.1. Kod mieszany pierwszego rzędu, który wzbogacimy o kolejne węzły r -narne, charakteryzuje się zmiennym średnim opóźnieniem koderu (patrz wzór (6.12)), lecz średnie opóźnienie jest nie większe niż $q - 1$ symboli wejściowych. Zatem dodatkowe węzły r -narne nie tylko zwiększają efektywność kompresji, ale zmniejszają także średnie opóźnienie koderu. Taki dwubuforowy kod o małym opóźnieniu może być stosowany w transmisji danych czasu rzeczywistego, gdzie dane pojawiają się na wejściu koderu na bieżąco. Poszukiwanie optymalnego kodu dwubuforowego wymaga wyznaczenia średniej długości słowa kodowego dla $D(r)$ kodów mieszanych z korzeniem r -narnym, których liczbę można obliczyć ze wzoru:

$$D(r) = \sum_{j=0}^h \frac{(n-r-j \cdot (r-2))!}{j! \cdot (n-r-j \cdot (r-1))!} = \sum_{j=0}^h \binom{n-r-j \cdot (r-2)}{j} \quad (6.13)$$

gdzie h wynosi:

$$h = \left\lfloor \frac{n-r}{r-1} \right\rfloor \quad (6.14)$$

Jeśli szukamy kompromisu między złożonością koderu, średnim opóźnieniem oraz efektywnością kodu, to powinniśmy wybrać jeden spośród tych kodów dwubuforowych o r -narnym korzeniu, których efektywność jest wyższa od efektywności binarnego kodu Huffmana. Rozpatrując źródło z przykładu 4.1 o $n = 8$ symbolach możemy wyznaczyć po jednym przypadku drzew binarno-siedmiowartościowego, binarno-pięciowartościowego

oraz $D(3) = \binom{5}{0} + \binom{4}{1} + \binom{3}{2} = 8$ przypadków drzew binarno-ternarnych (patrz tabela 5.16).

Spośród nich tylko pięć kodów przedstawionych w tabeli 6.9 ma krótszą średnią długość słowa kodowego niż kod binarny, dla którego $L_2 = 2.76$ przy klasie kodów $r_0 = 2$, $r_1 = 3$, $q_1 = 41$ (bez użycia kodu dwufazowego). Dla tych pięciu kodów możemy zmniejszać wzorcowe wartości q_1 odpowiednio do $\{29, 17, 5\}$. Jak widać z tabeli 6.9, duży wpływ na opóźnienie ma dobór wzorcowych wartości q . Dla większych r wpływ ten jest jeszcze większy. Dobór najlepszego kodu zależy od wymagań, jakie narzucamy koderowi. Np. chcemy zna-

leżć kod o najmniejszym średnim opóźnieniu, którego efektywność przekracza 99.15%. Co oznacza, że należy szukać wśród kodów spełniających założenie $L_{mix} < \frac{H(S)}{0.9915} = 2.71279$. Kod o skrótowym zapisie $r_w = \{2, 2, 3, 2, 3\}$ daje najmniejsze opóźnienie $T_{sr} = T(3) = 11.78195$ oraz spełnia założenie, gdyż $L_{mix} = 2.71235 < 2.71279$.

Tabela 6.9 Dwubuforowe kody mieszane o efektywności wyższej, niż dla binarnego kodu Huffmana (dla danych z przykładu 4.1)

$L_{2,3}$	r_w	$g(3)$	$T(3)$
$q_1 = 41$			
2.70537	2 2 2 2 2 3	1.00	40.00000
2.70854	2 2 3 2 3	1.33	29.82707
2.70829	2 3 2 2 3	1.21	32.88430
2.69659	3 2 2 2 3	1.19	33.45378
2.73805	3 3 2 3	1.50	26.33333
$q_1 = 29$			
2.70621	2 2 2 2 2 3	1.00	28.00000
2.70966	2 2 3 2 3	1.33	20.80451
2.70931	2 3 2 2 3	1.21	22.96694
2.69759	3 2 2 2 3	1.19	23.36975
2.73931	3 3 2 3	1.50	18.33333
$q_1 = 17$			
2.70824	2 2 2 2 2 3	1.00	16.00000
2.71235	2 2 3 2 3	1.33	11.78195
2.71177	2 3 2 2 3	1.21	13.04959
2.70000	3 2 2 2 3	1.19	13.28571
2.74235	3 3 2 3	1.50	10.33333
$q_1 = 5$			
2.72000	2 2 2 2 2 3	1.00	4.00000
2.72800	2 2 3 2 3	1.33	2.75940
2.72600	2 3 2 2 3	1.21	3.13223
2.71400	3 2 2 2 3	1.19	3.20168
2.76000	3 3 2 3	1.50	2.33333

W podobny sposób można analizować średnie opóźnienie dla kodu mieszanego o wielu buforach r -narnych. Wówczas średnie opóźnienie wynosi $T_{sr} = \max \{T(r_1), T(r_2), \dots, T(r_{m-1})\}$. Poniżej przedstawione są wyniki analizy kilku wcześniej omawianych przykładów, w których korzeń ma trzech potomków. Dla każdej z niezerowych wartości $g(r)$, gdy dane są następujące pary parametrów wzorcowych: $r_1 = 3, q_1 = 41, r_2 = 5, q_2 = 3, r_3 = 7, q_3 = 11$, wyznaczamy wartość wyrażenia (6.12). Największa spośród wartości $T(r)$ określa średnie opóźnienie T_{sr} kodu mieszanego. Dla źródła z przykładu 4.1 otrzymano kod binarno-ternarny, dla którego $g(3) = 1.19$, czyli $T_{sr} = T(3) = 33.45378$, a także kod mieszany,

dla którego $g(3) = 1$ oraz $g(5) = 0.33$, wówczas $T_{sr} = T(3) = 41$. Dla źródła z przykładu 4.6 otrzymano kod binarno-ternarny, dla którego $g(3) = 1.427$, czyli $T_{sr} = T(3) = 27.73160$, a także kod mieszany, dla którego $g(3) = 1.778$, $g(5) = 0.112$ oraz $g(7) = 0.111$, wówczas $T_{sr} = T(7) = 98.0991$. Dla źródła z przykładu 5.2 otrzymano kod mieszany, dla którego $g(3) = 1.4$ oraz $g(7) = 0.035$, wówczas $T_{sr} = T(7) = 313.28571$. Dla źródła z przykładu 5.3 otrzymano kod mieszany, dla którego $g(3) = 1.15$ oraz $g(5) = 0.05$, wówczas $T_{sr} = T(5) = 59$. Dla źródła z przykładu 5.4 otrzymano kod mieszany, dla którego $g(3) = 1.0819$ oraz $g(5) = 0.077$, wówczas $T_{sr} = T(5) = 37.96104$.

7.1. Własności dynamicznego kodu Huffmana

Działanie dynamicznego (adaptacyjnego) kodu Huffmana polega na tworzeniu drzewa kodowego na bieżąco, co oznacza, że po każdym odczytaniu z wejścia symbolu następuje jego zakodowanie i aktualizacja drzewa. W odróżnieniu od statycznego kodu Huffmana, nie musimy przysyłać razem z plikiem dodatkowego opisu drzewa kodowego (lub opisu liczników symboli źródła). Do podstawowych własności dynamicznego kodu Huffmana można zaliczyć:

- Brak potrzeby użycia informacji o liczniku pozwalający uzyskać dobre rezultaty dla krótkich plików, dla których dołączenie nagłówka wiązałoby się ze zbyt dużym narzutem niwelującym zysk w przypadku użycia kompresji statycznej.
- Możliwość adaptacji, czyli dostosowywania na bieżąco drzewa kodowego w trakcie kompresji do bieżących własności rozkładu prawdopodobieństwa symboli źródła, która pozwala w wielu przypadkach uzyskać lepszą efektywność w porównaniu do statycznej metody Huffmana.
- Możliwość kompresji strumienia danych, którego nie znamy w całości w momencie rozpoczęcia kodowania. Ma to miejsce np. przy transmisji danych pojawiających się na wejściu kodera na bieżąco (w czasie rzeczywistym).

W przypadku kodera dynamicznego musimy się liczyć z tym, że na początku nie wiemy, jakie symbole będą kodowane, więc musimy zarezerwować dodatkowy symbol, który będzie dalej nazywany *znacznikiem* i oznaczany przez „9” [21, 42]. Kod znacznika informuje dekodera, że pojawił się nowy symbol nie występujący wcześniej w drzewie kodowym. Konieczność istnienia dodatkowego symbolu o umownej liczności 0 (najmniej prawdopodobny symbol źródła) powoduje jednak zmniejszenie efektywności kompresji. Wada ta uwiadamia się zwłaszcza dla plików o dużym rozmiarze, w których rozkład prawdopodobieństwa

7. Dynamiczne mieszane kody Huffmana

W tym rozdziale zostaną przedstawione zalety i wady dynamicznego kodu Huffmana. W punkcie 7.2 zostaną zaprezentowane trzy metody wykorzystujące dynamiczną wersję mieszanego kodu Huffmana. Dwa kolejne punkty zawierają przykłady praktycznej realizacji dynamicznego kodu Huffmana, a także testy porównujące binarną wersję oraz omówione wcześniej trzy metody dynamicznych mieszanych kodów Huffmana z ich statycznymi odpowiednikami i kodem arytmetycznym.

7.1. Własności dynamicznego kodu Huffmana

Działanie dynamicznego (adaptacyjnego) kodu Huffmana polega na tworzeniu drzewa kodowego na bieżąco, co oznacza, że po każdym odczytanim z wejścia symbolu następuje jego zakodowanie i aktualizacja drzewa. W odróżnieniu od statycznego kodu Huffmana, nie musimy przysyłać razem z plikiem dodatkowo opisu drzewa kodowego (lub opisu liczności symboli źródła). Do podstawowych własności dynamicznego kodu Huffmana można zaliczyć:

- Brak potrzeby użycia informacji o liczności pozwalający uzyskać dobre rezultaty dla krótkich plików, dla których dołączenie nagłówka wiązałoby się ze zbyt dużym narzutem niwelującym zysk w przypadku użycia kompresji statycznej.
- Możliwość adaptacji, czyli dostosowywania się kształtu drzewa kodowego w trakcie kompresji do lokalnych własności rozkładu prawdopodobieństwa symboli źródła, która pozwala w wielu przypadkach uzyskać lepszą efektywność w porównaniu do statycznej metody Huffmana.
- Możliwość kompresji strumienia danych, którego nie znamy w całości w momencie rozpoczęcia kodowania. Ma to miejsce np. przy transmisji danych pojawiających się na wejściu kodera na bieżąco (w czasie rzeczywistym).

W przypadku kodera dynamicznego musimy się liczyć z tym, że na początku nie wiemy, jakie symbole będą kodowane, więc musimy zarezerwować dodatkowy symbol, który będzie dalej nazywany *znacznikiem* i oznaczany przez „*” [21, 42]. Kod znacznika informuje dekodera, że pojawił się nowy symbol nie występujący wcześniej w drzewie kodowym. Konieczność istnienia dodatkowego symbolu o umownej liczności 0 (najmniej prawdopodobny symbol źródła) powoduje jednak zmniejszenie efektywności kompresji. Wada ta uwiadamia się zwłaszcza dla plików o dużym rozmiarze, w których rozkład prawdopodobieństwa

stwa występowania symboli nie wykazuje lokalnych zmian. Wówczas dynamiczny koder Huffmana daje nieco dłuższy plik wynikowy niż w przypadku zastosowania kodowania statycznego, co zostanie pokazane w przykładzie 7.2. Drugą wadą kodera dynamicznego jest większa złożoność obliczeniowa spowodowana częstą potrzebą aktualizacji kształtu drzewa kodowego. Dla binarnego kodu Huffmana istnieją algorytmy szybkiej aktualizacji drzewa [21, 62]. Jednak dla kodów mieszanych stanowi to duży problem ze względu na dużą złożoność obliczeniową tworzenia optymalnego kodu mieszanego.

7.2. Algorytmy mieszanych dynamicznych kodów Huffmana

Aktualizacja drzewa kodowego opiera się na jego konstrukcji na podstawie listy licznosci symboli posortowanej niemalejąco. Na początku tej listy znajduje się znacznik „*” z licznoscią 0. Jeśli na wejściu pojawił się nowy symbol, to dołączany jest do listy na końcu elementów o licznosci 1. Dla znalezienia optymalnego mieszanego kodu Huffmana należy sprawdzić wszystkie możliwe drzewa zgodne z algorytmem Huffmana, wybierając kod o minimalnej średniej długości słowa kodowego L_{mix} . Aktualizacja drzewa kodowego po znalezieniu optymalnego kodu mieszanego o skrótowym zapisie $r_w = \{r_1, r_2, \dots, r_k\}$ polega na utworzeniu drzewa w następujący sposób:

Dopóki na liście licznosci jest więcej niż jeden symbol:

{

 Połącz r najmniej prawdopodobnych symboli (nadając im kolejno wartości od 0 do $r - 1$) tworząc węzeł o licznosci n_r będącej sumą licznosci łączonych symboli. Węzeł umieszczamy na liście licznosci na końcu elementów o licznosci n_r .

}

Dzięki takiemu algorytmowi (zaproponowanemu przez Vittera w pracy [86] dla drzew binarnych) uzyskujemy nie tylko minimalną średnią długość słowa kodowego L_{mix} , lecz także

minimalną łączną długość ścieżek $\sum_{i=1}^n n_i$, co w przypadku dynamicznego kodu może wpły-

nać pozytywnie na efektywność kompresji [21]. Praktyczna realizacja dla kompresji plików wymaga ustalenia dodatkowych zasad. Symbole źródła traktujemy jako ośmiobitowe elementy. Poniżej znajduje się algorytm *pierwszej metody kodowania dynamicznego*:

Na początku na liście znajduje się tylko znacznik „*” o licznosci 0. Wczytujemy pierwszy symbol wysyłając go bez zmian na wyjście (8 bitów) i umieszczamy go na końcu listy z licznoscią 1. Następnie dopóki są symbole na wejściu:

{ 7.3. Przykłady tworzenia dynamicznego kodu Huffmana

- Wczytaj symbol z wejścia.
 - Jeśli symbol znajduje się na liście licznosci, to wyślij na wyjście odpowiadający mu kod, a następnie zwiększ licznosc tego symbolu o 1.
 - W przeciwnym wypadku wyślij na wyjście kod znacznika, a po nim ośmiobitową reprezentację nowego symbolu. Umieść nowy symbol na końcu listy z licznoscia 1.
 - Dokonaj aktualizacji drzewa kodowego.
- }

Na końcu wyślij na wyjście znacznik „*”, a po nim kod o wartości 256 (kod końca pliku). Aby można było jednoznacznie dekodować liczbę 256, należy ją zapisać jako 9 jedynek „11111111”. Oznacza to, że symbol 255 także powinien być zakodowany przy pomocy 9 bitów – 8 jedynek i zera „111111110”.

Metoda druga zakłada znajomość całego ciągu źródłowego przed rozpoczęciem kodowania. Wówczas znamy długość pliku i nie musimy używać dziewięciobitowego symbolu końca pliku. Wiemy też ile symboli źródła pojawi się w strumieniu danych. Oznacza to, że po nadejściu ostatniego nowego symbolu kod znacznika staje się zbędny, więc usuwamy go z listy symboli źródła. W nagłówku pliku wynikowego zapisujemy długość pliku źródłowego (np. jako liczbę 24 bitową) oraz ośmiobitową liczbę n symboli źródłowych występujących w kodowanym pliku. Wtedy nagłówek składa się tylko z czterech bajtów.

Metoda trzecia różni się od pierwszej tym, że kod mieszany może się składać jedynie z elementów binarnych i ternarnych. Dzięki temu przy niewielkim, często niezauważalnym spadku efektywności zyskujemy na uproszczeniu algorytmu kodowania. Zmniejsza się złożoność obliczeniowa, a także skraca się wielkość opóźnienia występującego w trakcie kompresji. Podobna propozycja dla kodu statycznego została omówiona w punkcie 6.1. Na podstawie wyników testów z punktu 7.4 można uznać dynamiczny kod binarno-ternarny jako najlepszy kompromis między wydajnością a złożonością obliczeniową i wielkością opóźnienia występującego podczas kompresji (oraz ewentualnej transmisji). Metody te nie uwzględniają proponowanego w pracy [27] „wygaszania efektu pamięciowego”, czyli mnożenia tablicy z rozkładem licznosci przez współczynnik z przedziału od 0 do 1 (po zakodowaniu każdych k symboli, gdzie k oznacza stały odstęp między poszczególnymi modyfikacjami rozkładu i np. $k = 256$). Propozycja ta pozwala na zmniejszanie (wygaszanie) prawdopodobieństw symboli, które nie pojawiają się na wejściu kodera od dłuższego czasu. Z własności tej korzysta dynamiczny koder arytmetyczny, którego wyniki zostaną uwzględnione w testach opisanych w punkcie 7.4.

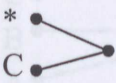
7.3. Przykłady tworzenia dynamicznego kodu Huffmana

Poniżej został przedstawiony przykład kompresji ciągu „CBABAADEBD” przy użyciu binarno-ternarnej wersji dynamicznego kodu Huffmana (metoda trzecia), dla którego liczba B złożona jest z $q = 5$ cyfr ternarnych. Dla tych dziesięciu symboli użycie metody pierwszej daje identyczny kod wynikowy.

Pierwszy symbol „C” – wyślij do bufora binarnego jego ośmiobitową reprezentację „01000011”.

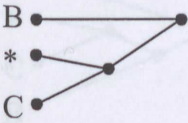
Lista	*	C
Liczność	0	1

Drzewo kodowe ma postać:



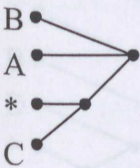
Drugi symbol „B” – nie ma go na liście, więc wyślij do bufora binarnego kod znacznika „0” oraz reprezentację binarną symbolu „B” = „01000010”. Po aktualizacji:

Lista	*	C	B
Liczność	0	1	1



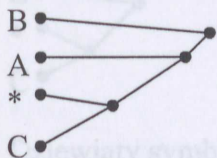
Trzeci symbol „A” – nie ma go na liście, więc wyślij do bufora binarnego kod znacznika „10” oraz reprezentację binarną symbolu „A” = „01000001”. Po aktualizacji:

Lista	*	C	B	A
Liczność	0	1	1	1



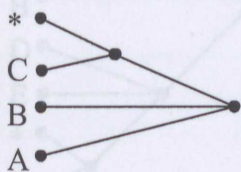
Czwarty symbol „B” – wyślij do bufora ternarnego kod symbolu „0₃”. Po aktualizacji:

Lista	*	C	A	B
Liczność	0	1	1	2



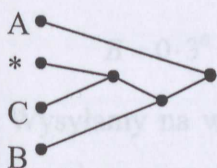
Piąty symbol „A” – wyślij do bufora binarnego kod symbolu „10”. Po aktualizacji:

Lista	*	C	B	A
Liczność	0	1	2	2



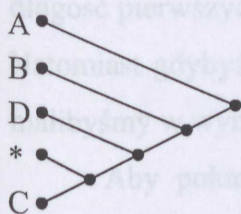
Szósty symbol „A” – wyslij do bufora ternarnego kod symbolu „2₃”. Po aktualizacji:

Lista	*	C	B	A
Liczność	0	1	2	3



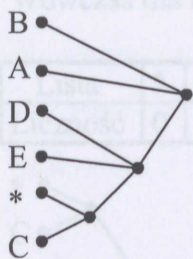
Siódmy symbol „D” – nie ma go na liście, więc wyślij do bufora binarnego kod znacznika „100” oraz reprezentację binarną symbolu „D” = „01000100”. Po aktualizacji:

Lista	*	C	D	B	A
Liczność	0	1	1	2	3



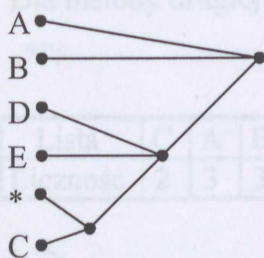
Ósmy symbol „E” – nie ma go na liście, więc wyślij do bufora binarnego kod znacznika „1110” oraz reprezentację binarną symbolu „E” = „01000101”. Po aktualizacji:

Lista	*	C	D	E	B	A
Liczność	0	1	1	1	2	3



Dziewiąty symbol „B” – wyślij do bufora ternarnego kod symbolu „0₃”. Po aktualizacji:

Lista	*	C	D	E	A	B
Liczność	0	1	1	1	3	3



Dziesiąty symbol „D” – wyślij do bufora ternarnego kod symbolu „2₃0₃”. Ponieważ bufor ternarny został zapelniony (jest w nim już pięć cyfr), to tworzymy na jego podstawie liczbę B :

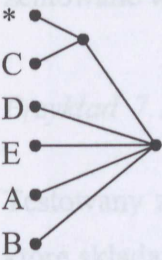
$$B = 0 \cdot 3^0 + 2 \cdot 3^1 + 0 \cdot 3^2 + 2 \cdot 3^3 + 0 \cdot 3^4 = 60$$

Wysyłamy na wyjście reprezentację binarną ośmiobitowej liczby $60 = „00111100”$, a następnie zawartość bufora binarnego. Teraz oba bufor są puste. Podczas dalszego kodowania ponownie wypełniamy je aż do momentu, gdy znów w buforze ternarnym znajdzie się pięć cyfr, poczym wysyłamy zawartość buforów wynikowych na wyjście, itd. Długość pliku powstałego z powyższego ciągu dziesięciu symboli po kompresji wynosi $8 + 52 = 60$ bitów. Gdyby kontynuować kompresję używając wartość $q = 41$ dla bufora ternarnego, to długość pierwszych dziesięciu opisanych powyżej symboli skróciłaby się do 59.92683 bita. Natomiast gdybyśmy użyli wyłącznie binarnego dynamicznego kodu Huffmana, to otrzymalibyśmy w wyniku ciąg złożony z 60 bitów.

Aby pokazać różnice między poszczególnymi metodami przyjmijmy, że źródło składało się $n = 5$ symboli $S = \{A, B, C, D, E\}$, a po uprzednio opisanych 10 elementach wczytano z wejścia ciąg „EEDC”.

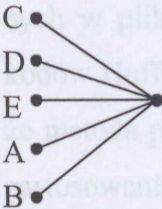
Wówczas dla metody pierwszej otrzymamy drzewo z korzeniem o pięciu potomkach:

Lista	*	C	A	B	E	D
Liczność	0	2	3	3	3	3



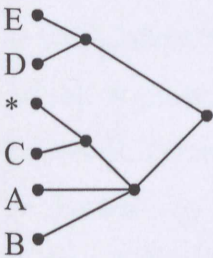
Dla metody drugiej otrzymamy drzewo z korzeniem o pięciu potomkach, ale bez znacznika „*“:

Lista	C	A	B	E	D
Liczność	2	3	3	3	3



Dla metody trzeciej otrzymamy drzewo o skrótowym opisie $r_w = \{2, 3, 2, 2\}$ i licznosci:

Lista	*	C	A	B	E	D
Liczność	0	2	3	3	3	3



7.4. Testy porównawcze dla statycznych i dynamicznych metod entropijnych

Poniżej znajdują się testy porównujące opisane w punkcie 7.2 trzy typy mieszanych kodów dynamicznych ze statycznym i binarnym dynamicznym kodem Huffmana. Zarówno dla statycznej i dynamicznej wersji kodów mieszanych obowiązywała ta sama klasa parametrów mieszanego kodu Huffmana przedstawiona w tabeli 4.6. Dla porównania przedstawione są także wyniki dla statycznego kodu arytmetycznego, który dzięki wysokiej precy-

zji zapisu skumulowanych licznosci uzyskuje wyniki bliskie wartosci wyliczonej z entropii dla symboli niezaleznych (1.2), a takze wyniki dla dynamicznego kodera arytmetycznego. Dla wersji statycznych zastosowano efektywne metody kodowania naglowka pliku zaprezentowane w punkcie 2.8.

Przyklad 7.1

Testowany zbior ma dlugosc 230103 bajtow. Tabela 7.1 zawiera licznosci symboli zrodla, ktore sklada sie z $n = 12$ symboli o rozkladzie geometrycznym dla $p = 0.7$. Minimalna dlugosc pliku wedlug entropii wynosi 81423.39 bajtow. Dla statycznego mieszanego kodu Huffmana dostajemy srednia $L_{mix} = 2.85166$ oraz drzewo o skrotowym zapisie $r_w = \{2, 2, 2, 3, 2, 3, 2, 3\}$. W przypadku binarnego kodu Huffmana srednia wynosi $L_{mix} = 2.85902$. Tabela 7.3 zawiera liczbe poszczegolnych symboli r_i -narnych wystepujacych w plikach wynikowych po zastosowaniu trzech metod dynamicznych mieszanym kodow Huffmana. Na podstawie wynikow przedstawionych w tabeli 7.2 mozna zauwazyc, ze metoda pierwsza i trzecia daly identyczny rezultat. Nieco lepszy wynik uzyskano przy zastosowaniu drugiej metody mieszanego kodu dynamicznego, ktora dorownala wynikowi statycznej wersji mieszanego kodu Huffmana.

Tabela 7.1 Licznosc symboli dla przykladu 7.1

Symbol	Licznosc
A	70000
B	49000
C	34300
D	24010
E	16807
F	11765
G	8235
H	5765
I	4035
J	2825
K	1977
L	1384

Tabela 7.2 Wyniki testów dla przykładu 7.1

Typ kompresji	Długość pliku [B]	W tym nagłówek [B]
Mieszany kod dynamiczny	82235	brak
Binarny kod dynamiczny	82422	brak
Mieszany kod dynamiczny (metoda druga)	82092	4
Binarny kod dynamiczny (metoda druga)	82320	4
Binarno-ternarny kod dynamiczny (metoda trzecia)	82235	brak
Statyczny mieszany kod Huffmana	82081	58
Statyczny binarny kod Huffmana	82292	58
Dynamiczny kod arytmetyczny	82468	brak
Statyczny kod arytmetyczny	81483	57.875

Tabela 7.3 Liczba symboli r_i -narnych w kodzie wynikowym dla przykładu 7.1

Typ kodu mieszanego	Liczba bitów	Liczba symboli ternarnych	Liczba symboli dla $r = 5$
Metoda pierwsza	170254	307565	7
Metoda druga	136470	328137	7
Metoda trzecia	170263	307570	

Przykład 7.2

Dany jest zbiór długości 1000000 bajtów. Źródło składa się z $n = 14$ symboli o rozkładzie przedstawionym w tabeli 5.17. Minimalna długość pliku według entropii 292069.57 bajtów. Szczegółowy opis parametrów statycznej wersji tego kodu został przedstawiony w przykładzie 5.2. Tabela 7.5 zawiera liczbę poszczególnych symboli r_i -narnych występujących w plikach wynikowych po zastosowaniu trzech metod dynamicznych mieszanych kodów Huffmana. Na podstawie wyników przedstawionych w tabeli 7.4 można zauważyć, że dynamiczny kod binarno-ternarny okazał się gorszy od metody pierwszej jedynie o 59 bajtów. Także w tym teście druga wersja dynamicznego mieszanego kodu prawie dorównuje statycznej wersji mieszanego kodu Huffmana.

Tabela 7.4 Wyniki testów dla przykładu 7.2

Typ kompresji	Długość pliku [B]	W tym nagłówek [B]
Mieszany kod dynamiczny	294155	brak
Binarny kod dynamiczny	301421	brak
Mieszany kod dynamiczny (metoda druga)	293790	4
Binarny kod dynamiczny (metoda druga)	301206	4
Binarno-ternarny kod dynamiczny (metoda trzecia)	294214	brak
Statyczny mieszany kod Huffmana	293560	66
Statyczny binarny kod Huffmana	300691	66
Dynamiczny kod arytmetyczny	296005	brak
Statyczny kod arytmetyczny	292138	66

Tabela 7.5 Liczba symboli r_i -narnych w kodzie wynikowym dla przykładu 7.2

Typ kodu mieszanego	Liczba bitów	Liczba symboli ternarnych	Liczba symboli dla $r = 5$	Liczba symboli dla $r = 7$
Metoda pierwsza	36737	1401252	2	33777
Metoda druga	75361	1434942	5	
Metoda trzecia	109799	1415389		

Przykład 7.3

Dany jest zbiór długości 340000 bajtów. Bezpamięciowe źródło składa się z $n = 17$ równoprawdopodobnych symboli. Minimalna długość pliku według entropii 173717.17 bajtów. Tabela 7.7 zawiera liczbę poszczególnych symboli r_i -narnych występujących w plikach wynikowych po zastosowaniu trzech metod dynamicznych mieszanych kodów Huffmana.

Tabela 7.6 Wyniki testów dla przykładu 7.3

Typ kompresji	Długość pliku [B]	W tym nagłówek [B]
Mieszany kod dynamiczny	176462	brak
Binarny kod dynamiczny	177700	brak
Mieszany kod dynamiczny (metoda druga)	175111	4
Binarny kod dynamiczny (metoda druga)	175096	4
Binarno-ternarny kod dynamiczny (metoda trzecia)	176462	brak
Statyczny mieszany kod Huffmana	173929	65
Statyczny binarny kod Huffmana	175083	65
Dynamiczny kod arytmetyczny	175151	brak
Statyczny kod arytmetyczny	173784	64.625

Tabela 7.7 Liczba symboli r_i -narych w kodzie wynikowym dla przykładu 7.3

Typ kodu mieszanego	Liczba bitów	Liczba symboli ternarnych	Liczba symboli dla $r = 5$	Liczba symboli dla $r = 7$
Metoda pierwsza	837921	361507	277	3
Metoda druga	1382209	7278	3036	18
Metoda trzecia	837927	361911		

Na podstawie wyników przedstawionych w tabeli 7.6 można zauważyć, że metoda pierwsza i trzecia dały identyczny rezultat. Znacznie lepszy wynik uzyskano przy zastosowaniu metody drugiej, lecz mimo to jest on gorszy od statycznej wersji mieszanego kodu Huffmana aż o 1182 bajty. Jest to najgorszy przypadek, gdyż dla długiego pliku o n równoprawdopodobnych i wzajemnie niezależnych symbolach źródła zalety kodów dynamicznych nie mogły się uwidocznić.

Przykład 7.4

Testowany zbiór ma długość 262144 bajtów. Źródło jest obrazem *Lena* w 16 odcieniach szarości przy rozdzielczości 512 na 512 pikseli. Liczności występujących w obrazie kolorów przedstawia tabela 7.8. Minimalna długość pliku według entropii wynosi 121301.87 bajtów. Dla statycznego mieszanego kodu Huffmana dostajemy średnią $L_{mix} = 3.720822$ oraz drzewo o skrótowym zapisie $r_w = \{2, 2, 2, 3, 2, 2, 3, 2, 2, 2, 2, 2, 2\}$.

Tabela 7.8 Liczność symboli dla przykładu 7.4

Kolor	Liczność
1	31665
14	28955
9	28603
10	23920
8	23173
6	22835
4	17313
5	17287
2	16882
12	13114
3	12500
13	10773
11	7920
0	5799
7	1382
15	23

Tabela 7.10 zawiera liczbę poszczególnych symboli r -narnych występujących w plikach wynikowych po zastosowaniu trzech metod dynamicznych mieszanych kodów Huffmana. Testowany obraz był zbiorem symboli o zmieniających się lokalnie własnościach rozkładu prawdopodobieństwa występowania kolorów, o czym może świadczyć bardzo dobry wynik dynamicznego koderu arytmetycznego. Jak wynika z tabeli 7.9 dopiero w tym przykładzie każdy z koderów dynamicznych okazał się efektywniejszy od swoich statycznych odpowiedników. I właśnie w przypadku plików o takich własnościach zaleca się stosowanie kodów dynamicznych.

Tabela 7.9 Wyniki testów dla przykładu 7.4

Typ kompresji	Długość pliku [B]	W tym nagłówek [B]
Mieszany kod dynamiczny	121951	brak
Binarny kod dynamiczny	122831	brak
Mieszany kod dynamiczny (metoda druga)	121951	4
Binarny kod dynamiczny (metoda druga)	122831	4
Binarno-ternarny kod dynamiczny (metoda trzecia)	121952	brak
Statyczny mieszany kod Huffmana	121987	63
Statyczny binarny kod Huffmana	122868	63
Dynamiczny kod arytmetyczny	118115	brak
Statyczny kod arytmetyczny	121367	62.75

Tabela 7.10 Liczba symboli r -narnych w kodzie wynikowym dla przykładu 7.4

Typ kodu mieszanego	Liczba bitów	Liczba symboli ternarnych	Liczba symboli dla $r = 5$
Metoda pierwsza	449435	187814	98345
Metoda druga	449597	187861	98230
Metoda trzecia	646845	207378	

Na podstawie przeprowadzonych badań wynika, że zastosowanie kodów dynamicznych jest uzasadnione w szczególnych przypadkach. Np. wzrost efektywności w porównaniu do metody statycznej uzyskujemy dla kompresji krótkich plików lub danych o lokalnie zmiennym rozkładzie prawdopodobieństwa występowania symboli źródła. Główną zaletą kodów dynamicznych jest możliwość kompresji strumienia danych pojawiających się na wejściu koderu na bieżąco, natomiast wadą jest większa złożoność obliczeniowa w porównaniu do kodów statycznych. Spośród trzech przedstawionych algorytmów dynamicznej kompresji z wykorzystaniem mieszanych kodów Huffmana najlepsza okazała się metoda 2, która wymaga jednak wiedzy o zawartości całego pliku przed jego zakodowaniem. Można pomi-

nać ten warunek ustalając np. $n = 256$. Jeśli w źródle wystąpią wszystkie symbole (np. ustalonych wcześniej 256 symboli), to metoda będzie działać prawidłowo, w przeciwnym przypadku wynik będzie zgodny z metodą pierwszą [42]. Metoda trzecia, to dynamiczny kod binarno-ternarny, który wprowadza nieduże opóźnienia i ma mniejszą złożoność od metody pierwszej, przy niewiele niższej efektywności kompresji. Pozwala on jednocześnie uzyskać wzrost efektywności względem binarnej dynamicznej wersji kodu Huffmana. Cechy te pozwalają uznać dynamiczny kod binarno-ternarny jako najlepszy kompromis między wydajnością a wielkością opóźnienia występującego podczas kompresji.

8.1. Kompresja elementów DC w standardzie JPEG

Jednym z najbardziej popularnych standardów strumienia kompresji obrazów jest JPEG. Ostatnim etapem kompresji w algorytmie JPEG jest kodowanie współczynników DCT [6, 20, 57]. Dla współczynników AC i DC stosuje się osobne metody kodowania ze względu na ich odmienne własności. Elementy AC kodowane są przy użyciu algorytmu RLE oraz binarnego kodu Huffmana. Natomiast w przypadku elementów DC koduje się tylko błąd predykcji, czyli różnicę aktualnej i poprzednio kodowanej wartości współczynnika DC. Następnie błędy te koduje się przy pomocy modyfikacji kodu Huffmana. Symbol kodowy składa się z przedrostka (słowa kodu Huffmana) określającego numer grupy oraz dodatkowych bitów, za pomocą których można zapisać reprezentację współczynnika DC w danej grupie. Grupa o numerze n posiada n dodatkowych bitów i określone zakodowanie liczb z przedziału od $-2^{n-1} + 1$ do -2^{n-1} oraz od 2^{n-1} do $2^n - 1$. Przy takiej strukturze danych dopasowuje się tabelę kodową zawierającą maksymalnie 16 grup. Propozycję przez JPEG uniwersalnej tabeli kodu Huffmana dla różnic współczynników luminancji DC przedstawia tabela 8.1 [21]. Przy tej samej tabeli kodowej sformułowanej przez twórców schematu JPEG, powstała na podstawie pomiarów wielu testowych obrazów. Podczas badań autor rozprawy przygotował do testów własny zestaw 12 obrazów (podobnie jak w przypadku 256 odcięci szarości), na podstawie których stworzone zostały dwie tabele (dla standardowej wartości dzielnika kwadratów elementów DC równego 16): dla binarnego kodu Huffmana oraz modyfikowanego kodu Huffmana (patrz tabela 8.2). Są to zachowane standardowe obrazy testowe (airfield, airplane, boat, bridge, camera, couple, crowd, lena, man, peppers, woman /) wykorzystywane np. w pracy [32] jak i zdjęcie z aparatu cyfrowego (osłona – ze zdjęć własnych). Powstała uniwersalna tabela n -ta może być dla każdego obrazu optymalna, więc przy jej użyciu podczas testów używano średniej długości słowa kodowego $L_c = 2,71934$, co dało efektywność kompresji szlifów $E_s = 92,47308\%$. Dla

8. Zastosowania mieszanych kodów Huffmana

W tym rozdziale zostaną przedstawione praktyczne zastosowania mieszanych kodów Huffmana do kompresji elementów *DC* w standardzie JPEG, a także do bezstratnej kompresji obrazów. W punkcie 8.3 zostanie przedstawiony algorytm szybkiej konwersji z liczby ternarnej zakodowanej binarnie do bloku cyfr ternarnych. Ostatni punkt zawiera prezentację autorskiej propozycji zastosowania kodów mieszanych w celu zwiększenia efektywności kodu Golomba, poprzedzonej omówieniem właściwości tego kodu.

8.1. Kompresja elementów *DC* w standardzie JPEG

Jednym z najbardziej popularnych standardów stratnej kompresji obrazów jest JPEG. Ostatnim etapem kompresji w algorytmie JPEG jest kodowanie współczynników DCT [6, 20, 57]. Dla współczynników *AC* i *DC* stosuje się osobne metody kodowania ze względu na ich odmienne własności. Elementy *AC* kodowane są przy użyciu algorytmu *RLE* oraz binarnego kodu Huffmana. Natomiast w przypadku elementów *DC* koder zapisuje tylko błąd predykcji, czyli różnicę aktualnej i poprzednio kodowanej wartości współczynnika *DC*. Następnie błędy te koduje się przy pomocy modyfikacji kodu Huffmana. Symbol kodowy składa się z przedrostka (słowa kodu Huffmana) określającego numer grupy oraz dodatkowych bitów, za pomocą których można zapisać reprezentację współczynnika *DC* w danej grupie. Grupa o numerze n posiada n dodatkowych bitów i umożliwia kodowanie liczb z przedziału od $-2^n + 1$ do -2^{n-1} oraz od 2^{n-1} do $2^n - 1$. Przy takiej strukturze danych dopuszcza się tabelę kodową zawierającą maksymalnie 16 grup. Proponowaną przez JPEG uniwersalną tabelę kodu Huffmana dla różnic współczynników luminancji *DC* przedstawia tabela 8.1 [21]. Przykładowa tabela kodowa oferowana przez twórców schematu JPEG, powstała na podstawie pomiarów wielu testowych obrazów. Podczas badań autor rozprawy przygotował do testów własny zestaw 12 obrazów (ośmiobitowa głębia luminancji – 256 odcieni szarości), na podstawie których stworzone zostały dwie tabele (dla standardowej wartości dzielnika kwantyzacji elementów *DC* równego 16): dla binarnego kodu Huffmana oraz mieszanego kodu Huffmana (patrz tabela 8.2). Są to zarówno standardowe obrazy testowe (*airfield*, *airplane*, *boats*, *brigde*, *camera*, *couple*, *crowd*, *lena*, *man*, *peppers*, *woman1*) wykorzystywane np. w pracy [52] jak i zdjęcie z aparatu cyfrowego (*justyna* – ze zbiorów własnych). Ponieważ uniwersalna tabela nie może być dla każdego obrazu optymalna, więc przy jej użyciu podczas testów otrzymano średnią długość słowa kodowego $L_2 = 2.91054$, co dało efektywność kompresji zaledwie $E_2 = 92.47508\%$. Dla

testowych obrazów można zaproponować tabelę kodu binarnego oraz mieszanego (binarno-ternarnego), w której pojawiają się elementy tylko z pierwszych ośmiu grup.

Tabela 8.1 Przykładowa tabela kodowa dla współczynników DC proponowana przez JPEG

Nr grupy	Kod Huffmana
0	00
1	010
2	011
3	100
4	101
5	110
6	1110
7	11110
8	111110
9	1111110
10	11111110
11	111111110

Tabela 8.2 Proponowane tabele kodu Huffmana

Nr grupy	Prawdopodobieństwo	Kod binarny	Kod mieszany
0	0.128596	100	0 ₂ 1 ₂ 0 ₂
1	0.185490	111	1 ₂ 1 ₃
2	0.198677	01	0 ₂ 0 ₂
3	0.193337	00	1 ₂ 2 ₃
4	0.156224	110	1 ₂ 0 ₃
5	0.100274	1011	0 ₂ 1 ₂ 1 ₂ 1 ₂
6	0.035668	10101	0 ₂ 1 ₂ 1 ₂ 0 ₂ 1 ₂
7	0.001735	10100	0 ₂ 1 ₂ 1 ₂ 0 ₂ 0 ₂

Dla proponowanego binarnego kodu Huffmana otrzymano średnią długość słowa kodowego $L_2 = 2.78306$, co dało efektywność kompresji $E_2 = 96.71087\%$. Dla kodu binarno-ternarnego (klasa $r_0 = 2, r_1 = 3, q_1 = 65$) otrzymano $L_{2,3} = 2.75455$, co dało procentową efektywność $E_{2,3} = 97.71196\%$. Dodatkowo po zastosowaniu kodu dwufazowego do kompresji bufora ternarnego uzyskano zmniejszenie średniej długości słowa kodowego do $\underline{L}_{2,3} = 2.75440$, co dało dalszy wzrost efektywności do $\underline{E}_{2,3} = 97.77173\%$. Uzyskanie większych efektywności jest trudne ze względu na małą liczbę symboli źródła (zaledwie 8 grup kodowych). Powyższe wyniki zostały opublikowane w [77] wraz z propozycją algorytmu konwersji z liczby B do bloku q cyfr ternarnych (patrz punkt 8.3).

8.2. Zastosowanie kodu binarno-ternarnego do bezstratnej kompresji obrazów

Często zachodzi potrzeba bezstratnej archiwizacji obrazów (bez jakiejkolwiek utraty jakości), np. w przypadku obrazów medycznych (badania USG itp.) [66]. W takich przypadkach stosuje się metody modelowania danych takie jak predykcja liniowa, pozwalające uzyskać wzrost efektywności kompresji [15, 17, 18, 37]. Dzięki zastosowaniu predykcji liniowej można usunąć część zależności między sąsiednimi pikselami otrzymując źródło o rozkładzie Laplace'a, co powoduje wzrost wydajności kompresji uzyskanej metodami entropijnymi (traktowanie źródła jako zbiór symboli niezależnych) [67]. Pozwala to otrzymać stopień kompresji (stosunek długości pliku przed kompresją do długości pliku po kompresji) na poziomie od ok. 1.5:1 do 3:1. Podczas badań autora rozprawy nad nowymi metodami uzyskiwania współczynników predykcji z wykorzystaniem algorytmów genetycznych zostały wykonane testy z użyciem prostego suboptymalnego kodera binarno-ternarnego kodu Huffmana ($r_0 = 2$, $r_1 = 3$, $q_1 = 5$). Predykcja liniowa zaledwie trzeciego rzędu w połączeniu z kodem binarno-ternarnym (BT-Huff) pozwoliła uzyskać lepsze rezultaty, niż w przypadku użycia takich popularnych formatów graficznych jak GIF oraz PNG. Jeszcze wyższy stopień kompresji możemy uzyskać dzięki użyciu dynamicznego kodera arytmetycznego (ze względu na niepełne usunięcie przez predykcję liniową zależności między sąsiednimi pikselami), a także dzięki zwiększeniu rzędu predykcji, co ma miejsce w przypadku metody CALIC (jest to metoda o wysokim stopniu skomplikowania algorytmu kodującego, często wykorzystywana do porównań efektywności innych metod). Wyniki badań dla czterech standardowych obrazów testowych (o rozmiarach 512 na 512 pikseli – 256 odcieni szarości) przedstawiające średnią liczbę bitów na piksel, które zamieszczone są w tabeli 8.3, zostały opublikowane w [81]. Mimo iż są to wyniki uwzględniające długość pliku wraz z nagłówkiem, to procentowa efektywność kompresji z użyciem kodu mieszanego w każdym z testów przekroczyła $E_{2,3} = 99.2\%$. Podczas testów wzięto też pod uwagę koder korzystający z tablicy 8.1 i zasady kompresji elementów *DC* w algorytmie JPEG, który nie potrzebował zapisu tej tabeli w nagłówku pliku wynikowego uznając ją za standardową.

Tabela 8.3 Średnia liczba bitów na piksel dla czterech standardowych obrazów testowych

Testowany plik	GIF	PNG	JPEG-DC	Predyktor + BT-Huff	Predyktor + kod arytmetyczny	CALIC
Lena	8.077	4.611	5.037	4.483	4.457	4.121
Airplane	6.547	4.275	4.720	4.258	4.172	3.743
Peppers	8.236	4.908	5.407	4.829	4.799	4.425
Woman1	6.938	4.983	5.396	4.987	4.951	4.555

8.3. Algorytm szybkiego konwertera binarno–ternarnego

Jak wynika ze wzoru (3.5) konwersja liczby o q cyfrach ternarnych do postaci liczby B wymaga $q - 1$ mnożeń i dodawań. Rozpatrzmy przykład dla $q = 41$, wówczas potrzeba 40 mnożeń i dodawań, które w praktyce dają się łatwo zredukować do dodawań i przesunięć bitowych. Ma to duże znaczenie w przypadku sprzętowej realizacji algorytmu, np. w zmodyfikowanej wersji kodera JPEG opisanej w punkcie 8.1. Trudniej jest dokonać konwersji odwrotnej, gdyż podstawowy algorytm wymaga aż 40 dzieleni liczby o rozmiarze 65 bitów, co oznacza (pomijając nawet czas innych potrzebnych operacji) wprowadzenie dużego opóźnienia przy dekodowaniu. Poniżej zostanie zaprezentowany nie wymagający dzielenia algorytm opracowany przez autora rozprawy i opublikowany w pracy [77]. Tablica $wynik[i]$ zawiera po zdekodowaniu liczby B zestaw $ile_cyfr = q$ cyfr ternarnych, natomiast tablica $potęga[i] = 3^i$ jest statyczną tablicą zawierającą kolejne potęgi trójki (obie tablice indeksowane są od zera):

Algorytm konwersji z liczby B do bloku q cyfr ternarnych:

Wyzeruj tablicę $wynik[i]$;

Dla każdego i od ile_cyfr-1 do 1 z krokiem -1

```
{ Jeśli ( (B - potęga[i]) >= 0 ) wtedy:
    { B = B - potęga[i];
      wynik[i] = 1;
      Jeśli ( (B - potęga[i]) >= 0 ) wtedy:
          { B = B - potęga[i];
            wynik[i] = 2;
          }
    }
}
```

wynik[0] = B;

Algorytm zastępuje dzielenie liczby B przez 3 zestawem odejmowań kolejnych potęg trójki i porównań znaku wyniku odejmowania. Liczba operacji arytmetycznych jest zależna od wartości konkretnych cyfr ternarych, ale maksymalnie potrzeba $2 \cdot (q - 1)$ odejmowań, a także $2 \cdot (q - 1) + 1$ przypisań liczby dwubitowej. Co oznacza dla naszego przykładu ($q = 41$) maksymalnie 80 odejmowań oraz 81 przypisań liczby dwubitowej. Dodatkowo wymagane jest przechowywanie w statycznej tablicy *potęga[i]* (np. w pamięci ROM) 41 kolejnych potęg trójki (liczb 65 bitowych). Algorytm cechuje prostota (mająca duże znaczenie w przypadku realizacji sprzętowej) oraz mały nakład obliczeniowy i małe opóźnienia, mimo operowania na liczbach 65 bitowych. W ogólności dla wartości r algorytm wymaga wykonania maksymalnie $r - 1$ odejmowań dla zdekodowania jednej cyfry r -narnej. Z punktu widzenia szybkości działania dla wyższych wartości r zastąpienie jednego dzielenia wieloma odejmowaniami może okazać się nieopłacalne.

Istnienie tego algorytmu uzasadnia stwierdzenie, że zaproponowane w rozprawie metody podnoszenia efektywności kompresji w przeciwieństwie np. do techniki rozszerzonych binarnych kodów Huffmana nie wymagają dużych nakładów obliczeniowych i sprzętowych.

8.4. Zastosowanie kodów mieszanych do zwiększenia efektywności kodu Golomba

W tym punkcie zostaną przedstawione własności podstawowego kodu Golomba (służącego między innymi do kompresji źródeł binarnych) wraz z jego rozszerzeniem. Autor rozprawy proponuje modyfikację kodu Golomba do kodu dwubuforowego r -narnobinarnego, który w wielu przypadkach prowadzi do zwiększenia efektywności tego kodu.

8.4.1. Własności podstawowego kodu Golomba

W 1966 roku Solomon W. Golomb w pracy [33] przedstawił podstawowe założenia i przykłady rodziny kodów, które przyjęto określać mianem kodów Golomba. Ze względów historycznych w tym punkcie używane będą po części oznaczenia zgodne z publikacją [33]. Kod ten służy do reprezentacji nieujemnych liczb całkowitych n , których prawdopodobieństwo wystąpienia jest zgodne z rozkładem geometrycznym $G(n)$:

$$G(n) = (1 - p) \cdot p^n \quad (8.1)$$

Jest to kod przedrostkowy dla źródła o nieskończonym alfabecie symboli (wartości n). W praktyce służy on do kodowania źródła binarnego, dla którego p oznacza prawdopodobieństwo wystąpienia bicia 0.

bieństwo wystąpienia bitu bardziej prawdopodobnego: $p = \max\{p(0), p(1)\} \geq 0.5$. Przyjmijmy, że $p = p(0)$, wówczas słowo kodu Golomba dla wartości n reprezentuje ciąg (symbol) składający się z n zer i jednej jedynki. Tak zdefiniowane źródło jest jednocześnie pewnym kodem przedrostkowym. Główną zaletą tego kodu jest to, że nie wymaga on używania tablic kodowych i jest stosunkowo prosty do implementacji sprzętowej. Słowo kodu Golomba składa się z dwóch części: przedrostka będącego numerem grupy u (w postaci kodu unarnego) oraz z kodu dwufazowego, za pomocą którego koduje się numer elementu v w danej grupie. Grupa o numerze u składa się z m elementów alfabetu źródła o kolejnych numerach od $u \cdot m$ do $u \cdot m + m - 1$ i przyjmuje się, że w przybliżeniu są one równoprawdopodobne. Liczba grupowa m została dobrana zgodnie z dopasowaniem do kodu unarnego, czyli $p^m = \frac{1}{2}$. Oznacza to, że wynosi ona:

$$m = -\frac{\log_{10} 2}{\log_{10} p} = \log_p \frac{1}{2} \quad (8.2)$$

Analiza redundancji dla m będących liczbą całkowitą została przeprowadzona w pracy [70]. Gdy ze wzoru (8.2) nie otrzymamy liczby całkowitej, to aby liczba m była wartością całkowitą, Golomb zaproponował jej zaokrąglanie do $\lfloor m \rfloor$ lub $\lfloor m \rfloor + 1$. Bardziej precyzyjnie liczbę elementów w grupie (liczbę grupową) określa wzór zaprezentowany w pracy [6]:

$$m = \left\lceil -\frac{\log_{10}(1+p)}{\log_{10} p} \right\rceil \quad (8.3)$$

W tym przypadku korzystając z własności logarytmów możemy użyć dowolnej (ale wspólnej) podstawy dla obu logarytmów. Tworząc słowo kodu Golomba dla liczby n wyznaczamy najpierw numer grupy w postaci:

$$u = \left\lfloor \frac{n}{m} \right\rfloor \quad (8.4)$$

a następnie numer elementu w grupie:

$$v = n - u \cdot m \quad (8.5)$$

Liczba v kodowana jest przy pomocy kodu dwufazowego (patrz punkt 2.3). Oznacza to, że w każdej grupie pierwszych $l = 2^k - m$ wartości kodowanych jest przy użyciu $\lfloor \log_2 m \rfloor$ bitów, natomiast pozostałych $m - l$ jest kodowanych w postaci liczby $v + 2^k - m$ przy pomocy $\lceil \log_2 m \rceil$ bitów, gdzie $k = \lceil \log_2 m \rceil$ [62]. Średnia długość binarna słowa kodowego L_G oznacza liczbę bitów potrzebną do zakodowania jednego bitu źródła binarnego (a nie

średnią długość symboli kodu Golomba). Możemy ją wyznaczyć dzieląc liczby n na dwa przedziały. Pierwszy przedział to wartości $0 \leq n < l$, dla którego słowa kodu Golomba są najkrótsze i mają długość k bitów. Drugi przedział dla $n \geq l$, dla którego słowa kodowe mają długość $k + 1 + u'$ bitów, gdzie $u' = \left\lfloor \frac{n-l}{m} \right\rfloor$, składa się z kolejnych m elementowych grup słów kodu Golomba o długości odpowiednio $k + 1, k + 2, \dots$ bitów. Każde słowo w postaci kodu Golomba dla liczby n reprezentuje $n + 1$ bitów źródła. Korzystając ze wzoru (2.19) możemy dla m nie będących potęgą dwójki przedstawić L_G w następującej postaci:

$$L_G = \frac{\sum_{n=0}^{l-1} k \cdot (1-p) \cdot p^n + \sum_{n=l}^{\infty} (k+1+u') \cdot (1-p) \cdot p^n}{\sum_{n=0}^{\infty} (n+1) \cdot (1-p) \cdot p^n} \quad (8.6)$$

Dla wartości m będących potęgą dwójki wartość l wynosi zero i wówczas pierwsza suma w liczniku wzoru (8.6) nie występuje (jest to tzw. kod Rice'a [58, 63]). Nieskończone sumy należy przekształcić do prostszej postaci. Po uproszczeniu wzoru (8.6) otrzymujemy [50]:

$$L_G = (1-p) \cdot \left(k + \frac{p^l}{1-p^m} \right) \quad (8.7)$$

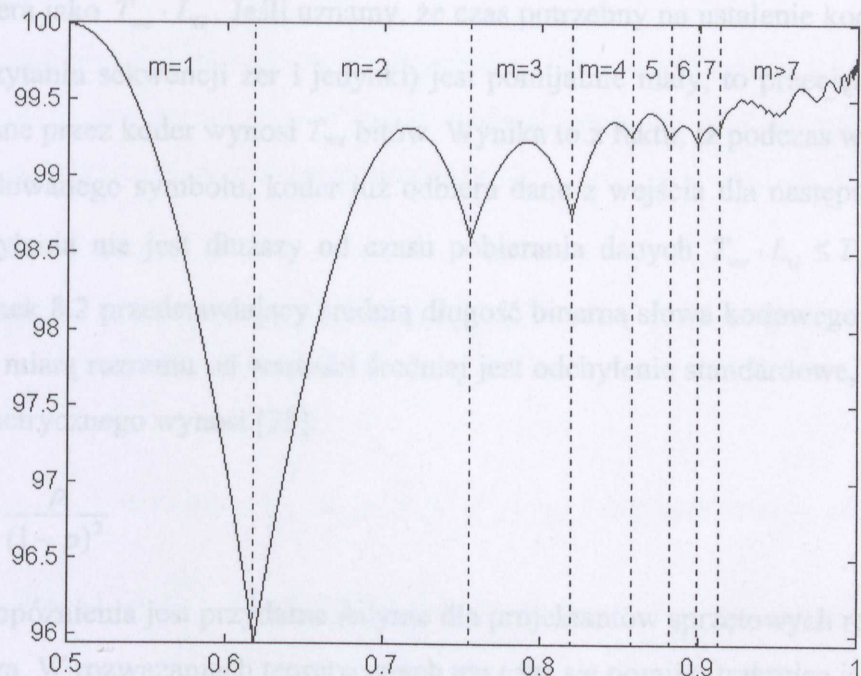
Co daje procentową efektywność kodu Golomba równą:

$$E_G = \frac{H_{01}}{L_G} \cdot 100\% = \frac{-p \cdot \log_2 p - (1-p) \cdot \log_2 (1-p)}{L_G} \cdot 100\% \quad (8.8)$$

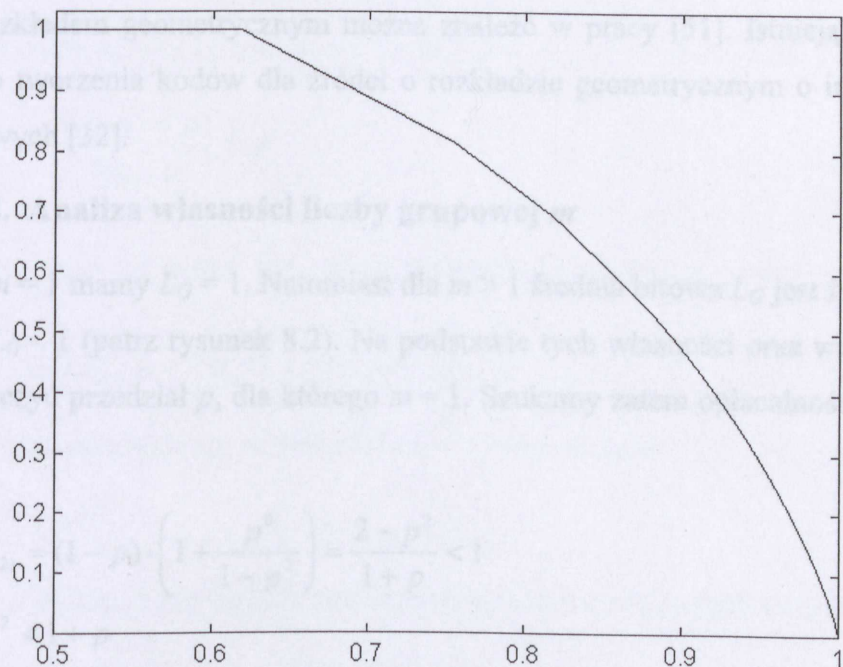
Ponieważ końcowy wzór na L_G jest często podawany w literaturze nieprawidłowo [26, 70, 72] (lub odnosi się wyłącznie do kodu Rice'a), to wyprowadzenie wzoru (8.7) zostanie umieszczone w dodatku A. Rysunek 8.1 przedstawia zależność procentowej efektywności kodu Golomba w funkcji prawdopodobieństwa p . Jak widać lokalne wartości minimalne oznaczają przejście (oznaczone pionowymi przerywanymi liniami) między kolejnymi liczbami grupowymi m . Aby zachować czytelność rysunku dla $m > 7$ ($p > 0.9115$) linie przerywane nie zostały umieszczone.

W przypadku transmisji strumienia bitów poddanego kodowaniu Golomba zachodzi opóźnienie czasowe wynikające z zasady działania kodera. Wczytuje on z wejścia bity o wartości „0” do momentu napotkania pierwszej „1”. Dopiero wtedy wysyła na wyjście odpowiedni kod. Średnia liczba bitów T_{we} to wartość oczekiwana liczby zer poprzedzających jedynekę w sygnale źródłowym [53]:

$$T_{we} = \sum_{n=0}^{\infty} (n+1) \cdot (1-p) \cdot p^n = \frac{1}{1-p} \quad (8.9)$$



Rysunek 8.1 Zależność procentowej efektywności kodu Golomba w funkcji p



Rysunek 8.2 Zależność średniej L_G kodu Golomba w funkcji p

Wyprowadzenie wzoru (8.9) znajduje się w dodatku A.

Jeśli przyjmiemy, że koder przyjmuje i wysyła dane w sposób szeregowy, to należy jeszcze wyznaczyć średnią liczbę bitów (średnią długość symboli kodu Golomba) na wyjściu dekodera jako $T_{we} \cdot L_G$. Jeśli uznamy, że czas potrzebny na ustalenie kodu wyjściowego (po wczytaniu sekwencji zer i jedynek) jest pomijalnie mały, to przeciętne opóźnienie wprowadzane przez koder wynosi T_{we} bitów. Wynika to z faktu, iż podczas wysyłania aktualnie zakodowanego symbolu, koder już odbiera dane z wejścia dla następnego symbolu, a czas wysyłania nie jest dłuższy od czasu pobierania danych $T_{we} \cdot L_G \leq T_{we}$, bo $L_G \leq 1$ (patrz rysunek 8.2 przedstawiający średnią długość binarną słowa kodowego L_G). Możemy przyjąć, że miarą rozrzutu od wartości średniej jest odchylenie standardowe, które dla rozkładu geometrycznego wynosi [25]:

$$\sigma = \frac{p}{(1-p)^2} \quad (8.10)$$

Ta analiza opóźnienia jest przydatna jedynie dla projektantów sprzętowych realizacji kodera i dekodera. W rozważaniach teoretycznych ten czas się pomija (traktując jako minimalny czas niezbędny do prawidłowej zamiany pojedynczego symbolu z wejścia na słowo kodu), gdyż przyjmuje się, że kod przedrostkowy jest kodem bez opóźnień.

Szczegółową analizę zastosowania rozszerzeń kodów Golomba dla źródeł z dwustronnym rozkładem geometrycznym można znaleźć w pracy [51]. Istnieją także nowsze podejścia do tworzenia kodów dla źródeł o rozkładzie geometrycznym o innej strukturze drzew kodowych [32].

8.4.2. Analiza własności liczby grupowej m

Dla $m = 1$ mamy $L_G = 1$. Natomiast dla $m > 1$ średnia bitowa L_G jest funkcją malejącą, taką że $L_G < 1$ (patrz rysunek 8.2). Na podstawie tych własności oraz wzoru (8.7) możemy wyznaczyć przedział p , dla którego $m = 1$. Szukamy zatem opłacalności kodu $L_G < 1$ dla $m = 2$.

$$L_{G(m=2)} = (1-p) \cdot \left(1 + \frac{p^0}{1-p^2}\right) = \frac{2-p^2}{1+p} < 1 \quad (8.11)$$

$$2 - p^2 < 1 + p$$

$$p^2 + p - 1 > 0 \quad (8.12)$$

Dla $\frac{1}{2} \leq p \leq 1$ rozwiązaniem tej nierówności jest $p > \frac{\sqrt{5}-1}{2} = 0.61803$, wówczas $m = 2$

staje się wartością optymalną. A zatem $m = 1$ jest optymalne dla $\frac{1}{2} \leq p < \frac{\sqrt{5}-1}{2}$, lecz dla

takiego zakresu prawdopodobieństw statystycznie nie uzyskujemy zysku z kompresji.

Kolejny krok, to znalezienie granicy między opłacalnością użycia kodu z wartością $m = 2$ oraz $m = 3$, polegające na rozwiązaniu nierówności:

$$L_{G(m=3)} < L_{G(m=2)}$$

$$(1-p) \cdot \left(2 + \frac{p}{1-p^3} \right) = \frac{2-2 \cdot p^3+p}{1+p+p^2} < \frac{2-p^2}{1+p}$$

$$0 < p^4 + p^3 - p$$

Co przy $\frac{1}{2} \leq p \leq 1$ upraszcza się do $p^3 + p^2 - 1 > 0$. Rozwiązaniem tej nierówności jest

$$p > \frac{1}{3} \cdot \left(\sqrt[3]{\frac{25-\sqrt{621}}{2}} + \sqrt[3]{\frac{25+\sqrt{621}}{2}} - 1 \right) = 0.75488, \text{ wówczas } m = 3 \text{ staje się wartością optymalną}$$

[65, 87]. Czyli $m = 2$ jest optymalne dla $\frac{\sqrt{5}-1}{2} < p \leq \frac{1}{3} \cdot \left(\sqrt[3]{\frac{25-\sqrt{621}}{2}} + \sqrt[3]{\frac{25+\sqrt{621}}{2}} - 1 \right)$.

Możemy także wyznaczyć przedziały optymalności m na podstawie równania (8.3). Z własności $\lceil x \rceil = m$ dla dowolnej liczby x spełniona jest nierówność $m-1 < x \leq m$, zatem w ten sposób możemy uzyskać progową wartość p w oparciu o wzór (8.3):

$$m-1 < -\frac{\log_{10}(1+p)}{\log_{10} p} \leq m$$

Ponieważ dla $\frac{1}{2} \leq p \leq 1$ mamy $\log_{10} p < 0$, to mnożąc obustronnie przez $-\log_{10} p$:

$$-(m-1) \cdot \log_{10} p < \log_{10}(1+p) \leq -m \cdot \log_{10} p$$

Po uproszczeniu otrzymujemy nierówność $(m+1)$ -tego stopnia:

$$1 \geq p^m + p^{m+1} \quad (8.11)$$

A także drugie ograniczenie będące rozwiązaniem nierówności m -tego stopnia:

$$p^m + p^{m-1} - 1 > 0 \quad (8.12)$$

Czyli jest to dokładnie taka sama zależność, jaką uzyskano przy wcześniejszych porównaniach stosując wzór (8.7) dla konkretnych wartości m . Nierówności (8.11) i (8.12) zapre-

zentował w 1975 roku Gallager w pracy [26] wraz z uzasadnieniem optymalności parametru m dla danego prawdopodobieństwa p :

$$p^m + p^{m+1} < 1 \leq p^m + p^{m-1} \tag{8.13}$$

Poniżej znajduje się przykład przedstawiony przez Golomba w pracy [33].

Przykład 8.1

Dane jest źródło binarne, w którym $p = p(0) = 0.95$. Wówczas $m = 14$, $k = 4$, $l = 2$. Tabela 8.4 przedstawia pierwsze 32 symbole kodu Golomba dla $m = 14$. Średnia liczba potrzebna do zakodowania jednego bitu źródła wynosi $L_G = 0.288079$, co daje efektywność $E_G = 99.41617\%$. Natomiast średnie opóźnienie kodera $T_{we} = 20$ bitów (przy odchyleniu standardowym równym $\sigma = 19.49359$).

Tabela 8.4 Kod Golomba z przykładu 8.1

n	kod Golomba	n	kod Golomba
0	0000	16	100100
1	0001	17	100101
2	00100	18	100110
3	00101	19	100111
4	00110	20	101000
5	00111	21	101001
6	01000	22	101010
7	01001	23	101011
8	01010	24	101100
9	01011	25	101101
10	01100	26	101110
11	01101	27	101111
12	01110	28	110000
13	01111	29	110001
14	10000	30	1100100
15	10001	31	1100101

8.4.3. Modyfikacja kodu Golomba

Zaproponowane w rozdziale 4 podejście pozwalające korzystać jednocześnie z różnych buforów r -narych kody r -narne prowadziło w większości przypadków do wzrostu efektywności kodu Huffmana. Podobne podejście możemy zastosować dla kodu Golomba. Propozycja użycia wielowartościowego kodu dla źródeł o nieskończonej liczbie symboli

była rozważana teoretycznie między innymi w pracy [40] (bez realizacji praktycznej dla komputerów opartych o binarny zapis liczb).

Dla niektórych wartości p możliwe jest utworzenie zmodyfikowanego kodu Golomba cechującego się wyższą efektywnością kompresji źródeł binarnych w porównaniu do podstawowego kodu Golomba. Symbol takiego kodu różni się jedynie w jego drugiej części (przedrostek unarny pozostaje bez zmian). Możemy przyjąć, że każda grupa składająca się z t elementów (traktowanych jako symbole prawie równoprawdopodobne) nie jest kodowana przy użyciu kodu dwufazowego lecz przy użyciu kodu mieszanego dla t symboli równoprawdopodobnych (wprowadzenie oznaczenia t w miejsce proponowanej przez Golomba wartości m jest istotne dla odróżnienia dwóch sytuacji i zostanie to wyjaśnione w dalszej części tego punktu). Każdą dodatnią liczbę całkowitą można przedstawić przy pomocy wzoru:

$$t = 2^s \cdot r \quad (8.14)$$

w postaci grupowej liczby t , gdzie s jest nieujemną liczbą całkowitą, natomiast r dodatnią liczbą nieparzystą. Na przykład dla $t = 20$ mamy $s = 2$ oraz $r = 5$. Mając dane prawdopodobieństwo p na podstawie wzoru (8.3) wyznaczamy wartość m . Jeśli m nie jest potęgą dwójki, to rozpatrujemy dwie sytuacje: dla $t = m$ oraz osobno dla $t = m - 1$, dlatego należy rozróżniać wartości t oraz m . Dla obu wartości t na podstawie wzoru:

$$L_{\underline{G}} = (1 - p) \cdot \left(\bar{b}_q(r) + s + 1 + \frac{p'}{1 - p'} \right) \quad (8.15)$$

wyznaczamy średnią liczbę bitów $L_{\underline{G}}$ potrzebną do zakodowania jednego bitu źródła (wprowadzenie wzoru (8.15) znajduje się w dodatku A). Następnie wybieramy mniejszą z dwóch wartości $L_{\underline{G}}$ i porównujemy ją z wartością L_G wyliczoną ze wzoru (8.7). Jeśli $L_{\underline{G}} \geq L_G$, to używamy podstawowego kodu Golomba. W przeciwnym wypadku, gdy:

$$\bar{b}_q(r) + s + 1 + \frac{p'}{1 - p'} < k + \frac{p'}{1 - p^m} \quad (8.16)$$

mamy do czynienia z modyfikacją kodu Golomba. Kod reprezentujący liczbę n składa się w pierwszej części z numeru u grupy przedstawionego w postaci przedrostka unarnego:

$$u = \left\lfloor \frac{n}{t} \right\rfloor \quad (8.17)$$

Gdy $s > 0$, to po przedrostku unarnym znajduje się środkowa część kodu będąca s bitową reprezentacją liczby t w postaci:

$$z = \left\lfloor \frac{n \bmod t}{r} \right\rfloor \quad (8.18)$$

Końcowa część symbolu zmodyfikowanego kodu Golomba składa się z r -narnej reprezentacji cyfry v danej wzorem:

$$v = n \bmod r \quad (8.19)$$

W praktycznych zastosowaniach należy dobrać odpowiednią wzorcową wartość q dla danej liczby r . Wówczas koder umieszcza kody wynikowe w dwóch buforach. W buforze binarnym zapisuje unarne reprezentacje przedrostków u oraz s bitowe reprezentacje liczb z . Natomiast w q elementowym buforze r -narnym zapisywane są kolejne cyfry v , aż do jego zapelnienia. Po zapelnieniu bufora r -narnego obliczana jest zgodnie ze wzorem (3.5) liczba B , a następnie zawartość obu buforów jest wysyłana na wyjście. A zatem opóźnienie kodera wynosi $q - 1$ symboli zmodyfikowanego kodu Golomba względem jego podstawowej wersji. Oznacza to, że zmodyfikowany koder musi wczytać z wejścia średnio o $(q - 1) \cdot T_{we}$ bitów więcej, niż podstawowy koder Golomba, gdzie T_{we} to wartość oczekiwana liczby zer poprzedzających jedynekę w sygnale źródłowym. Na podstawie wzoru (8.9) średnie opóźnienie kodera wynosi $T_{we(q)}$ bitów odczytanych z wejścia:

$$T_{we(q)} = \frac{q - 1}{1 - p} \quad (8.20)$$

8.4.4. Przykłady zmodyfikowanego kodu Golomba

Przeanalizujemy sytuację dla $t = m = 3$, wtedy $s = 0$ oraz $r = 3$. Wówczas dla $q = 41$ nierówność (8.16) przyjmuje postać:

$$\begin{aligned} \frac{65}{41} + 0 + 1 + \frac{p^3}{1 - p^3} &< 2 + \frac{p^1}{1 - p^3} \\ \frac{24}{41} &< \frac{p - p^3}{1 - p^3} \end{aligned}$$

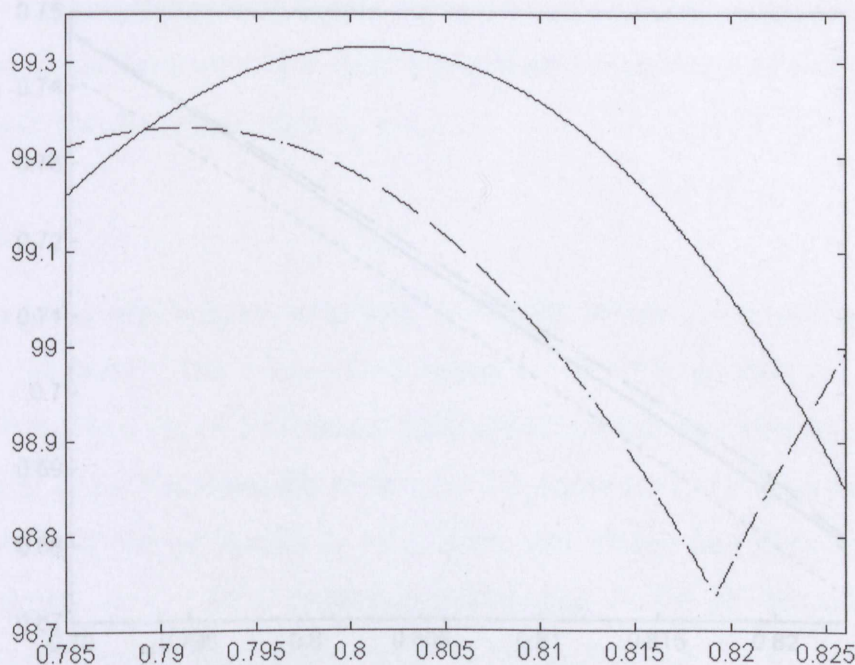
Po uproszczeniu otrzymujemy nierówność kwadratową:

$$17 \cdot p^2 + 17 \cdot p - 24 > 0$$

Dla $\frac{1}{2} \leq p \leq 1$ rozwiązaniem nierówności jest $p > \frac{1}{2} \cdot \left(\sqrt{\frac{113}{17}} - 1 \right) = 0.789095$. A zatem jest to dolna granica opłacalności zastosowania zmodyfikowanego kodu Golomba dla $t = 3$. Ponieważ dla wyższych wartości t rozwiązanie nierówności (8.16) w sposób analityczny

jest trudne, więc pozostałe przedstawione wyniki zostały wyznaczone numerycznie z dokładnością do 4 miejsc po przecinku. Podstawowy kod Golomba dla $m = 3$ jest stosowany w przedziale prawdopodobieństw $0.75488 < p < 0.8192$, natomiast opłacalność modyfikacji kodu Golomba przy $t = 3$ oraz $q = 41$ zachodzi dla $0.789095 < p < 0.8240$, czyli nawet przy $m = 4$. Dlatego właśnie przy doborze wartości t rozpatruje się dwa przypadki. Rysunek 8.3 przedstawia porównanie procentowej efektywności w funkcji p dla podstawowego kodu Golomba (linia przerywana) oraz zmodyfikowanego kodu Golomba (linia ciągła) dla parametrów $t = 3$ oraz $q = 41$. Natomiast rysunek 8.4 przedstawia średnią liczbę bitów potrzebną do zakodowania jednego bitu źródła L_G (linia przerywana), $L_{\underline{G}}$ dla $t = 3$ i $q = 41$ (linia ciągła) oraz entropię H_{01} źródła binarnego (linia kropkowana) w funkcji prawdopodobieństwa p . Np. dla $p = 0.8191$ zysk między zastosowaniem podstawowego i zmodyfikowanego kodu Golomba wynosi $L_G - L_{\underline{G}} = 0.00235744$ bita na symbol źródła binarnego. Wtedy procentowa efektywność dla kodu podstawowego wynosi $E_G = 98.73897\%$, natomiast dla zmodyfikowanego kodu Golomba wynosi

$$E_{\underline{G}} = \frac{H_{01}}{L_{\underline{G}}} \cdot 100\% = 99.07711\%.$$



Rysunek 8.3 Porównanie procentowej efektywności kodów Golomba w funkcji p .
Kod podstawowy (linia przerywana) i zmodyfikowany (linia ciągła)

Jeśli dodatkowo użyjemy kodu dwufazowego do kompresji liczby B , to aby obliczyć średnią bitową $L_{\underline{G}}$, wystarczy zamienić średnią liczbę bitów potrzebną do reprezentacji jednej cyfry r -narnej z $b_q(r)$ na $\underline{b}_q(r)$:

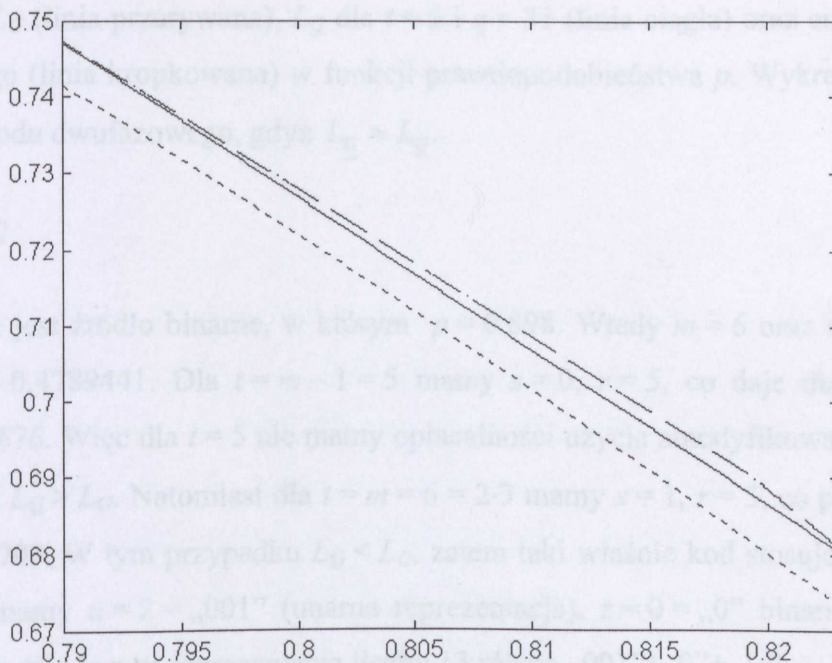
$$L_{\underline{G}} = (1 - p) \cdot \left(\underline{b}_q(r) + s + 1 + \frac{p^t}{1 - p^t} \right) \quad (8.21)$$

Wówczas procentowa efektywność dla zmodyfikowanego kodu Golomba (z użyciem kodu dwufazowego) dana jest wzorem:

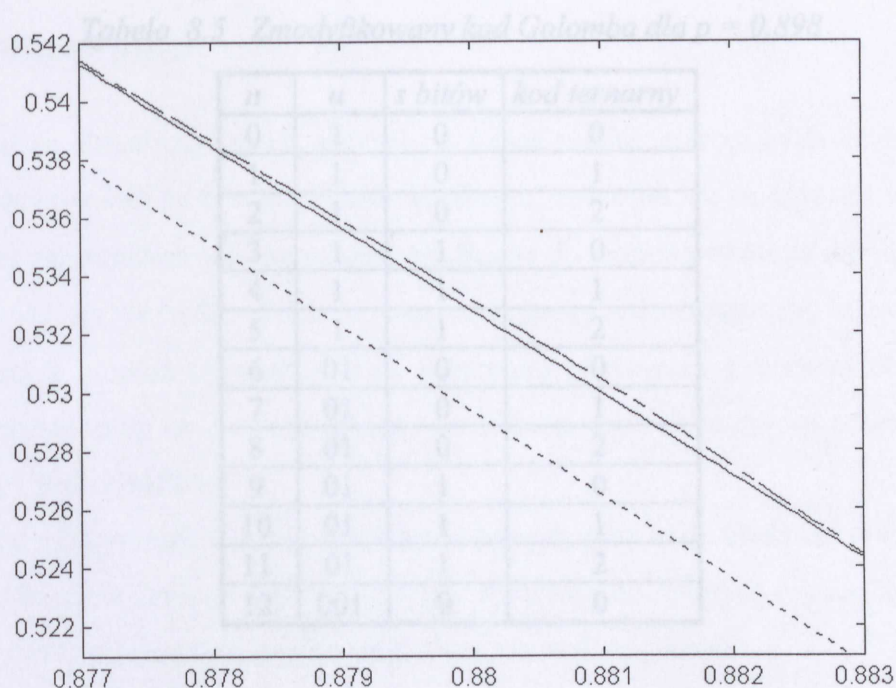
$$E_{\underline{G}} = \frac{H_{01}}{L_{\underline{G}}} \cdot 100\% = \frac{-p \cdot \log_2 p - (1 - p) \cdot \log_2 (1 - p)}{L_{\underline{G}}} \cdot 100\% \quad (8.22)$$

Co dla $p = 0.8191$ daje efektywność $E_{\underline{G}} = 99.08443\%$ oraz wagę zysku $W = 27.64803\%$.

Przeanalizujemy wartość $t = m = 5$, wówczas $s = 0$ oraz $r = 5$. Wartość $m = 5$ uzyskujemy, gdy $0.8566 < p < 0.8813$. Dla $r = 5$ oraz $q = 31$ opłacalność zmodyfikowanej metody uzyskujemy, gdy $0.8754 < p < 0.8835$, czyli nawet przy $m = 6$. Np. dla $p = 0.8812$ mamy $E_G = 99.25424\%$ oraz $E_{\underline{G}} = 99.33709\%$ [85].



Rysunek 8.4 Zależność L_G (linia przerywana), $L_{\underline{G}}$ dla $t = 3$, $q = 41$ (linia ciągła), H_{01} (linia kropkowana) w funkcji p



Rysunek 8.5 Zależność L_G (linia przerywana), $L_{\underline{G}}$ dla $t = 5$, $q = 31$ (linia ciągła), H_{01} (linia kropkowana) w funkcji p

Rysunek 8.5 przedstawia średnią liczbę bitów potrzebną do zakodowania jednego bitu źródła L_G (linia przerywana), $L_{\underline{G}}$ dla $t = 5$ i $q = 31$ (linia ciągła) oraz entropię H_{01} źródła binarnego (linia kropkowana) w funkcji prawdopodobieństwa p . Wykres nie uwzględnia użycia kodu dwufazowego, gdyż $L_{\underline{G}} \approx L_G$.

Przykład 8.2

Dane jest źródło binarne, w którym $p = 0.898$. Wtedy $m = 6$ oraz średnia długość bitowa $L_G = 0.4789441$. Dla $t = m - 1 = 5$ mamy $s = 0$, $r = 5$, co daje dla $q = 3$ średnią $L_{\underline{G}} = 0.4831676$. Więc dla $t = 5$ nie mamy opłacalności użycia zmodyfikowanego kodu Golomba, gdyż $L_{\underline{G}} > L_G$. Natomiast dla $t = m = 6 = 2 \cdot 3$ mamy $s = 1$, $r = 3$, co przy $q = 41$ daje $L_{\underline{G}} = 0.4781707$. W tym przypadku $L_{\underline{G}} < L_G$, zatem taki właśnie kod stosujemy. Wtedy np. dla $n = 13$ mamy $u = 2 = „001”$ (unarna reprezentacja), $z = 0 = „0”$ binarnie, $v = 1 = „1”$ ternarnie. Co daje łączną reprezentację liczby 13 równą „001”+„0”+„1₃”.

W tabeli 8.5 zawarte są słowa dla pierwszych $n \leq 12$ symboli zmodyfikowanego kodu Golomba. Na podstawie wzoru (8.20) dla $p = 0.898$ otrzymujemy średnie opóźnienie kodera $T_{we(41)} = 392.15686$ bita odczytanych z wejścia.

Tabela 8.5 Zmodyfikowany kod Golomba dla $p = 0.898$

n	u	s bitów	kod ternarny
0	1	0	0
1	1	0	1
2	1	0	2
3	1	1	0
4	1	1	1
5	1	1	2
6	01	0	0
7	01	0	1
8	01	0	2
9	01	1	0
10	01	1	1
11	01	1	2
12	001	0	0

Z drugiej strony zaproponowany przez autora algorytm kompresji wykorzystujący mieszany kod Huffmanna ma następujące właściwości:

- efektywność mieszanego kodu Huffmanna jest w zdecydowanej większości przypadków wyższa od efektywności binarnego kodu Huffmanna (w szczególnym przypadku oba te kody są równoważne);
- algorytm charakteryzuje się dużą elastycznością implementacyjną i małymi wymaganiami pamięciowymi, podobnie jak ma to miejsce w przypadku binarnego kodu Huffmanna;
- dzięki odpowiedniej reprezentacji danych możliwe jest użycie wielu różnych elementów wartościowych (r -narych) w jednym kodzie, z zachowaniem wysokiej efektywności zapisu cyfr r -narych.

Wnioski te są zgodne z tezą niniejszej rozprawy. Oryginalny wkład własny autora w pracy stanowi:

- propozycja konstrukcji kodów mieszanych będących uogólnieniem kodu Huffmanna;
- rozwiązanie problemu efektywnego zapisu cyfr r -narych (nie tylko dwójki), co jest podstawą praktycznego użycia r -narych kodów Huffmanna (zaczynając od zapisu 9 cyfr r -narych, które następnie kompresowane są przez użycie kodu dwufazowego);
- propozycja efektywnej realizacji mieszanego kodu Huffmanna, co jest algorytmu pozwalającego na łączenie wielu różnych symboli r -narych w jednym kodzie w taki sposób, aby uzyskać kod o większej efektywności niż binarny kod Huffmanna.

9. Podsumowanie

Analiza aktualnego stanu wiedzy na temat kodów entropijnych cechujących się wyższą efektywnością od binarnego kodu Huffmana wykazała ich następujące wady:

- duże zapotrzebowanie na pamięć niezbędną do przechowywania dużych tablic kodowych (przez koder i dekodek rozszerzonego kodu Huffmana czy kodu Tunstalla);
- większy stopień skomplikowania algorytmu (zwłaszcza dekodera) dla kompresji arytmetycznej, co oznacza dłuższy czas kompresji i dekompresji względem binarnego kodu Huffmana;
- niska efektywność zapisu wielowartościowych słów kodu Huffmana (w odniesieniu do aktualnie użytkowanych architektur sprzętowych opartych o binarną reprezentację liczb) nie pozwalająca na praktyczne ich wykorzystanie.

Z drugiej strony zaproponowany przez autora algorytm kompresji wykorzystujący mieszany kod Huffmana ma następujące własności:

- efektywność mieszanego kodu Huffmana jest w zdecydowanej większości przypadków wyższa od efektywności binarnego kodu Huffmana (w najgorszym przypadku oba te kody są równoważne);
- algorytm charakteryzuje się niską złożonością implementacyjną i małymi wymaganiami pamięciowymi, podobnie jak ma to miejsce w przypadku binarnego kodu Huffmana;
- dzięki odpowiedniej reprezentacji danych możliwe jest użycie wielu różnych elementów wielowartościowych (r -narnych) w jednym kodzie, z zachowaniem wysokiej efektywności zapisu cyfr r -narnych.

Własności te są zgodne z tezą niniejszej rozprawy. Oryginalny **wkład własny** autora w pracy stanowi:

- propozycja konstrukcji kodów mieszanych będących uogólnieniem kodu Huffmana;
- rozwiązanie problemu efektywnego zapisu cyfr r -narnych (nie będących potęgą dwójki), co jest podstawą praktycznego użycia r -narnych kodów Huffmana (łączenie w bloki po q cyfr r -narnych, które następnie kompresowane są przy użyciu kodu dwufazowego);
- propozycja efektywnej realizacji mieszanego kodu Huffmana, to jest algorytmu pozwalającego na łączenie wielu różnych symboli r -narnych w jednym kodzie w taki sposób, aby uzyskać kod o większej efektywności niż binarny kod Huffmana;

- opisanie podstawowych zasad konstruowania drzewa optymalnego kodu mieszane-
go (przy danych ograniczeniach);
- wykonanie badań testowych potwierdzających wzrost efektywności uzyskany dzięki
zastosowaniu kodów mieszanych (w porównaniu do binarnego kodu Huffmana) za-
równo dla zbiorów generowanych w sposób pseudolosowy jak i dla rzeczywistych
plików z danymi o różnych rozkładach prawdopodobieństw;
- wyznaczenie wykładniczej złożoności konstrukcji podstawowego algorytmu poszu-
kiwania optymalnego mieszane-
go kodu Huffmana (metoda porównań) wraz z me-
todą analitycznego wyznaczania dokładnej liczby iteracji dla metody porównań;
- zdefiniowanie warunków, dla których możliwe jest zmniejszenie liczby iteracji po-
trzebnych do wyznaczenia optymalnego kodu Huffmana (metoda integralnych pod-
drzew oraz propozycja poszukiwania poddrzew integralnych binarnie);
- analiza optymalnego binarno-ternarnego kodu Huffmana będącego propozycją
zmniejszenia złożoności implementacyjnej kodu mieszane-
go;
- trzy propozycje użycia suboptymalnych kodów mieszanych o krótkim czasie kon-
strukcji drzewa kodowego:
 - suboptymalny kod dwupoziomowy;
 - metoda kolejnych uściśleń;
 - rozszerzony kod dwupoziomowy;
- analiza opóźnień wprowadzanych przez mieszany kod Huffmana i propozycja uży-
cia suboptymalnych kodów o stałym opóźnieniu;
- propozycja użycia optymalnych i suboptymalnych kodów o małym opóźnieniu;
- przedstawienie trzech metod kodowania wykorzystujących kody mieszane w celu
podniesienia efektywności dynamicznego kodu Huffmana;
- praktyczne zastosowanie kodu binarno-ternarnego jako modyfikacja przedrostko-
wego kodu Huffmana używanego przy kompresji elementów *DC* w algorytmie
JPEG;
- opracowanie algorytmu szybkiego konwertera binarno-ternarnego, który nie wyma-
ga wykonywania operacji dzielenia;
- propozycja użycia kodów mieszanych w celu podniesienia efektywności kodu Go-
lomba;
- analiza efektywności kodu dwufazowego wraz z przedstawieniem prostych algo-
rytmów kodera i dekodera;

- opracowanie algorytmu doboru parametrów wzorcowych q zapewniających wysoką efektywność zapisu cyfr r -narnych;
- przedstawienie dwóch algorytmów tworzenia krótkich informacji nagłówkowych dla statycznych metod entropijnych (opracowanych na podstawie odpowiednio dobranych informacji znanych z literatury);
- wykonanie oprogramowania w języku C opartego na poszczególnych zaproponowanych w pracy algorytmach, a także oprogramowania testującego działanie i umożliwiającego generowanie wykresów zaprezentowanych w pracy (w tym program generujący źródła o założonym przez użytkownika rozkładzie).

W pracy przedstawiono szczegółowy opis powyższych zagadnień poparty ponad dwudziestoma przykładami, wynikami testów oraz analizą matematyczną (wykorzystując do tego celu ponad sto wzorów). Wyniki badań zostały zaprezentowane w kilkunastu referatach na konferencjach o charakterze zarówno krajowym jak i międzynarodowym [68, 75, 76, 77, 78, 79, 81, 82, 83, 84, 85]. Oprócz samego zdefiniowania zasad działania algorytmu kompresji opartej na mieszanym kodzie Huffmana (zarówno w wersji statycznej jak i dynamicznej), autor przeanalizował także problemy związane ze znalezieniem optymalnego kodu mieszanego wraz z propozycjami ich rozwiązania. Jednak dla źródeł o dużej liczbie symboli (pomijając kody o z góry zdefiniowanych drzewach) przy obecnych możliwościach technicznych wprowadzenie kodu optymalnego w czasie rzeczywistym może okazać się niepraktyczne. Dlatego też autor przedstawił kilka propozycji konstruowania drzew suboptymalnego kodu mieszanego o krótkim czasie konstrukcji drzewa kodowego.

Dalsze kierunki badań:

Jak już wspomniano, istotnym problemem jest szybka generacja mieszanych kodów Huffmana dla bardzo dużych źródeł. Konieczne są tu poszukiwania dalszych algorytmów konstrukcji kodów suboptymalnych, a w miarę możliwości także optymalnych. Z praktycznego punktu widzenia duże znaczenie mają powiązania algorytmów z ich realizacją sprzętową, gdyż często od tego zależy, czy dana metoda będzie w przyszłości wykorzystywana w praktyce (np. w układach o małych wymaganiach energetycznych).

Dodatek A - Wyprowadzenia wzorów (8.7), (8.9) oraz (8.15)

Wyprowadzenie wzoru na średnią liczbę bitów L_G potrzebną do zakodowania jednego bitu źródła binarnego przy użyciu kodu Golomba opiera się na własnościach skończonego i nieskończonego ciągu geometrycznego.

$$L_G = \frac{\sum_{n=0}^{l-1} k \cdot (1-p) \cdot p^n + \sum_{n=l}^{\infty} (k+1+u') \cdot (1-p) \cdot p^n}{\sum_{n=0}^{\infty} (n+1) \cdot (1-p) \cdot p^n} = \frac{a+b}{T_{we}} \quad (\text{A.1})$$

Na podstawie oznaczeń wzoru (A.1) możemy go podzielić na trzy człony:

$$a = k \cdot (1-p) \cdot \sum_{n=0}^{l-1} p^n \quad (\text{A.2})$$

$$b = (1-p) \cdot \sum_{n=l}^{\infty} (k+1+u') \cdot p^n \quad (\text{A.3})$$

$$T_{we} = (1-p) \cdot \sum_{n=0}^{\infty} (n+1) \cdot p^n \quad (\text{A.4})$$

Gdzie $u' = \left\lfloor \frac{n-l}{m} \right\rfloor$, natomiast wartość $l = 2^k - m$ oznacza liczbę pierwszych symboli o najkrótszej długości słowa kodu Golomba równej $k = \lceil \log_2 m \rceil$. Kolejne m elementowe grupy słów kodu Golomba mają długości odpowiednio $k+1, k+2, \dots$ bitów. Dla wartości m będących potęgą dwójki wartość l wynosi zero i wówczas pierwsza suma (oznaczona dalej jako a) w liczniku wzoru (A.1) nie występuje. Jeżeli suma l wyrazów skończonego ciągu geometrycznego wynosi:

$$\sum_{n=0}^{l-1} p^n = p^0 + p^1 + \dots + p^{l-1} = \frac{1-p^l}{1-p} \quad (\text{A.5})$$

To wzór (A.2) upraszcza się do:

$$a = k \cdot (1-p^l) \quad (\text{A.6})$$

Jeżeli $u' = \left\lfloor \frac{n-l}{m} \right\rfloor$, to upraszczając wzór (A.3) możemy zastosować podział na m elementowe sumy i zastąpić n przez $l + j \cdot m$. Wtedy można użyć podstawienia $u' = j$:

$$\begin{aligned}
b &= (1-p) \cdot \sum_{j=0}^{\infty} \left((k+1+j) \cdot \sum_{i=0}^{m-1} p^{(l+j \cdot m)+i} \right) = (1-p) \cdot \sum_{j=0}^{\infty} \left((k+1+j) \cdot p^l \cdot p^{j \cdot m} \cdot \sum_{i=0}^{m-1} p^i \right) \\
b &= (1-p) \cdot p^l \cdot \sum_{j=0}^{\infty} \left((k+1+j) \cdot p^{j \cdot m} \cdot \frac{1-p^m}{1-p} \right) = (1-p^m) \cdot p^l \cdot \sum_{j=0}^{\infty} ((k+1+j) \cdot p^{j \cdot m}) \\
b &= (1-p^m) \cdot p^l \cdot \left((k+1) \cdot \sum_{j=0}^{\infty} p^{j \cdot m} + \sum_{j=0}^{\infty} j \cdot p^{j \cdot m} \right)
\end{aligned}$$

Ale $\sum_{j=0}^{\infty} p^{j \cdot m} = p^0 + p^m + p^{2 \cdot m} + \dots = \frac{1}{1-p^m}$ oraz:

$$\begin{aligned}
\sum_{j=0}^{\infty} j \cdot p^{j \cdot m} &= 0 + p^m + 2 \cdot p^{2 \cdot m} + 3 \cdot p^{3 \cdot m} + \dots = \\
&= (p^m + p^{2 \cdot m} + p^{3 \cdot m} + \dots) + (p^{2 \cdot m} + p^{3 \cdot m} + p^{4 \cdot m} + \dots) + (p^{3 \cdot m} + p^{4 \cdot m} + p^{5 \cdot m} + \dots) + \dots = \\
&= \frac{p^m}{1-p^m} + \frac{p^{2 \cdot m}}{1-p^m} + \frac{p^{3 \cdot m}}{1-p^m} + \dots = \frac{p^m}{(1-p^m)^2}
\end{aligned}$$

A zatem wartość b upraszcza się do:

$$b = (1-p^m) \cdot p^l \cdot \left(\frac{k+1}{1-p^m} + \frac{p^m}{(1-p^m)^2} \right) = p^l \cdot \left(k+1 + \frac{p^m}{1-p^m} \right) \quad (\text{A.7})$$

Aby uprościć wzór (A.4), należy obliczyć sumę szeregu [8]:

$$\begin{aligned}
\sum_{n=0}^{\infty} (n+1) \cdot p^n &= p^0 + 2 \cdot p^1 + 3 \cdot p^2 + \dots = \\
&= (p^0 + p^1 + p^2 + \dots) + (p^1 + p^2 + p^3 + \dots) + (p^2 + p^3 + p^4 + \dots) + \dots = \\
&= \frac{p^0}{1-p} + \frac{p^1}{1-p} + \frac{p^2}{1-p} + \dots = \frac{1}{(1-p)^2}
\end{aligned}$$

Po podstawieniu do (A.4) otrzymujemy wartość oczekiwaną długości symboli kodu Golomba:

$$T_{we} = \frac{1}{1-p} \quad (\text{A.8})$$

Podstawiając do (A.1) wzory (A.6), (A.7) oraz (A.8):

$$L_G = (1-p) \cdot \left(k - k \cdot p^l + k \cdot p^l + p^l + \frac{p^{m+l}}{1-p^m} \right) = (1-p) \cdot \left(k + \frac{p^l - p^{m+l} + p^{m+l}}{1-p^m} \right)$$

W ten sposób otrzymujemy końcową postać L_G :

$$L_G = (1-p) \cdot \left(k + \frac{p^l}{1-p^m} \right) \quad (\text{A.9})$$

Uproszczenie wzoru dla zmodyfikowanego kodu Golomba przebiega w podobny sposób:

$$L_{\underline{G}} = \frac{\sum_{n=0}^{\infty} \left(\left(\lfloor \frac{n}{t} \rfloor + \bar{b}_q(r) + s + 1 \right) \cdot (1-p) \cdot p^n \right)}{\sum_{n=0}^{\infty} (n+1) \cdot (1-p) \cdot p^n} = \frac{c}{T_{we}} \quad (\text{A.10})$$

Mianownik jest zgodny ze wzorem (A.4) uproszczonym do postaci wzoru (A.8). Rozpatrzymy zatem licznik dzieląc na t elementowe grupy i wyliczając ich wewnętrzne sumy:

$$c = (1-p) \cdot \sum_{j=0}^{\infty} \left((j + \bar{b}_q(r) + s + 1) \cdot p^{jt} \cdot \sum_{i=0}^{t-1} p^i \right)$$

$$c = (1-p) \cdot \frac{1-p^t}{1-p} \cdot \sum_{j=0}^{\infty} \left((j + \bar{b}_q(r) + s + 1) \cdot p^{jt} \right)$$

$$c = (1-p^t) \cdot \left((\bar{b}_q(r) + s + 1) \sum_{j=0}^{\infty} p^{jt} + \sum_{j=0}^{\infty} j \cdot p^{jt} \right)$$

Przy czym poszczególne sumy upraszczają się do:

$$\sum_{j=0}^{\infty} p^{jt} = p^0 + p^t + p^{2t} + \dots = \frac{1}{1-p^t}$$

$$\sum_{j=0}^{\infty} j \cdot p^{jt} = 0 + p^t + 2 \cdot p^{2t} + 3 \cdot p^{3t} + \dots =$$

$$= (p^t + p^{2t} + p^{3t} + \dots) + (p^{2t} + p^{3t} + p^{4t} + \dots) + (p^{3t} + p^{4t} + p^{5t} + \dots) + \dots =$$

$$= \frac{p^t}{1-p^t} + \frac{p^{2t}}{1-p^t} + \frac{p^{3t}}{1-p^t} + \dots = \frac{p^t}{(1-p^t)^2}$$

I możemy je podstawić do licznika c :

$$c = (1-p^t) \cdot \left(\frac{\bar{b}_q(r) + s + 1}{1-p^t} + \frac{p^t}{(1-p^t)^2} \right) = \bar{b}_q(r) + s + 1 + \frac{p^t}{1-p^t}$$

Co daje uproszczenie wzoru (A.10) do końcowej postaci:

$$L_{\underline{G}} = (1-p) \cdot \left(\bar{b}_q(r) + s + 1 + \frac{p^t}{1-p^t} \right) \quad (\text{A.11})$$

Dodatek B - Kody źródłowe

Poniżej znajduje się część kodu źródłowego zawierająca jedynie ważniejsze funkcje. Jest to testowa wersja algorytmu poszukiwania optymalnego mieszanego kodu Huffmana metodą porównań. Wersja testowa zapewnia zapis do pliku tekstowego pełnej informacji o każdej z D iteracji tworzenia drzew kodu mieszanego (zarówno w kolejności wyznaczanych drzew jak i w kolejności posortowanej według rosnącej średniej L_{mix}). Dla ułatwienia testów parametry rozkładu testowanego źródła S są wczytywane z pliku tekstowego, który zawiera liczbę n symboli źródła, a po niej w kolejnych wierszach niemalejący ciąg licznosci lub prawdopodobieństw symboli źródła. W celu uzyskania wysokiej prędkości działania zastosowano tablice statyczne o charakterze globalnym. Dzięki temu działanie tego rekurencyjnego algorytmu nie wymaga przekazywania zbędnych zestawów parametrów (wystarczy tylko numer poziomu rekurencji).

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>

#define ile 25 //liczba symboli źródła
#define ile_iteracji 677000 //dla 25 symboli źródła
typedef unsigned char byte;

FILE *wyjscie, *wyjscie2, *wejscie;

static int w, z, pp, p, i, j, n, nn, ile_testow, wsk;
static double min;
static long int iter;
static double l, binarne, entropia, temp;
static int prime[54]; //Zawiera zestaw liczb pierwszych mniejszych od 256
static double dlugosc[54]; //Zawiera długości gałęzi dla liczb pierwszych (patrz tabela 4.6)
static int licznik[ile]; //Zawiera liczbę elementów na  $p$ -tym poziomie
static int baza[ile]; //Zawiera liczbę potomków węzła na  $p$ -tym poziomie
static int wynik[ile]; //Zawiera wynikowy ciąg dla najkrótszego  $L_{mix}$ 
static double waga[ile+1][ile+1]; //Zawiera wagi na  $p$ -tym poziomie  $i$ -tego elementu
static double tab[ile+1][ile+1];
static double niesort[ile_iteracji]; //Wartości  $L$  dla każdej iteracji przed sortowaniem
static int iteracje[ile_iteracji][ile]; //Zawiera liczbę elementów dla danej iteracji
/*-----*/
void dane_rzeczywiste(void) //funkcje pominięte (Wprowadzanie parametrów)
void dane_wejsciove(void)
/*-----*/
//Tu wprowadzamy licznosci poszczególnych symboli lub ich prawdopodobieństwa

for (i=0;i<ile;i++)
    for (j=0;j<ile;j++) {
        waga[i][j] = 0; //Wyzerowanie tablicy wag
```



```

        tab[i][j] = 0; //oraz tablicy prawdopodobieństw na wszystkich poziomach
    }

    iter = 0;
    l = 999.0; //Startowa wartość  $L_{mix}$  (maksymalna - poza zakresem)
    binarne = 0.0;
    fscanf(wejście,"%d",&n); //Odczyt liczby symboli wejściowych w danym teście
    nn = n;
    p = 0; //Poziom zerowy - zawiera posortowany rozkład prawdopodobieństw

    for (i=0;i<n;i++) fscanf(wejście,"%lf",&tab[0][i]);
    //Odczyt prawdopodobieństw lub licznosci posortowanych rosnąco

    licznik[p] = nn;

    fprintf(wyjście,"Zestaw nr: %d - Liczba prawdopodobieństw: %d\n",z+1,n);
    fprintf(wyjście2,"Zestaw nr: %d - Liczba prawdopodobieństw: %d",z+1,n);
}
/*-----*/
void sporzadz_drzewo(void)
{
    if(waga[p][0]<binarne) fprintf(wyjście,"____"); //Jeśli  $L_{mix} < L_2$ 
    fprintf(wyjście,"L = %1.4f p=%2d ",waga[p][0],p); //Wyświetl każdą iterację

    niesort[iter] = waga[p][0]; //Przygotowanie tablic do sortowania

    if (waga[p][0]<l) { //Zapamiętaj i zapisz gdy znaleziono lepsze rozwiązanie
        l = waga[p][0]; //l = Najlepszy aktualny wynik  $L_{mix}$ 
        pp = p;
        for (i=0; i<pp; i++) wynik[i] = baza[i];
    }

    for (i=p-1; i<n; i++) baza[i] = 0; //Niszczenie wartościowości na wyższych poziomach
    if (iter==0) binarne = l; //Pomiar dla drzewa binarnego
    iter++;
}
/*-----*/
void rekurencja(void)
{
    int j, k, znacznik;
    double sum, wg, dlugosc_wg;

    j = 0;
    if (nn>1) { //Wykonuj jeśli są jeszcze co najmniej 2 węzły
        while (prime[j]<=nn) { //Dla każdej liczby pierwszej

            k = prime[j];
            dlugosc_wg = dlugosc[j];
            j++;
            sum = 0;
            wg = 0;
            baza[p] = k; //Wartościowość - liczba gałęzi łączonych na tym poziomie

            for (i=0; i<k; i++) {
                sum += tab[p][i];
                wg += (waga[p][i] + dlugosc_wg) * tab[p][i];
            }
        }
    }
}

```



```

//Średnia długość słowa kodowego dla elementów już podłączonych do tworzonego węzła
void rekurencja()
{
    wg = wg/sum;
    i = 0;
    znacznik = 1; //Tworzenie następnego poziomu

    while (znacznik) { //Przesortowanie
        if ( (i+k<nn) && (tab[p][i+k]<sum) ) {
            tab[p+1][i] = tab[p][i+k];
            waga[p+1][i] = waga[p][i+k];
            i++;
        }
        else znacznik = 0;
    }

    tab[p+1][i] = sum;
    waga[p+1][i] = wg;

    while (i+k<nn) {
        tab[p+1][i+1] = tab[p][i+k];
        waga[p+1][i+1] = waga[p][i+k];
        i++;
    }

    p = p+1;
    nn = nn-k+1;
    licznik[p] = nn;

    rekurencja(); //Wejście do następnego poziomu

    p = p-1;
    nn = nn+k-1;

} //endwhile

} else sporzadz_drzewo(); //endif - zapisz wyniki do pliku
}
/*-----*/
void entropy(void)
{
    entropia = 0.0;
    temp = 0;
    for (i=0; i<n; i++) temp += tab[0][i]; ///
    for (i=0; i<n; i++) entropia -= (tab[0][i]/temp) * log( (tab[0][i]/temp) );
    entropia /= log(2);
    fprintf(wyjście,"nEntropia = %1.10fn",entropia);
    fprintf(wyjście2,"nEntropia = %1.10fn",entropia);
}

```



```

/*-----*/
void posortuj(void)
{
    fprintf(wyjście2, "\n\n");
    for (j=0; j<iter; j++) {
        min = 999.0;
        wsk = 0;
        for (i=0; i<iter; i++)
            if (niesort[i]<min) {
                min = niesort[i];
                wsk = i;
            }

        if (min<binarne) fprintf(wyjście2, " L = %lf ", min);
        else fprintf(wyjście2, "L = %lf ", min);
        for (w=0; w<n; w++)
            fprintf(wyjście2, "%d ", iteracje[wsk][w]);
        fprintf(wyjście2, "\n");
        niesort[wsk] = 1000;
    }
    fprintf(wyjście2, "-----\n");
}
/*-----*/
int main()
{
    wyjście = fopen("wynik", "wt");
    wyjście2 = fopen("sort", "wt");
    wejście = fopen("dane", "rt");

    fscanf(wejście, "%d", &ile_testow); //Odczyt liczby testów w pliku

    for (z=0; z<ile_testow; z++) {
        dane_wejsciowe(); //Ustawienie parametrów
        rekurencja(); //Wykonanie kolejnego testu
        entropy();

        //Tu znajdują się (pominięte) operacje zapisu wyników do plików tekstowych

        fclose(wyjście);
        posortuj();
    }

    fclose(wyjście2);
    fclose(wejście);
    return 1;
}

```



Literatura

1. **J. Abrahams, M. J. Lipman** „Zero-redundancy coding for unequal code symbol costs”, IEEE Transactions on Information Theory, vol. 38, no. 5, s. 1583-1586, wrzesień 1992
2. **N. Abramson** „Teoria informacji i kodowania”, Państwowe Wydawnictwo Naukowe, Warszawa 1969
3. **Y. S. Abu-Mostafa, R. J. McEliece** „Maximal codeword lengths in Huffman codes”, Computers and Mathematics with Applications, no. 39, s. 129-134, 2000
4. **A. V. Aho, J. E. Hopcroft, J. D. Ullman** „Projektowanie i analiza algorytmów”, Wydawnictwo Helion, Gliwice 2003
5. **A. V. Aho, J. D. Ullman** „Wykłady z informatyki z przykładami w języku C”, Wydawnictwo Helion, Gliwice 2003
6. **V. Bhaskaran, K. Konstantinides** „Image and video compression standards – algorithms and architectures” S.Ed., Kluwer Academic Publishers, 1997
7. **A. Bookstein, S.T. Klein, T. Raita** „Is Huffman coding dead?”, Research and Development in Information Retrieval, s. 80-87, 1993
8. **S. D. Bradley** „Coding the binary symmetric source with a fidelity criterion”, IEEE Transactions on Information Theory, vol. 17, no. 6, s. 749-751, listopad 1971
9. **S. Brandt** „Analiza Danych”, Wydawnictwo Naukowe PWN, Warszawa 1999
10. **M. Buro** „On the maximum length of Huffman codes”, Information Processing Letters, no. 45, s. 219-223, 1993
11. **R. M. Capocelli, A. De Santis** „A note on D-ary Huffman codes”, IEEE Transactions on Information Theory, vol. 37, no. 1, s. 174-179, styczeń 1991
12. **R. M. Capocelli, A. De Santis** „New Bounds on the Redundancy of Huffman Codes”, IEEE Transactions on Information Theory, vol. 37, no. 4, s. 1095-1104, lipiec 1991
13. **Y. Choueka, S. T. Klein, Y. Perl** „Efficient variant of Huffman codes in high level languages”, Proc. 8-th ACM-SIGIR Conf., s. 122-130, Montreal 1985
14. **R. A. Chowdhury, M. Kaykobad, I. King** „An efficient decoding technique for Huffman codes”, Information Processing Letters, vol. 81, s. 305-308, 2002
15. **R. Cierniak, L. Rutkowski** „Nowe metody kompresji obrazu – zastosowanie sieci neuronowych”, Wydawnictwo Gnome, Katowice 2000
16. **T. H. Cormen, C. E. Leiserson, R. L. Rivest** „Wprowadzenie do algorytmów”, Wyd. 5, Wydawnictwa Naukowo-Techniczne, Warszawa 2001

17. **J. Cytowski** „*Metody i algorytmy sztucznej inteligencji w cyfrowym przetwarzaniu sygnałów*”, Akademicka Oficyna Wydawnicza PLJ, Warszawa 1999
18. **A. Czyżewski** „*Dźwięk cyfrowy. Wybrane zagadnienia teoretyczne, technologia, zastosowania*”, Akademicka Oficyna Wydawnicza EXIT, Warszawa 1998
19. **R. De Prisco, A. De Santis** „*On the redundancy achieved by Huffman codes*” Design, codes and cryptography, vol. 15, s. 29-45, 1998
20. **M. Domański** „*Zaawansowane techniki kompresji obrazów i sekwencji wizyjnych*”, Wydawnictwo Politechniki Poznańskiej, Poznań 2000
21. **A. Drozdek** „*Wprowadzenie do kompresji danych*”, Wydawnictwa Naukowo-Techniczne, Warszawa 1999
22. **A.-R. Elabdalla, M. I. Irshid** „*An efficient bitwise Huffman coding technique based on source mapping*”, Computers and Electrical Engineering 27, s. 265-272, 2001
23. **F. Fabris** „*Variable-Length -to- Variable-Length source coding: A greedy step-by-step algorithm*”, IEEE Transactions on Information Theory, vol. 38, s. 1609-1617, wrzesień 1992
24. **Z. Fortuna, B. Macukow, J. Wąsowski** „*Metody numeryczne*”, Wyd. 4, Wydawnictwa Naukowo-Techniczne, Warszawa 1998
25. **L. Gajek, M. Kałuszka** „*Wnioskowanie statystyczne – modele i metody*”, Wyd. 2 Wydawnictwa Naukowo-Techniczne, Warszawa 1996
26. **R. G. Gallager** „*Optimal source codes for geometrically distributed integer alphabets*”, IEEE Transactions on Information Theory, s. 228-230, marzec 1975
27. **R. G. Gallager** „*Variations on a theme by Huffman*”, IEEE Transactions on Information Theory, vol. 24, no. 6, s. 668-674, listopad 1978
28. **N. C. Geçkinli** „*Two corollaries to the Huffman coding procedure*”, IEEE Transactions on Information Theory, s. 342-344, maj 1975
29. **E. N. Gilbert** „*Coding with digits of unequal cost*”, IEEE Transactions on Information Theory, vol. 41, no. 2, s. 596-600, marzec 1995
30. **J. D. Golić, M. M. Obradović** „*A lower bound on the redundancy of D-ary Huffman codes*”, IEEE Transactions on Information Theory, vol. 33, no. 6, s. 910-911, listopad 1987
31. **M. J. Golin, G. Rote** „*A dynamic programming algorithm for constructing optimal prefix-free codes with unequal letter costs*”, IEEE Transactions on Information Theory, vol. 44, no. 5, s. 1770-1781, wrzesień 1998
32. **M. J. Golin** „*A combinatorial approach to Golomb forests*”, Theoretical Computer Science, no. 263, s. 283-304, 2001

33. **S. W. Golomb** „*Run-length encoding*”, IEEE Transactions on Information Theory, vol. 12, s. 399-401, lipiec 1966
34. **M. Grabowski** „*Analiza matematyczna – Powtórzenie, ćwiczenia i zbiór zadań*”, Wydawnictwa Naukowo-Techniczne, Warszawa 1997
35. **T. Guionnet, Ch. Guillemot** „*Robust decoding of arithmetic codes for image transmission over error-prone channels*”, Proc. of ICIP-03, Barcelona 2003
36. **U. Günther, R. Niolescu** „*A low-cost decoder for arbitrary binary variable-length codes*”, źródło internetowe: <http://citeseer.ist.psu.edu/99958.html>
37. **K. Heim** „*Metody kompresji danych*”, Wydawnictwo MIKOM, Warszawa 2000
38. **D. A. Huffman** „*A method for the construction of minimum-redundancy codes*”, Proc. IRE, vol. 40, s. 1098-1101, 1952
39. **A. Kato, Te Sun Han, H. Nagaoka** „*A D-ary Huffman code for a class of sources with countably infinite alphabets*”, Proc. 1995 IEEE Int. Symp. on Information Theory, p. 324, 1995
40. **A. Kato, Te Sun Han, H. Nagaoka** „*Huffman coding with an infinite alphabet*”, IEEE Transactions on Information Theory, vol. 42, no. 3, s. 977-984, maj 1996
41. **G. O. H. Katona, T. O. H. Nemetz** „*Huffman codes and self-information*”, IEEE Transactions on Information Theory, vol. 22, no. 3, s. 337-340, maj 1976
42. **D. E. Knuth** „*Dynamic Huffman coding*”, Journal of Algorithms, vol. 6, s. 163-180, 1983
43. **T. Linder, V. Tarokh, K. Zeger** „*Existence of optimal prefix codes for infinite source alphabets*”, IEEE Transactions on Information Theory, vol. 43, no. 6, s. 2026-2028, listopad 1997
44. **D. Long and W. Jia** „*On the optimal coding*”, Lecture Notes in Computer Science, vol. 2195, s. 94-101, 2001
45. **G. Longo, G. Galasso** „*An application of informational divergence to Huffman codes*”, IEEE Transactions on Information Theory, vol. 28, no. 1, s. 36-43, lipiec 1966
46. **Z. Łukasik** „*Teoria informacji i sygnałów*”, Wyd. 3, Wydawnictwo Politechniki Radomskiej, Radom 1999
47. **D. J. C. MacKay** „*Information Theory, Inference, and Learning Algorithms*”, Version 6.0, Cambridge University Press, czerwiec 2003
48. **D. Manstetten** „*Tight bounds on the redundancy of Huffman codes*”, IEEE Transactions on Information Theory, vol. 38, no. 1, s. 144-151, styczeń 1992

49. **J. L. Massey** „*Applied digital information theory I*”, script only for personal use, kopia elektroniczna
50. **B. J. McKenzie, T. Bell** „*Compression of sparse matrices by blocked Rice coding*”, IEEE Transactions on Information Theory, vol. 47, no. 3, s. 1223-1230, marzec 2001
51. **N. Merhav, G. Seroussi, M. J. Weinberger** „*Optimal prefix codes for sources with two-sided geometric distributions*”, International Symposium on Information Theory, Ulm, Germany 1997
52. **B. Meyer, P. Tischer** „*TMW – a new method for lossless image compression*”, 1997 Picture Coding Symposium, VDE 1997, s. 533-538, Berlin
53. **W. Mizerski i inni** „*Tablice matematyczne*”, Wyd. 4, Wydawnictwo Adamantan, Warszawa 2001
54. **A. Moffat, A. Turpin** „*Efficient construction of minimum-redundancy codes for large alphabets*”, IEEE Transactions on Information Theory, vol. 44, no. 4, s. 1650-1657, lipiec 1998
55. **B. Mrozek, Z. Mrozek** „*Matlab 6 – poradnik użytkownika*”, Wydawnictwo PLJ, Warszawa 2001
56. **H.-O. Peitgen, H. Jürgens, D. Saupe** „*Fraktale – granice chaosu*”, Tom I, Wydawnictwo Naukowe PWN, Warszawa 1995
57. **A. Przelaskowski** „*Kompresja danych obrazowych. Zarys zagadnień istotnych*”, Akademicka Oficyna Wydawnicza PLJ, W przygotowaniu
58. **R. F. Rice** „*Some practical universal noiseless coding techniques*”, Jet Propulsion Laboratory, JPL Publication 79-22, Pasadena, CA, marzec 1979
59. **J. J. Rissanen, G. G. Langdon** „*Arithmetic coding*”, IBM Journal of Research and Development, no. 23, vol. 2, s. 149-162, marzec 1979
60. **D. E. Robling Denning** „*Kryptografia i ochrona danych*”, Wydawnictwa Naukowo-Techniczne, Warszawa 1993
61. **L. G. Rueda, B. J. Oommen** „*A nearly-optimal Fano-based coding algorithm*”, Information Processing and Management, vol. 40, s. 257-268, 2004
62. **K. Sayood** „*Kompresja danych – wprowadzenie*”, Wyd. 2, Wydawnictwo RM, Warszawa 2002
63. **K. Sayood (editor)** „*Lossless Compression Handbook*”, Academic Press USA, 2003
64. **C. E. Shannon** „*A mathematical theory of communication*”, The Bell System Technical Journal, vol. 27, s. 379-423, 623-656, lipiec, październik 1948
65. **W. Sierpiński** „*Podstawy algebry wyższej*”, Warszawa-Wrocław 1946, kopia elektroniczna z bazy ICM

66. **W. Skarbek i inni** „*Multimedia i standardy kompresji danych*”, Akademicka Oficyna Wydawnicza PLJ, Warszawa 1998
67. **W. Skarbek** „*Metody reprezentacji obrazów cyfrowych*”, Akademicka Oficyna Wydawnicza PLJ, Warszawa 1993
68. **R. Stasiński, G. Ulacha** „*Huffman codes revisited*”, Proceedings of 24th Symposium of Information Theory in The Benelux, Veldhoven 2003
69. **P. J. Swaszek, S. Wali** „*Generating probabilities with a specified entropy*”, Journal of The Franklin Institute, no. 337, s. 97-103, Elsevier Science Ltd., 2000
70. **W. Szpankowski** „*Asymptotic average redundancy of Huffman (and other) block codes*”, IEEE Transactions on Information Theory, vol. 46, no. 7, s. 2434-2443, listopad 2000
71. **K. Świącicki, A. Walat** „*Szkolny słownik informatyki*”, Polska Oficyna Wydawnicza BGW, Warszawa 1992
72. **H. Tanaka** „*Data structure of Huffman codes and its application to efficient encoding and decoding*”, IEEE Transactions on Information Theory, vol. 33, no. 1, s. 154-156, styczeń 1987
73. **J. R. Taylor** „*Wstęp do analizy błędu pomiarowego*”, Wydawnictwo Naukowe PWN, Warszawa 1995
74. **B. P. Tunstall** „*Synthesis of Noiseless Compression Codes*”, Ph. D. thesis, Georgia Institute of Technology, Atlanta, GA, wrzesień 1967
75. **G. Ulacha** „*Kompresja danych z wykorzystaniem kodów mieszanych*”, Materiały VII Sesji Naukowej Informatyki Wydziału Informatyki Politechniki Szczecińskiej, Szczecin 2002
76. **G. Ulacha** „*Analiza konstrukcji optymalnych kodów mieszanych*”, Materiały VIII Sesji Naukowej Informatyki Wydziału Informatyki Politechniki Szczecińskiej, Szczecin 2003
77. **G. Ulacha** „*Wzrost efektywności kompresji JPEG – algorytm szybkiego dekodera liczb ternarnych*”, Materiały VI konferencji Reprogramowalne Układy Cyfrowe, Szczecin 2003
78. **G. Ulacha** „*Podstawowe warunki stosowania mieszanych kodów Huffmana*”, Materiały V Ogólnopolskich Warsztatów Doktoranckich, Gliwice 2003
79. **G. Ulacha** „*Efektywność mieszanych kodów Huffmana*”, Materiały konferencyjne VIII Poznańskich Warsztatów Telekomunikacyjnych, Poznań 2003

80. **G. Ulacha** „*Metody tworzenia efektywnych rozszerzonych kodów Huffmana o małych wymaganiach pamięciowych*”, Materiały konferencyjne VIII Poznańskich Warsztatów Telekomunikacyjnych, Poznań 2003
81. **G. Ulacha, R. Maciaszczyk, T. Mąka** „*Zastosowanie predykcji liniowej do bezstratnej kompresji obrazów przy wykorzystaniu algorytmów genetycznych*”, Materiały VII Sesji Naukowej Informatyki Wydziału Informatyki Politechniki Szczecińskiej, Szczecin 2002
82. **G. Ulacha, R. Stasiński** „*Source binary coding using non-binary Huffman codes*”, Materiały konferencyjne Krajowego Sympozjum Nauk Radiowych URSI, Poznań 2002
83. **G. Ulacha, R. Stasiński** „*Mixed Huffman codes*”, Proceedings of 9th Advanced Computer Systems, Międzyzdroje 2002
84. **G. Ulacha, R. Stasiński** „*Efektywne kodowanie binarne elementów r-narnych dla potrzeb kodowania źródła*”, Materiały konferencyjne VII Poznańskich Warsztatów Telekomunikacyjnych, Poznań 2002
85. **G. Ulacha, R. Stasiński** „*On the optimization of Golomb codes using block coding of m-nary elements*”, Proceedings of 10th Advanced Computer Systems, Międzyzdroje 2003
86. **J. S. Vitter** „*Dynamic Huffman coding*”, ACM Transactions on Mathematical Software, vol. 15, s. 158-167, 1989
87. **W. Waliszewski i inni** „*Matematyka – encyklopedia szkolna*”, Wydawnictwa Szkolne i Pedagogiczne, Warszawa 1990
88. **M. Wang** „*Almost asymptotically optimal flag encoding of the integers*”, IEEE Transactions on Information Theory, vol. 34, no. 2, s. 324-326, marzec 1988
89. **T. A. Welch** „*A technique for high-performance data compression*”, IEEE Computer, s. 8-19, czerwiec 1984
90. **F. Willems, Y. Shtakov, T. Tjalkens** „*Reflections on the prize paper: 'The context-tree weighting method: basic properties'*”, IEEE Information Theory Society Newsletter, vol. 47, no. 1, s. 1, 20-27, marzec 1997
91. **I. H. Witten, R. M. Neal, J. G. Cleary** „*Arithmetic coding for data compression*”, Communications of the ACM, vol. 30, no. 6, s. 520-540, czerwiec 1987
92. **A. Zalewski** „*Programowanie w językach C i C++ z wykorzystaniem Borland C++*”, Wydawnictwo Nakom, Poznań 1996
93. **J. Ziv, A. Lempel** „*A universal algorithm for sequential data compression*”, IEEE Transactions on Information Theory, vol. 23, no. 3, s. 337-343, maj 1977

94. **J. Ziv, A. Lempel** „*Compression of individual sequences via variable-rate coding*”,
IEEE Transactions on Information Theory, vol. 24, no. 5, s. 530-536, wrzesień 1978

Biblioteka Wydziału Informatyki
Politechniki Szczecińskiej

DISS-33

Biblioteka Główna
Politechniki Szczecińskiej

SIN . 3199



015-003199-00-0

ARF-33

Spis treści

1. WPROWADZENIE	1
1.1. PODSTAWOWE OKREŚLENIA UŻYWANE W PRACY	2
1.2. PODSTAWOWE KODY	2
2. CHARAKTERYSTYKA	3
2.1. BINARYNY KOD HUFFMANA	4
2.2. KOD DWUBAZOWY	6
2.3. WIELOWARTOŚCIOWY KOD HUFFMANA	6
2.4. ROTACJONALNY KOD HUFFMANA	6
2.5. KOD TERNARYJNY I INNE KODY ENTROPICZNE	7
2.6. KOD ARYTMETYCZNY I INNE KODY ENTROPICZNE	7
3. EFEKTYWNA REPREZENTACJA SYMBOLI WIELOWARTOŚCIOWYCH	7
3.1. ALGORYTM	9
4. MIESZANE KODY HUFFMANA	9
4.1. PODSTAWOWE WŁAŚCIWOŚCI MIESZANYCH KODÓW HUFFMANA	11
4.2. TESTY EFEKTYWNOŚCI	12
5. KOMBINACJA OPTIMALNYCH MIESZANYCH KODÓW HUFFMANA	14
5.1. PROBLEMA INTEGRALNY	14
5.2. PROBLEMA FRAKCYJNY	14
5.3. ALGORYTM	14
6. SYMBIOTYCZNE MIESZANE KODY HUFFMANA O KRÓTKIM	15
6.1. CZYNNIKI KONSTRUKCJI DRZEWIA KODOWEGO	15
6.2. ALGORYTM	15
6.3. KODY KOLEJNYCH WSKAZIWEK	16
6.4. ZMIENIEN	16
6.5. NAJYMOCNIENIE	16
6.6. TESTY	17
6.7. WNIOSKI	17
6.8. WNIOSKI	17
6.9. WNIOSKI	17
6.10. WNIOSKI	17
6.11. WNIOSKI	17
6.12. WNIOSKI	17
6.13. WNIOSKI	17
6.14. WNIOSKI	17
6.15. WNIOSKI	17
6.16. WNIOSKI	17
6.17. WNIOSKI	17
6.18. WNIOSKI	17
6.19. WNIOSKI	17
6.20. WNIOSKI	17
6.21. WNIOSKI	17
6.22. WNIOSKI	17
6.23. WNIOSKI	17
6.24. WNIOSKI	17
6.25. WNIOSKI	17
6.26. WNIOSKI	17
6.27. WNIOSKI	17
6.28. WNIOSKI	17
6.29. WNIOSKI	17
6.30. WNIOSKI	17
6.31. WNIOSKI	17
6.32. WNIOSKI	17
6.33. WNIOSKI	17
6.34. WNIOSKI	17
6.35. WNIOSKI	17
6.36. WNIOSKI	17
6.37. WNIOSKI	17
6.38. WNIOSKI	17
6.39. WNIOSKI	17
6.40. WNIOSKI	17
6.41. WNIOSKI	17
6.42. WNIOSKI	17
6.43. WNIOSKI	17
6.44. WNIOSKI	17
6.45. WNIOSKI	17
6.46. WNIOSKI	17
6.47. WNIOSKI	17
6.48. WNIOSKI	17
6.49. WNIOSKI	17
6.50. WNIOSKI	17
6.51. WNIOSKI	17
6.52. WNIOSKI	17
6.53. WNIOSKI	17
6.54. WNIOSKI	17
6.55. WNIOSKI	17
6.56. WNIOSKI	17
6.57. WNIOSKI	17
6.58. WNIOSKI	17
6.59. WNIOSKI	17
6.60. WNIOSKI	17
6.61. WNIOSKI	17
6.62. WNIOSKI	17
6.63. WNIOSKI	17
6.64. WNIOSKI	17
6.65. WNIOSKI	17
6.66. WNIOSKI	17
6.67. WNIOSKI	17
6.68. WNIOSKI	17
6.69. WNIOSKI	17
6.70. WNIOSKI	17
6.71. WNIOSKI	17
6.72. WNIOSKI	17
6.73. WNIOSKI	17
6.74. WNIOSKI	17
6.75. WNIOSKI	17
6.76. WNIOSKI	17
6.77. WNIOSKI	17
6.78. WNIOSKI	17
6.79. WNIOSKI	17
6.80. WNIOSKI	17
6.81. WNIOSKI	17
6.82. WNIOSKI	17
6.83. WNIOSKI	17
6.84. WNIOSKI	17
6.85. WNIOSKI	17
6.86. WNIOSKI	17
6.87. WNIOSKI	17
6.88. WNIOSKI	17
6.89. WNIOSKI	17
6.90. WNIOSKI	17
6.91. WNIOSKI	17
6.92. WNIOSKI	17
6.93. WNIOSKI	17
6.94. WNIOSKI	17
6.95. WNIOSKI	17
6.96. WNIOSKI	17
6.97. WNIOSKI	17
6.98. WNIOSKI	17
6.99. WNIOSKI	17
6.100. WNIOSKI	17

AUTOREFERAT PRACY DOKTORSKIEJ:

„Metoda kompresji danych z wykorzystaniem zmiennych alfabetów symboli transmisyjnych”

Biblioteka Główna
Zachodniopomorskiego Uniwersytetu
Technologicznego w Szczecinie
SIN.3199+Autoreferat



015-003199-00-0

Autor: mgr inż. Grzegorz Ulacha

Promotor: dr hab. inż. Ryszard Stasiński, prof. PS

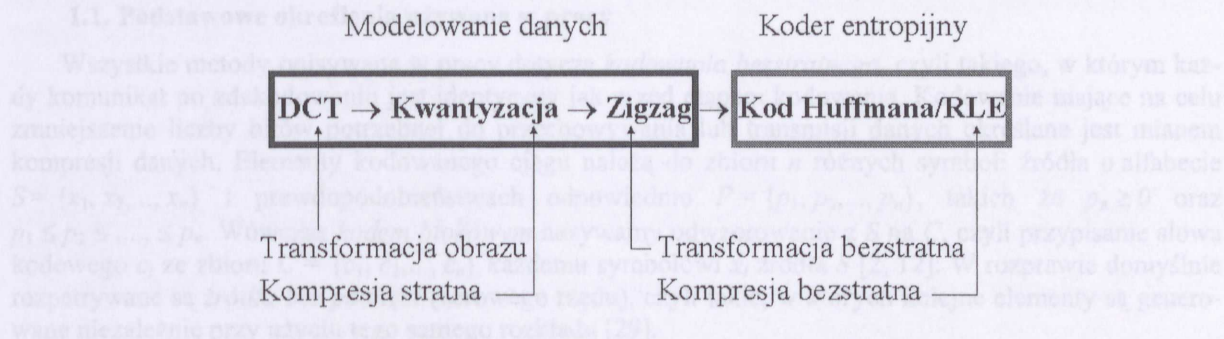
Szczecin, 2004r.

Spis treści

1. WPROWADZENIE.....	1
1.1. PODSTAWOWE OKREŚLENIA UŻYWANE W PRACY	2
1.2. PODSTAWOWE CECHY KODÓW	2
2. CHARAKTERYSTYKA PODSTAWOWYCH KODÓW ENTROPIJNYCH – STAN AKTUALNY	3
2.1. BINARNY KOD HUFFMANA.....	4
2.2. KOD DWUFAZOWY	6
2.3. WIELOWARTOŚCIOWY KOD HUFFMANA	6
2.4. ROZSZERZONY KOD HUFFMANA	6
2.5. KOD TUNSTALLA I JEGO MODYFIKACJE	7
2.6. KOD ARYTMETYCZNY I INNE KODY ENTROPIJNE	7
3. EFEKTYWNA REPREZENTACJA SYMBOLI WIELOWARTOŚCIOWYCH	7
3.1. ALGORYTM WYZNACZANIA WZORCOWYCH DŁUGOŚCI BLOKÓW (Q) ELEMENTÓW R -NARNYCH ..	9
4. MIESZANE KODY HUFFMANA.....	9
4.1. PODSTAWOWE WŁASNOŚCI MIESZANYCH KODÓW HUFFMANA	11
4.2. TESTY EFEKTYWNOŚCI.....	12
5. KONSTRUKCJA OPTYMALNYCH MIESZANYCH KODÓW HUFFMANA.....	14
5.1. METODA INTEGRALNYCH PODDRZEW	14
5.2. PODDRZEWA INTEGRALNE BINARNIE.....	14
5.3. KOD BINARNO-TERNARNY	14
6. SUBOPTYMALNE MIESZANE KODY HUFFMANA O KRÓTKIM CZASIE KONSTRUKCJI DRZEWA KODOWEGO	15
6.1. METODA KOLEJNYCH UŚCIŚLEŃ	15
6.2. KOD DWUPOZIOMOWY Z WYKORZYSTANIEM METODY KOLEJNYCH UŚCIŚLEŃ	15
6.3. SUBOPTYMALNY KOD MIESZANY ZE STAŁYM OPÓŹNIENIEM.....	16
6.4. OPTYMALNY I SUBOPTYMALNY KOD MIESZANY Z MAŁYM OPÓŹNIENIEM	16
7. DYNAMICZNE MIESZANE KODY HUFFMANA.....	17
8. ZASTOSOWANIA MIESZANYCH KODÓW HUFFMANA	17
8.1. KOMPRESJA ELEMENTÓW DC W STANDARDZIE JPEG.....	17
8.2. ZASTOSOWANIE KODU BINARNO-TERNARNEGO DO BEZSTRATNEJ KOMPRESJI OBRAZÓW	18
8.3. ZASTOSOWANIE KODÓW MIESZANYCH DO ZWIĘKSZENIA EFEKTYWNOŚCI KODU GOLOMBA	18
9. PODSUMOWANIE.....	19
LITERATURA	21

1. Wprowadzenie

W systemach informatycznych istotnym problemem jest sposób efektywnego przechowywania i transmisji danych. Należy dążyć do minimalizacji kosztów użytkowania systemów informatycznych, np. dzięki stosowaniu wydajnych technik kompresji danych. Oprócz zwykłej archiwizacji danych techniki takie mają zastosowanie np. przy zapisie filmów na płytach DVD, w cyfrowych aparatach fotograficznych, transmisji telewizji cyfrowej oraz UMTS. Ogólnie metody kompresji danych można podzielić na dwa główne typy: *stratną* (np. JPEG, MP3) i *bezstratną* (odwracalną np. kompresja dokumentów, obrazu PNG). Na rysunku 1 przedstawiono podział kodera JPEG - jednej z popularnych metod kompresji obrazu. Stratne metody kompresji składają się z etapu modelowania danych (użycie transformat, częściowego usuwania informacji) oraz z etapu końcowego, w którym dane przygotowane w pierwszym etapie poddawane są bezstratnej kompresji entropijnej. Zaproponowana w rozprawie metoda dotyczy wyłącznie bezstratnej kompresji danych.



Rys. 1 Podział kodera JPEG na dwa główne etapy kodowania

Tematem niniejszej rozprawy doktorskiej jest:

„Metoda kompresji danych z wykorzystaniem zmiennych alfabetów symboli transmisyjnych”.

Opracowana metoda kompresji danych jest rozwinięciem binarnego kodu Huffmana. **Celem pracy** jest:

„Opracowanie algorytmu kompresji danych opartego na mieszanych kodach Huffmana, wykorzystujących jednocześnie zastosowanie wielu kodów r -narnych”.

Nowością tej metody jest łączne wykorzystanie zalet binarnych i wielowartościowych kodów Huffmana, poprzez użycie w trakcie kodowania wielu elementów o różnych wartościowościach. Do uzyskania wysokiej efektywności binarnego kodowania cyfr r -narnych wykorzystywana jest ich reprezentacja w postaci liczb złożonych z q cyfr r -narnych zwanych w pracy liczbami B , oraz dalsza kompresja tych liczb przy użyciu kodu dwufazowego. Proponowana przez autora praktyczna realizacja metody kompresji określana jest mianem mieszanego kodu Huffmana ze względu na duże podobieństwa do binarnego kodu Huffmana pod względem ich własności. Poniżej przedstawiona jest **teza pracy**:

„Proponowany algorytm zapewnia efektywność kompresji nie mniejszą od efektywności binarnego kodu Huffmana przy jednocześnie niskiej złożoności implementacyjnej: $E_{mix} \geq E_2$ ”.

W tezie pracy zawarto dwie główne zalety proponowanej metody kompresji: ma ona niską złożoność implementacyjną (w porównaniu do rozszerzonych kodów Huffmana), a także charakteryzuje się wzrostem wydajności kompresji w stosunku do binarnego kodu Huffmana. Poziom wzrostu efektywności zależy od rozkładu prawdopodobieństwa kodowanego źródła i w najgorszym przypadku mieszany kod Huffmana jest identyczny z binarnym kodem Huffmana, który jest podzbiorem kodów mieszanych dla danego źródła. W trakcie badań autor rozprawy korzystał z własnego oprogramowania w języku C oraz z pakietu Matlab. Rysunki i tabele zawarte w autoreferacie stanowią źródło własne.

Praca składa się z 9 rozdziałów oraz dwóch dodatków. Pierwszy rozdział zawiera podstawowe definicje dotyczące kodów. Podstawowym celem tej pracy jest poszukiwanie kodów efektywniejszych niż binarny kod Huffmana. Przedstawione w rozdziale 2 kody o tej własności posiadają wady, które utrudniają ich praktyczne wykorzystanie (duże wymagania pamięciowe kodera i dekodera lub długi czas wykonania zadania). W rozdziale 3 przedstawiono sposób efektywnej reprezentacji cyfr wielowartościowych, co pozwo-

lilo na praktyczne wykorzystanie wielowartościowych kodów Huffmana. W rozdziale 4 autor zaproponował łączenie różnych wartościowości w jednym mieszanym kodzie Huffmana, przedstawiając własności takiego kodu wraz z przykładami. Kolejny rozdział przedstawia sposób konstrukcji drzewa optymalnego kodu mieszanego dla danego źródła, a także metody przyspieszania etapu konstrukcji takiego kodu. Dla zastosowań, w których podstawowym problemem jest krótki czas konstrukcji drzewa kodowego dla źródeł o dużej liczbie symboli n , autor rozprawy w rozdziale 6 przedstawił kilka propozycji tworzenia suboptymalnych kodów mieszanych. Ponadto omówione zostały propozycje kodów suboptymalnych o stałym oraz o małym opóźnieniu wprowadzanym przez koder i dekoder. Rozdział 7 zawiera analizę działania dynamicznej wersji kodów mieszanych. W kolejnym rozdziale umieszczono przykłady zastosowań mieszanych kodów Huffmana do wzrostu efektywności kodu Golomba, a także algorytmu stratnej (JPEG) i bezstratnej (z użyciem predykcji liniowej) kompresji obrazów. Ostatni rozdział przedstawia wnioski dotyczące zaproponowanej przez autora metody poprawy efektywności kompresji danych. W dodatku A umieszczono wyprowadzenia niektórych wzorów, natomiast dodatek B zawiera elementy kodu źródłowego powstałego na podstawie algorytmu budowy optymalnego mieszanego kodu Huffmana.

1.1. Podstawowe określenia używane w pracy

Wszystkie metody opisywane w pracy dotyczą *kodowania bezstratnego*, czyli takiego, w którym każdy komunikat po zdekodowaniu jest identyczny jak przed etapem kodowania. Kodowanie mające na celu zmniejszenie liczby bitów potrzebnej do przechowywania lub transmisji danych określane jest mianem kompresji danych. Elementy kodowanego ciągu należą do zbioru n różnych symboli źródła o alfabecie $S = \{x_1, x_2, \dots, x_n\}$ i prawdopodobieństwach odpowiednio $P = \{p_1, p_2, \dots, p_n\}$, takich że $p_i \geq 0$ oraz $p_1 \leq p_2 \leq \dots \leq p_n$. Wówczas *kodem blokowym* nazywamy odwzorowanie z S na C , czyli przypisanie słowa kodowego c_i ze zbioru $C = \{c_1, c_2, \dots, c_n\}$ każdemu symbolowi x_i źródła S [2, 12]. W rozprawie domyślnie rozpatrywane są *źródła bez pamięci* (zerowego rzędu), czyli takie, w których kolejne elementy są generowane niezależnie przy użyciu tego samego rozkładu [29].

1.2. Podstawowe cechy kodów

W tym punkcie przedstawione zostaną podstawowe definicje i twierdzenia dotyczące omawianych w pracy kodów blokowych na podstawie [2, 11, 12, 14, 26, 28, 29, 33].

Definicja 1.1 Kod nazywamy *r-narnym* (*r*-wartościowym), jeżeli słowa kodu składają się z cyfr *r*-narnych (z zakresu od zera do $r - 1$). W szczególności w kodzie binarnym ($r = 2$) słowa kodu składają się z zer i jedynek.

Definicja 1.2 Kod nazywamy *kodem przedrostkowym* (prefiksowym), jeżeli nie możemy otrzymać żadnego słowa kodu z innego słowa kodu przez dopisanie do niego cyfr *r*-narnych, czyli żadne słowo nie jest przedrostkiem żadnego innego słowa kodu.

Kody przedrostkowe są *kodami bez opóźnienia* [2]. W pracy rozpatrywane są tylko kody przedrostkowe stanowiące podzbiór kodów jednoznacznie dekodowalnych. Każdy kod przedrostkowy może być opisany drzewem kodowym, którego definicja została przedstawiona poniżej [3, 4]:

Definicja 1.3 Drzewo *r*-narne (w szczególności binarne) składa się z węzłów, z których wychodzi maksymalnie *r* gałęzi w kierunku węzłów potomnych. Węzeł posiada maksymalnie jednego rodzica. Węzeł nie posiadający rodzica nazywany jest *korzeniem drzewa*, a węzeł nie posiadający potomków określany jest *liściem*.

Drzewo *r*-narne jest kompletne, gdy poza liśćmi każdy z węzłów posiada dokładnie *r* potomków. Pojedynczy węzeł nie posiadający rodzica ani potomków także jest drzewem (węzeł ten jest wówczas jednocześnie korzeniem i liściem).

Definicja 1.4 Drzewo mieszane to takie drzewo, w którym każdy *i*-ty węzeł nie będący liściem może mieć dowolną liczbę $r_i \geq 2$ potomków.

Poniżej zostało przedstawione twierdzenie L. G. Krafta z 1949 roku dotyczące kodów przedrostkowych:

Twierdzenie 1.1 (Twierdzenie Krafta) *r*-narny (*r* wartościowy) kod przedrostkowy C złożony ze słów kodu $\{c_1, c_2, \dots, c_n\}$ o długościach $\{l_1, l_2, \dots, l_n\}$ istnieje wtedy i tylko wtedy, gdy:

$$\sum_{i=1}^n r^{-l_i} \leq 1 \quad (1.1)$$

Nierówność staje się równością Krafta dla kompletnych kodów przedrostkowych. Dowód twierdzenia znajduje się w pracy [29]. Jedną z podstawowych cech charakteryzujących źródło jest entropia:

Definicja 1.5 Entropią źródła złożonego z n statystycznie niezależnych symboli o alfabetcie $S = \{x_1, x_2, \dots, x_n\}$ i odpowiadającym im zestawie prawdopodobieństw wystąpienia $P = \{p_1, p_2, \dots, p_n\}$ jest funkcja:

$$H(S) = -\sum_{i=1}^n p_i \cdot \log_2 p_i \quad (1.2)$$

Entropia jest średnią ilością informacji uzyskaną przez obserwację pojedynczej wiadomości na wyjściu źródła. W przypadku binarnego kodowania komunikatów entropia jest miarą określającą minimalną średnią liczbę bitów potrzebnych do zakodowania pojedynczego symbolu z ciągu utworzonego z kolejnych symboli wygenerowanych przez źródło. Funkcja entropii jest ograniczona w następujący sposób:

$$0 = H(1, 0, \dots, 0) \leq H(p_1, \dots, p_n) \leq H\left(\frac{1}{n}, \dots, \frac{1}{n}\right) = \log_2 n \quad (1.3)$$

Jak wykazał Shannon, dla dowolnej bezstratnej metody kompresji średnia liczba bitów potrzebnych do zakodowania symboli źródła jest nie mniejsza od wartości entropii [31]. Dla kodów przedrostkowych można wyznaczyć *średnią długość L_{sr} słowa kodowego* (wartość oczekiwaną), gdzie l_i jest długością słowa kodu c_i kodującego symbol x_i :

$$L_{sr} = \sum_{i=1}^n p_i \cdot l_i \quad (1.4)$$

Jeżeli binarny kod przedrostkowy zostanie przedstawiony w postaci drzewa, to można wyliczyć L_{sr} także jako sumę prawdopodobieństw wystąpienia wszystkich węzłów tego drzewa (bez prawdopodobieństw liści tego drzewa).

Definicja 1.6 Efektywnością kompresji $\hat{E}$ nazywamy stosunek wartości entropii do średniej długości kodu.

$$\hat{E} = \frac{H(S)}{L_{sr}} \quad (1.5)$$

W pracy jest często używana procentowa efektywność kompresji E :

$$E = \hat{E} \cdot 100\% = \frac{H(S)}{L_{sr}} \cdot 100\% \quad (1.6)$$

W przypadku analizy każdej modyfikacji binarnego kodu Huffmana możemy porównywać zmiany efektywności między podstawowym i zmodyfikowanym kodem Huffmana. Służy do tego *procentowa waga zysku*, która jest stosunkiem zysku między średnią długością L_{mix} słowa nowego kodu a średnią długością L_2 binarnego kodu Huffmana, do maksymalnego możliwego zysku, jaki można uzyskać (osiągając 100% efektywności kodu):

$$W = \frac{L_2 - L_{mix}}{L_2 - H(S)} \cdot 100\% \quad (1.7)$$

Definicja 1.7 Kodem optymalnym nazywamy kod, dla którego średnia długość słowa kodowego L_{sr} jest najmniejsza spośród wszystkich kodów dla danego rozkładu prawdopodobieństwa P . Dla takiego kodu zachodzi zależność:

$$H(p_1, \dots, p_n) \leq L_{sr} < H(p_1, \dots, p_n) + 1 \quad (1.8)$$

Zależność (1.8) dotyczy kodów blokowych, dla których każdy symbol c_i reprezentuje pojedynczy symbol źródła x_i . Jej dowód przedstawiono w pracach [12, 29].

2. Charakterystyka podstawowych kodów entropijnych – stan aktualny

Metody kodowania źródła znane były jeszcze przed pojawieniem się pierwszych komputerów. Zarówno opracowania teoretyczne, jak i praktyczne metody dotyczyły zastosowań między innymi w transmisji danych (np. kod Morse'a). Teorię zawartości informacyjnej sygnałów i związanych z nimi kodów (w tym kodów dla kanałów bezszumowych) usystematyzował Claude Shannon przedstawiając ją w 1948 roku w pracy [31]. Praca ta stała się bazą wiedzy dla podstaw teorii informacji, której szybki rozwój związany był z rozwojem telekomunikacji i maszyn cyfrowych. Shannon przedstawił wiele cech optymalnych kodów blokowych, lecz algorytm konstrukcji optymalnego kodu blokowego dla źródła o skończonej liczbie symboli opisał dopiero w roku 1952 Dawid Huffman w pracy [22]. Prosta konstrukcja drzewa kodowego i wy-

soka efektywność kompresji sprawiły, iż algorytm ten stał się jednym z najczęściej wykorzystywanych kodów do kompresji danych różnego typu. Analiza własności kodu Huffmana pozwoliła naukowcom na rozwój wielu metod kompresji danych także dla źródeł o nieskończonej liczbie symboli. Do takich kodów należą metody uniwersalnego kodowania źródła [30] takie, jak np. kod Golomba, który służy do kodowania źródeł binarnych (szczegółowy opis własności kodu został przedstawiony w 8 rozdziale rozprawy). Golomb opisał zasadę tworzenia takiego kodu w 1966 roku w pracy [20], natomiast dokładną analizę tego kodu przeprowadzili w 1975 roku Gallager i Van Voorhis w pracy [15]. Jest to jeden z najpopularniejszych kodów dla źródeł o rozkładzie geometrycznym, często wykorzystywany do kompresji obrazu i dźwięku. W 1973 roku N. Faller oraz w 1978 roku Gallager przedstawili zasadę działania oraz analizę dynamicznej wersji kodu Huffmana [16]. Rozwinięcia tych propozycji powstały w latach osiemdziesiątych. Między innymi J. Vitter w pracy [43] zaprezentował zalety binarnego kodu Huffmana o minimalnej wariancji. Dokładny opis kodów dynamicznych znajduje się w rozdziale 7 rozprawy. Badania własności binarnego kodu Huffmana są prowadzone do dziś, czego najlepszym przykładem jest praca [35]. Wcześniej analizą redundancji oraz maksymalnej długości słów kodowych zajmowali się między innymi Gallager, Capocelli i De Santis, Manstetten, Katona, Buro [7, 9, 16, 24, 27]. Znacznie mniej publikacji odnosi się do wielowartościowych kodów Huffmana, gdyż dotyczą one głównie opracowań teoretycznych. Książka Abramsona z 1963 roku (polskie wydanie z roku 1969 [2]) przedstawia, obok binarnej i wielowartościowej wersji kodu Huffmana, także jego wersję rozszerzoną (rozszerzony kod Huffmana), która polega na tworzeniu makrosymboli. Kod rozszerzony pozwala uzyskać zwiększenie efektywności względem binarnej wersji kodu Huffmana. Propozycja ta ma jednak istotne wady (między innymi duże wymagania pamięciowe), które często uniemożliwiają jej praktyczne zastosowanie [29]. Nieco lepszy sposób tworzenia makrosymboli zaproponował w 1967 roku w swojej rozprawie doktorskiej Tunstall [36]. Wzrost efektywności po zastosowaniu jednocześnie kodów Tunstalla i Huffmana nie wymaga już tak dużej liczby makrosymboli, lecz i ta propozycja może być uciążliwa np. dla sprzętowej realizacji takiego algorytmu kodera. W ostatnich dziesięciu latach coraz popularniejsza staje się arytmetyczna metoda kompresji danych, po raz pierwszy opisana przez Abramsona już w roku 1963 w pracy [2], na podstawie pomysłu opracowanego przez Eliasa. Podobnie jak rozszerzenia kodu Huffmana, koder arytmetyczny pozwala na uzyskanie wysokiej efektywności kompresji. Niestety odbywa się to kosztem dużej złożoności implementacyjnej kodera a zwłaszcza dekodera (w porównaniu do binarnego kodu Huffmana). W pracy umieszczono dwie metody efektywnego zapisu w nagłówku pliku wynikowego pełnej wiedzy o liczności symboli źródła. Zostały one opracowane przez autora rozprawy na podstawie własnych badań i propozycji opisanych w pracy [6].

Podstawy autorskiej propozycji użycia *mieszanych kodów Huffmana* do uzyskania wysokiej efektywności kompresji omówione zostaną w 0 punkcie tego referatu. Zarówno koder, jak i dekodek mieszanego kodu Huffmana nie wymagają użycia dużego rozmiaru pamięci, co jest główną wadą rozszerzonych kodów Huffmana oraz kodu Tunstalla-Huffmana. Mieszane kody Huffmana wykorzystują symbole wielowartościowe, lecz ich stosowanie związane jest wyłącznie z binarnymi systemami zapisu i transmisji danych. Nie należy ich zatem kojarzyć z rozważaniami na temat optymalnych lub suboptymalnych kodów przedrostkowych, w których rozpatruje się możliwość zapisu słów kodu o poszczególnych cyfrach (sygnałach) charakteryzujących się różnymi kosztami [1, 18, 19].

2.1. Binarny kod Huffmana

W 1952 roku David Huffman przedstawił w pracy [22] algorytm i własności kodu, który jest optymalnym binarnym kodem przedrostkowym dla źródła S . Jest to kod optymalny wśród kodów binarnych, dla których każdy symbol c_i reprezentuje pojedynczy symbol źródła x_i . Poniżej przedstawiono własności binarnego kodu Huffmana dla źródła o alfabecie $S = \{x_1, x_2, \dots, x_n\}$, prawdopodobieństwach odpowiednio $P = \{p_1, p_2, \dots, p_n\}$ oraz długościach słów kodowych $\{l_1, l_2, \dots, l_n\}$:

- kod jest optymalny i kompletny, więc spełnia nierówność Krafta (która ma postać równości) oraz zależność (1.8);
- jeżeli $p_j > p_i$, to $l_j \leq l_i$;
- dwa symbole o najmniejszych prawdopodobieństwach mają słowa kodowe o tej samej, maksymalnej długości $l_{\max} = l_1 = l_2$, różniące się tylko ostatnim bitem.

Poniżej znajduje się algorytm tworzenia binarnego kodu Huffmana opracowany na podstawie [21, 32]:

- Utwórz listę n pojedynczych drzew z każdego symbolu źródła i ustaw te drzewa według niemalejącej kolejności odpowiadających im prawdopodobieństw (lub liczności);
- Dopóki istnieją co najmniej dwa drzewa:
 - pobierz dwa drzewa t_1 i t_2 o najmniejszych prawdopodobieństwach p_1 i p_2 oraz użyj ich jako potomków dla nowego drzewa (węzła) o prawdopodobieństwie $p_1 + p_2$;

2.2. Kod dwufazowy

W przypadku źródła o rozkładzie jednostajnym dane jest n równoprawdopodobnych zdarzeń elementarnych o $p_1 = p_2 = \dots = p_n = 1/n = p$. Symbole x_i tego kodu numerujemy od 0 do $n-1$ i w przypadku kodu o stałej długości są one zapisane jako słowa t bitowe, gdzie t jest liczbą całkowitą, taką że:

$$2^{t-1} < n \leq 2^t \quad (2.1)$$

Dla kodu o stałej długości, gdy n nie jest potęgą dwójki, kody o numerach od n do $2^t - 1$ nie są używane. Dla źródeł o rozkładzie jednostajnym można użyć binarnego kodu Huffmana, który nazywamy *kodem dwufazowym*, gdyż część symboli reprezentowana jest przez $t-1$ bitów (naturalna reprezentacja dla $x_i < 2^t - n$), a pozostała część przez t bitów w postaci liczby $x_i + 2^t - n$ [33]. Średnia długość kodu dwufazowego wynosi:

$$L_{sr} = t + 1 - \frac{2^t}{n} \quad (2.2)$$

Natomiast procentowa efektywność takiego kodu dana jest wzorem:

$$E = \hat{E} \cdot 100\% = \frac{\log_2 n}{t + 1 - \frac{2^t}{n}} \cdot 100\% \quad (2.3)$$

W przypadku bardzo dużej liczby symboli kodowych pojawia się problem związany z objętością przechowywanej w pamięci całej tabeli kodowej. Kod dwufazowy posiada proste w implementacji algorytmy kodowania i dekodowania, które zostały zaprezentowane w rozprawie. Nie wymagają one przechowywania w pamięci całej tabeli kodowej. Wystarczy jedynie znajomość liczby n symboli źródła. Dzięki tym własnościom kod dwufazowy ma duże zastosowanie praktyczne np. jako jeden z elementów kodów Golomba. W przypadku realizacji sprzętowej koder i dekodery wymagają jedynie operacji dodawania, odejmowania i przesunięcia bitowego. Szczegółową analizę własności tego kodu zawiera punkt 2.3 rozprawy.

2.3. Wielowartościowy kod Huffmana

Wielowartościowym (niebinarnym) kodem Huffmana nazywamy taki kod, w którym elementy kodu zapisywane są w r -narnym alfabecie dla $r > 2$ [29]. Ma on podobne własności do binarnego kodu Huffmana, przy czym w drzewie kodowym łączy się jednym węzłem r potomków. Może to jednak prowadzić do sytuacji niekompletności drzewa, w której jeden z węzłów będzie miał mniej niż r potomków. Jest to pierwszy węzeł tworzony z y symboli o najmniejszych prawdopodobieństwach, gdzie $2 \leq y \leq r$:

$$y = (n - 2) \bmod (r - 1) + 2 \quad (2.4)$$

Do tej pory wielowartościowe kody Huffmana nie znajdowały praktycznego zastosowania w metodach kompresji danych ze względu na niekompletność drzewa i małą efektywność zapisu cyfr r -narnych. Teoretyczne rozważania przedstawione zostały w wielu pracach takich jak [2, 8, 23, 25]. Praktyczne zastosowania dotyczyły jedynie kodów dla r będących potęgą dwójki [10].

2.4. Rozszerzony kod Huffmana

Jedną z podstawowych metod zwiększania efektywności kodu Huffmana jest tworzenie kodu rozszerzonego. Polega ona na łączeniu ze sobą k symboli źródła S w jeden makrosymbol (blok symboli). W rzeczywistości mamy do czynienia z k -krotnym rozszerzeniem źródła, a rozszerzenie kodu Huffmana jest tylko konsekwencją rozszerzenia źródła. Jeżeli źródło S składa się z n symboli, to rozszerzone źródło $S^{(k)}$ będzie zawierać $N_k = n^k$ słów kodowych *rozszerzonego alfabetu* (makrosymboli), będących zestawem wszystkich możliwych ciągów k -elementowych utworzonych z symboli pierwotnego źródła S . Dla takiego źródła $S^{(k)}$ tworzymy binarny kod Huffmana. Dla większych wartości k metoda staje się niepraktyczna ze względu na wykładniczy wzrost rozmiaru alfabetu, dla którego należy budować i przechowywać w pamięci n^k słów rozszerzonego kodu Huffmana [29]. Np. dla źródła o $n = 256$ symbolach przy $k = 5$ potrzebujemy aż $N_k = 2^{40}$ słów rozszerzonego kodu Huffmana. Jest to główna wada tej propozycji zwiększania efektywności binarnego kodu Huffmana.

Przykład 2.2

Dane jest trójsymbolowe źródło S o rozkładzie $P = \{0.04, 0.16, 0.8\}$, dla którego tworzymy binarny kod Huffmana. Średnia długość słowa kodu wynosi $L_{sr} = 0.8 + 2 \cdot (0.16 + 0.04) = 1.2$. Daje to efektywność zaledwie $E_2 = 72.19281\%$. Możemy utworzyć np. źródło $S^{(2)}$ składające się ze wszystkich możliwych par symboli, dla którego budujemy kod Huffmana (rozszerzony, składający się z 9 symboli). Dzięki temu uzyskujemy średnią $L_{sr}^{(2)} = 0.9112$ bita na symbol, co daje wzrost efektywności do $E^{(2)} = 95.07394\%$. Dla źródła $S^{(3)}$ zawierającego wszystkie trójki symboli uzyskujemy średnią długość słowa kodu Huffmana $L_{sr}^{(3)} = 0.87554$. Dla tego kodu składającego się z 27 makrosymboli efektywność wzrasta aż do $E^{(3)} = 98.94607\%$, co daje procentową wagę zysku równą $W = 97.23464\%$.

2.5. Kod Tunstalla i jego modyfikacje

Kod Tunstalla jest kodem przedrostkowym o stałej długości m bitowych słów kodowych. Jego budowa polega na łączeniu wielu symboli pierwotnego źródła w bloki (makrosymbole). W odróżnieniu od rozszerzonego kodu Huffmana, poszczególne makrosymbole mogą się składać z różnej liczby k_i symboli pierwotnego źródła [36]. Tworzenie takiego kodu wiąże się z dużym wzrostem liczby makrosymboli, czyli ze wzrostem zapotrzebowania na pamięć niezbędną do przechowywania słów kodowych i odpowiadających im ciągów symboli pierwotnego źródła. Stosowanie słów o stałej m bitowej długości jest główną zaletą tego kodu, ponieważ zapobiega to propagacji błędów np. podczas transmisji danych. Dla makrosymboli o zróżnicowanych prawdopodobieństwach nie jest to jednak dobrym rozwiązaniem pod względem efektywności. Rezygnacja z tej zalety dzięki dodatkowemu zastosowaniu kodu Huffmana dla makrosymboli źródła Tunstalla prowadzi do zwiększenia efektywności takiego kodu, który możemy określać mianem *kodu Tunstalla-Huffmana* [13]. Odwołując się do danych z przykładu 2.2 możemy utworzyć siedmioelementowy kod Tunstalla-Huffmana. Średnia długość słowa kodowego wynosi zaledwie $L_{sr} = 0.87344$, co odpowiada efektywności $E = 99.18382\%$ (waga zysku $W = 97.86359\%$).

2.6. Kod arytmetyczny i inne kody entropijne

Osobnym typem bezpamięciowych kodów entropijnych jest kod arytmetyczny, który nie należy do klasy kodów posiadających jednoznacznie dekodowalną książkę kodów [33]. Jest zatem określany mianem kodu nieblokowego, który jest jednak kodem jednoznacznie dekodowalnym w sensie potocznym [2]. W praktyce stosuje się całkowitoliczbową wersję kodu arytmetycznego, który koduje cały kompresowany ciąg danych do jednej liczby określanej mianem *znacznika* stanowiącego wynikowy ciąg bitów. Szczegółowe opisy algorytmu kodera i dekodera znajdują się w pracach [29, 45]. Przy założeniu precyzyjnego opisu liczności symboli w kodowanym ciągu uzyskujemy bardzo wysoką efektywność kompresji, przewyższającą efektywność kodu Huffmana. Algorytm kodera arytmetycznego jest bardziej skomplikowany od kodera binarnego kodu Huffmana, choć wymagania pamięciowe mają zbliżone. Ponadto zachowanie wysokiej efektywności wymaga użycia liczb o wysokiej precyzji (np. rejestrów źródłowych 32 bitowych), na których dokonuje się operacji dzielenia oraz mnożenia (rejestr wynikowy powinien więc być odpowiednio 64 bitowy). Główną wadą w porównaniu do metody Huffmana jest złożoność dekodera kodu arytmetycznego, który dla każdego dekodowanego elementu wymaga przeszukiwania tabeli skumulowanych licznosci oraz większej liczby operacji arytmetycznych niż koder. Stanowi to dużą wadę, ponieważ w przypadku archiwizacji najczęściej operacja kodowania wykonywana jest raz, a dekodowanie odbywa się wielokrotnie.

Biorąc pod uwagę często występujące sytuacje źródeł, w których dane stanowią zbiór elementów zależnych (kolejne elementy ciągu są skorelowane), możemy zwiększyć stopień kompresji przy użyciu algorytmów słownikowych [29]. Dwie podstawowe metody kompresji LZ77 i LZ78 zostały opisane przez Jacoba Ziva i Abrahama Lempela w pracach [46, 47]. Jednak dopiero kolejne modyfikacje tych algorytmów różniące się między innymi strukturą danych implementującą słownik pozwoliły uzyskiwać wysoki stopień kompresji danych (np. algorytm LZW używany przy kompresji obrazów typu *.gif*) [12]. Powszechne metody kompresji takie jak ARJ, Lharc, PKZip, RAR itp. bazują na algorytmach słownikowych lecz w połączeniu z innymi jednocześnie stosowanymi algorytmami modelowania danych oraz z kodem Huffmana lub kodem arytmetycznym.

3. Efektywna reprezentacja symboli wielowartościowych

W punkcie 2.3 wspomniano o problemie małej efektywności zapisu liczb r -narych (wielowartościowych) np. w pamięci komputera, czy na nośnikach danych, gdyż obecnie używana technika cyfrowa oparta jest o binarną reprezentację liczb. Ponieważ zakładamy, że wszystkie symbole źródła o numerach od 0 do

$r - 1$ mają równe prawdopodobieństwa występowania, to problem małej efektywności nie pojawia się jedynie dla r będących potęgą dwójki. W ogólności do zapisu jednej liczby r -narnej potrzeba $b(r)$ bitów na jej naturalny binarny zapis, gdzie $b(r)$ jest liczbą całkowitą taką, że:

$$\log_2 r \leq b(r) < \log_2 r + 1 \quad (3.1)$$

Procentowa efektywność zapisu wynosi wówczas:

$$E = \frac{\log_2 r}{b(r)} \cdot 100\% \quad (3.2)$$

Rozważmy najprostszy przykład dla $r = 3$. Do zakodowania jednej cyfry ternarnej potrzeba aż $b(3) = 2$ bitów, a więc procentowa efektywność wynosi zaledwie $E = 79.24813\%$. Posługując się kodem dwufazowym (patrz punkt 2.2), uzyskamy średnią długość jednej cyfry r -narnej:

$$\underline{b}(r) = b(r) + 1 - \frac{2^{b(r)}}{r} \quad (3.3)$$

Wówczas procentowa efektywność wynosi:

$$\underline{E} = \frac{\log_2 r}{\underline{b}(r)} \cdot 100\% \quad (3.4)$$

Co przy zakodowaniu pojedynczej cyfry ternarnej daje $\underline{E} = 95.09775\%$. Uzyskanie jeszcze większej efektywności jest możliwe dzięki użyciu zamiany bloków cyfr r -narnych na postać binarną. Możemy to wykonać tworząc liczbę B złożoną z q cyfr r -narnych:

$$B = \sum_{i=0}^{q-1} a_i \cdot r^i \quad (3.5)$$

Liczba B przyjmuje wartości od 0 do $r^q - 1$. Możemy przyjąć, że każda z tych wartości jest równoprawdopodobna, przy założeniu, że kolejne cyfry r -narne są równoprawdopodobne i niezależne. Do reprezentacji liczby B wykorzystujemy kod o stałej długości, a do jej zapisu wystarczy $b(r^q)$ bitów, gdzie $b(r^q) \leq q \cdot b(r)$. Np. $b(5) = 3$, natomiast $b(5^3) = 7 < 3 \cdot b(5) = 9$. Dla powyższej techniki grupowania q cyfr r -narnych możemy wyznaczyć średnią liczbę bitów $\bar{b}_q(r)$ potrzebną do zapisu pojedynczej cyfry r -narnej jako:

$$\bar{b}_q(r) = \frac{b(r^q)}{q} \quad (3.6)$$

Używając blokowej zamiany mamy do czynienia z kompresją, gdyż zmniejsza się średnia liczba bitów potrzebna do reprezentacji pojedynczej cyfry r -narnej. Wówczas wzór na procentową efektywność kompresji przyjmuje następującą postać:

$$E_{r(q)} = \frac{\log_2 r}{\bar{b}_q(r)} \cdot 100\% \quad (3.7)$$

Rozpatrzmy zatem liczbę B złożoną z $q = 5$ cyfr ternarnych $\{a_0, a_1, a_2, a_3, a_4\}$:

$$B = a_0 \cdot 3^0 + a_1 \cdot 3^1 + a_2 \cdot 3^2 + a_3 \cdot 3^3 + a_4 \cdot 3^4 \leq 242 < 2^8$$

Wtedy do zapisu liczby B wystarczy $b(3^5) = 8$ bitów. Daje to średnią liczbę bitów potrzebną do reprezentacji pojedynczej cyfry ternarnej równą $\bar{b}_5(3) = 1.6$, co oznacza, że efektywność wzrasta do $E_{3(5)} = 99.06016\%$. Dla dalszego wzrostu efektywności kompresji (efektywności reprezentacji cyfr r -narnych) możemy zastosować kod dwufazowy dla liczby B , która nie jest potęgą dwójki. Pozwoli to na zmniejszenie średniej długości jednej cyfry r -narnej do:

$$\underline{b}_q(r) = \frac{\underline{b}(r^q)}{q} = \frac{b(r^q) + 1 - \frac{2^{b(r^q)}}{r^q}}{q} \quad (3.8)$$

Otrzymujemy kod o zmiennej długości symboli kodowych (słowa kodu mają $b(r^q)$ lub $b(r^q) - 1$ bitów), dla którego wzór na zmodyfikowaną procentową efektywność kompresji przyjmuje postać:

$$\underline{E}_{r(q)} = \frac{\log_2 r}{\underline{b}_q(r)} \cdot 100\% \tag{3.9}$$

Dla liczby B złożonej z $q = 5$ cyfr ternarnych, dzięki użyciu kodu dwufazowego, średnia długość jednej cyfry ternarnej zmniejsza się do $\underline{b}_5(3) = 1.5893$ bita, natomiast efektywność wzrasta do $\underline{E}_{3(5)} = 99.72706\%$.

3.1. Algorytm wyznaczania wzorcowych długości bloków (q) elementów r-narnych

W rozdziale 3 rozprawy przedstawiono szczegółową analizę efektywności konwersji liczb r -narnych, na podstawie której został następnie zaprezentowany algorytm wyznaczania wzorcowych wartości q . Najlepszy rezultat uzyskamy, jeśli będziemy używać wzorcowych wartości q dla liczb B , a następnie stosować kod dwufazowy do kompresji liczby B . W tabeli 3.1 znajdują się parametry dla przykładowych wzorcowych par (r_i, q_i) takich, że $r < 30$ [38], a także porównanie efektywności kodów o stałej i zmiennej długości. Dzięki zastosowaniu odpowiednich wzorcowych wartości q oraz kodu dwufazowego dla liczb B łatwo uzyskujemy procentowe efektywności kompresji $\underline{E}_{r(q)} > 99.7\%$.

Tabela 3.1 Parametry dla przykładowych wzorcowych q przy $r < 30$

r	q	$b(r^q)$	$\underline{b}(r)$	$\underline{E} [\%]$	$\bar{\underline{b}}_q(r)$	$E_{r(q)} [\%]$	$\underline{b}_q(r)$	$\underline{E}_{r(q)} [\%]$
3	41	65	$1\frac{2}{3}$	95.09775	$1\frac{24}{41}$	99.97456	1.5851	99.99229
5	31	72	$2\frac{2}{5}$	96.74700	$2\frac{10}{31}$	99.97190	2.3221	99.99151
7	16	45	$2\frac{6}{7}$	98.25742	$2\frac{13}{16}$	99.81706	2.8088	99.94749
11	13	45	$3\frac{6}{11}$	97.57371	$3\frac{6}{13}$	99.93914	3.4601	99.98172
13	17	63	$3\frac{10}{13}$	98.17493	$3\frac{12}{17}$	99.85313	3.7012	99.95823
17	11	45	$4\frac{2}{17}$	99.26696	$4\frac{1}{11}$	99.91576	4.0885	99.97491
19	4	17	$4\frac{6}{19}$	98.42759	$4\frac{1}{4}$	99.95125	4.2486	99.98513
23	13	59	$4\frac{14}{23}$	98.15276	$4\frac{7}{13}$	99.67170	4.5274	99.91504
29	8	39	$4\frac{26}{29}$	99.21229	$4\frac{7}{8}$	99.65089	4.8626	99.90442

4. Mieszane kody Huffmana

Dzięki efektywnej reprezentacji symboli r -narnych możliwa jest efektywna realizacja *mieszanych kodów Huffmana* [34, 39, 40]. Konstrukcja drzew kodowych dla tych kodów polega na łączeniu w każdym z kolejno tworzonych węzłów dowolnej odpowiednio dobranej liczby r_i najmniej prawdopodobnych potomków tak, aby zachowana została własność rodzeństwa. *Skrótowny opis konstrukcji* takiego drzewa można przedstawić jako ciąg kroków, w których w każdym tworzymy węzeł w_i łączący r_{w_i} potomków:

$r_w = \{r_{w_1}, r_{w_2}, r_{w_3}, \dots\}$. W praktyce efektywne kodowanie tą metodą można uzyskać dzięki *stosowaniu osobnych buforów* dla każdej wartości r_i , do których są zapisywane r_i -narne elementy składowe słów kodu mieszanego. Kod mieszany ma tę własność, że w odróżnieniu od kodów wielowartościowych jest *zawsze kodem kompletnym* (żaden z węzłów nie może posiadać nieużywanych potomków). *Przykładowa struktura pliku* dla takiego kodu składa się z nagłówka zawierającego zestaw licznosci, następnie znajduje się wskaźnik długości bufora binarnego, po nim jest umieszczony cały bufor binarny, a na końcu bufor ternarny w postaci ciągu liczb B . Gdy istnieją inne bufory r -narne, to umieszcza się je kolejno za buforem ternarnym wraz ze wskaźnikami długości każdego bufora. Istnieje też możliwość użycia kodera, który nie musi oczekiwać, aż na wejściu pojawi się informacja o końcu kodowanego pliku. Wystarczy, aby każdy z używanych buforów r_i -narnych zapełnił się q_i cyframi. Wówczas można zakodować pierwszy zestaw liczb B_i i wysłać je na wyjście.

Przykład 4.1

Dane jest ośmiosymbolowe źródło S o rozkładzie przedstawionym w tabeli 4.1, dla którego tworzymy binarno-ternarny kod Huffmana. Wartość entropii tego źródła wynosi $H(S) = 2.68973$ bita na symbol.

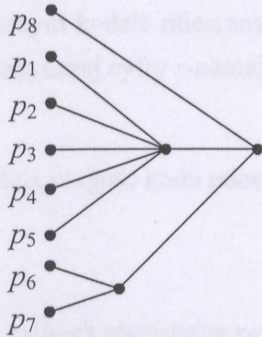
Średnia długość binarnego kodu Huffmana wynosi $L_2 = 2.76$ co daje procentową efektywność $E_2 = 97.45394\%$, natomiast dla kodu binarno-ternarnego przy $q = 5$ średnia wynosi $L_{2,3} = 2.714$, co odpowiada efektywności $E_{2,3} = 99.10570\%$. Drzewo kodu binarno-ternarnego można opisać skrótowo w następujący sposób: $r_w = \{3, 2, 2, 2, 3\}$. Jeżeli dla tego samego źródła użyjemy kodu o trzech buforach, tzn. oprócz kodu binarnego ($r_0 = 2$) zastosujemy kod ternarny ($r_1 = 3$) z parametrem $q_1 = 5$ oraz pięciowartościowy ($r_2 = 5$) z parametrem $q_2 = 3$, to otrzymamy jeszcze wyższą efektywność. Drzewo kodowe o skrótowym zapisie $r_w = \{5, 2, 3\}$ przedstawia rysunek 4.1 [41]. Średnia długość takiego kodu mieszanego wynosi

$$L_{mix} = L_{2,3,5} = \frac{8}{5} + \frac{7}{3} \cdot \frac{6+6+7+7+7}{100} + \frac{17+17}{100} = 2.71. \text{ Wyliczenia } L_{mix} \text{ oparte na podstawie drzewa}$$

kodu mieszanego uwzględniają długości gałęzi, czyli wartości oznaczające średnią liczbę bitów $\bar{b}_q(r)$ potrzebną do zakodowania jednej cyfry r -narnej (przy drzewie binarnym długość gałęzi nie była brana pod uwagę, ponieważ jej wartość wynosiła 1). Dla takiego kodu otrzymujemy procentową efektywność równą $E_{mix} = E_{2,3,5} = 99.25198\%$. Jeżeli zwiększymy parametry wzorcowe konwersji binarnej odpowiednio do $q_1 = 41$ oraz $q_2 = 31$, to otrzymamy średnią $L_{mix} = 2.69182$, co daje wzrost efektywności do $E_{mix} = 99.92240\%$. Waga zysku wynosi wówczas $W = 97.02761\%$, co oznacza znaczne zbliżenie się średniej L_{mix} do wartości entropii w porównaniu z wartością L_2 . Tabela 4.1 zawiera symbole dla kodu binarno-ternarnego, kodu mieszanego przy $r_1 = 3, r_2 = 5$, a także binarnego kodu Huffmana. Jak widać z powyższego przykładu właściwy dobór par wzorcowych parametrów (r_i, q_i) wpływa na uzyskanie odpowiednio wysokiej efektywności E_{mix} mieszanego kodu Huffmana.

Tabela 4.1 Kody Huffmana z przykładu 4.1

p_i	Kod binarno-ternarny	Kod mieszany $r_1 = 3, r_2 = 5$	Kod binarny
$p_1 = 0.06$	$2_3 1_2 0_3$	$1_3 0_5$	1110
$p_2 = 0.06$	$2_3 1_2 1_3$	$1_3 1_5$	1111
$p_3 = 0.07$	$2_3 1_2 2_3$	$1_3 2_5$	0000
$p_4 = 0.07$	$0_3 0_2 0_2$	$1_3 3_5$	0001
$p_5 = 0.07$	$0_3 0_2 1_2$	$1_3 4_5$	110
$p_6 = 0.17$	$0_3 1_2$	$2_3 0_2$	001
$p_7 = 0.17$	$2_3 0_2$	$2_3 1_2$	10
$p_8 = 0.33$	1_3	0_3	01



Rysunek 4.1 Drzewo mieszanego kodu Huffmana z przykładu 4.1

Jedną z dodatkowych cech kodu mieszanego jest fakt, iż dla tego samego źródła i tych samych buforów r_i mogą istnieć drzewa optymalnego kodu mieszanego o różnych kształtach w zależności od zastosowanych wzorcowych parametrów q_i . Dlatego zachodzi potrzeba sprecyzowania ograniczeń konstrukcyjnych danego kodu:

Definicja 4.1 Konstrukcyjną klasą kodów mieszanych nazywamy zbiór kodów, dla których koder i dekodek spełnia określone ograniczenia konstrukcyjne. Do ograniczeń tych zaliczamy zestaw parametrów wykorzystywanych do utworzenia optymalnego kodu mieszanego, czyli liczbę m używanych buforów r_i -narych, zestaw odpowiadających im parametrów $r_0, r_1, \dots, r_{m-1}$ i związanych z nimi wzorcowych wartości $q_1, q_2, \dots, q_{m-1}$ (oraz $b(r_i^{q_i})$), a także opcjonalne parametry potrzebne podczas dekodowania.

Do parametrów opcjonalnych można zaliczyć skrótowy opis konstrukcji drzewa r_w , (co pozwala pominąć etap poszukiwania optymalnego w danej klasie drzewa podczas etapu dekompresji), a także to, czy zasto-

sowano kod dwufazowy do kompresji liczb B . Jeśli np. zdefiniujemy klasę kodów mieszanych dla przykładu 4.1 w następujący sposób: „ $m = 3, r_0 = 2, r_1 = 3, r_2 = 5, q_1 = 41, q_2 = 31$, używany kod dwufazowy”, to otrzymamy średnią $L_{mix} = 2.69139$ bita na symbol oraz efektywność takiego kodu równą $E_{mix} = 99.93843\%$. Na podstawie cech kodu opisanych w pracy [44]:

Definicja 4.2 *Idealną klasą kodu mieszane* nazywamy klasę kodów bez ograniczeń konstrukcyjnych. Dla takiej klasy istotny jest jedynie zestaw parametrów r_i określających taki kod, w którym do reprezentacji pojedynczej cyfry r_i -narnej wystarcza $\log_2 r_i$ bitów.

Pomijając przypadki źródeł, dla których prawdopodobieństwa są postaci 2^{-l} , dla całkowitych dodatnich wartości l , klasa idealna jest pojęciem teoretycznym i stanowi granicę możliwości uzyskania wzrostu efektywności przez kod mieszany o zdefiniowanym zestawie m buforów r_i -narnych. Jeżeli np. zdefiniujemy idealną klasę kodów mieszanych dla przykładu 4.1 jako „ $m = 3, r_0 = 2, r_1 = 3, r_2 = 5$ ”, to teoretyczna procentowa efektywność kodu mieszane

4.1. Podstawowe własności mieszanych kodów Huffmana

Posługując się definicją klasy kodów można uszczegółowić pojęcie kodu mieszane

Definicja 4.3 Dla źródła S o n symbolach *optymalnym w danej klasie mieszany* kodem Huffmana jest kod o najkrótszej średniej długości słowa kodowego L_{mix} , dla którego każdy z węzłów drzewa kodowego nie będący liściem może łączyć dowolną określoną w danej klasie kodów liczbę $2 \leq r_i \leq n$ potomków.

W przypadku klasy bez ograniczeń (klasy idealnej) możemy mówić o *optymalnym kodzie mieszany*. Realizacja kodera i dekodera kodu mieszane

$$g(r) = \sum_{j=1}^{n_r} p_{w_j} \quad (4.1)$$

Waga $w(r)$ cyfry r -narnej w danym kodzie mieszany

$$w(r) = g(r) \cdot \bar{b}_q(r) \quad (4.2)$$

Suma wszystkich wag daje średnią długość kodu mieszane

$$L_{mix} = \sum_{i=0}^{m-1} w(r_i) \quad (4.3)$$

Gdzie m jest liczbą wszystkich różnych elementów r_i -narnych występujących w danym kodzie mieszany. Kod mieszany określamy mianem *mieszane* kodu Huffmana na ze względu na duże podobieństwa własności do binarnego kodu Huffmana:

- kod mieszany jest kodem przedrostkowym, gdyż można go przedstawić w postaci drzewa kodowego, którego liście odpowiadają poszczególnym symbolom źródła;
- jeżeli $p_j > p_i$, to $l_j \leq l_i$;
- dwa symbole (lub r_i symboli) o najmniejszych prawdopodobieństwach mają słowa kodowe o tej samej, maksymalnej długości l_{max} i różnią się tylko ostatnim bitem (lub cyfrą r_i -narną, gdy węzeł drzewa nie będący liściem łączy ze sobą r_i symboli źródła o najniższym prawdopodobieństwie);
- mając dany skrótowy opis drzewa kodu mieszane $r_w = \{r_{w_1}, r_{w_2}, r_{w_3}, \dots\}$ łączymy w każdym i -tym kroku r_{w_i} symboli lub węzłów o aktualnie najmniejszym prawdopodobieństwie według algorytmu Huffmana (dla kodu binarnego r_{w_i} było zawsze równe dwa);
- kod mieszany jest zawsze kodem kompletnym.

4.2. Testy efektywności

Dla klasy zdefiniowanej w tabeli 3.1 (bez użycia kodu dwufazowego), a także dla klasy idealnej dokonano pomiarów wpływu liczby n elementów źródła na procentowy udział binarnego kodu Huffmana we wszystkich testach, które polegały na znalezieniu optymalnego w danej klasie mieszanego kodu Huffmana. Każde badanie wykonano dla $2^{17} = 131072$ testowych źródeł o n symbolach. Rozkład prawdopodobieństwa dla testowych źródeł był generowany w sposób pseudolosowy (generator *Rand* języka C o w przybliżeniu jednostajnym rozkładzie). Procentowe wyniki testów dla $n \leq 15$, w których binarny kod Huffmana uzyskał najkrótszą średnią długość słowa kodowego L_{mix} wśród wszystkich kodów mieszanych przedstawia tabela 4.2. Na podstawie wyników widać, że wartości dla klasy rzeczywistej i idealnej różnią się w niewielkim stopniu. Uzasadnia to, iż ograniczenie klasy kodów do zbioru wartości r_i będących liczbami pierwszymi jest dobrą propozycją uproszczenia konstrukcji kodu mieszanego. Dla $n \geq 12$ przypadki uzasadnionego użycia binarnego kodu Huffmana (jako optymalnego kodu mieszanego) nie stanowią nawet jednego procenta źródeł testowych. Dla wyższych wartości n pomiar nie jest już wystarczająco precyzyjny. Mimo zwiększenia liczby testów do 300000 dla $n = 20$ nie uzyskano ani jednego przypadku, w którym użycie kodu binarnego byłoby uzasadnione. Na podstawie tych wyników możemy wywnioskować, że wzrost efektywności uzyskany dla kodu mieszanego (względem binarnego kodu Huffmana) jest zjawiskiem powszechnym zwłaszcza w sytuacjach praktycznych, w których liczba n symboli źródła często znacznie przekracza wartość 20.

Tabela 4.2 Wyniki testów kodów mieszanych

Liczba n symboli źródła	Dla klasy idealnej [%]	Dla klasy testowej [%]
3	82.615662	82.667542
4	47.846985	47.922516
5	39.065552	39.154816
6	25.956726	26.042938
7	13.333893	13.404083
8	5.573273	5.610651
9	2.482605	2.515411
10	1.639557	1.671600
11	1.264954	1.302338
12	0.832367	0.855255
13	0.453186	0.462341
14	0.193787	0.198364
15	0.062561	0.063324

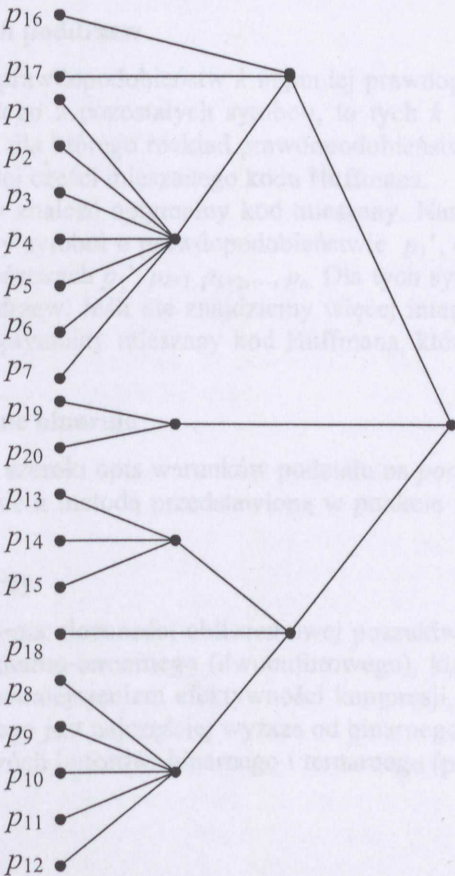
Przykład 4.2

Dane jest źródło S składające się z $n = 20$ symboli o rozkładzie przedstawionym w tabeli 4.3, w której znajdują się także zestawy słów binarnego, binarno-ternarnego i mieszanego kodu Huffmana. Rysunek 4.2 przedstawia kształt drzewa dla mieszanego kodu Huffmana o skrótowym opisie $r_w = \{7, 5, 3, 3, 3, 2, 3\}$. Średnia długość słowa kodu binarnego wynosi $L_2 = 3.762$, co daje procentową efektywność na poziomie $E_2 = 98.95126\%$. Jeżeli zamiast kodu binarnego użyjemy kodu mieszanego z daną klasą: $r_0 = 2, r_1 = 3, r_2 = 5, r_3 = 7, q_1 = 41, q_2 = 31, q_3 = 16$, wtedy $L_{mix} = 0.333 + 1.778 \cdot \bar{b}_q(3) + 0.112 \cdot \bar{b}_q(5) + 0.111 \cdot \bar{b}_q(7)$. Dla klasy zdefiniowanej w przykładzie daje to średnią $L_{mix} = L_{2,3,5,7} = 3.7241$, co odpowiada efektywności $E_{mix} = 99.95836\%$ (waga zysku $W = 96.06975\%$). Jeżeli dodatkowo użyjemy kodu dwufazowego do kompresji liczb B , to średnia długość słowa kodowego zmniejszy się do $\underline{L}_{mix} = 3.72314$ i uzyskamy wówczas efektywność równą $\underline{E}_{mix} = 99.98409\%$. W przypadku klasy idealnej dla kodu mieszanego uzyskamy średnią długość słowa kodowego równą $L_{mix} = 3.72274$ oraz teoretyczną efektywność $E_{mix} = 99.99492\%$. Ograniczenie kodu mieszanego do binarno-ternarnego kodu Huffmana z daną klasą: $r_0 = 2, r_1 = 3, q_1 = 41$ spowoduje tylko nieznaczny wzrost średniej długości kodu mieszanego równej $L_{2,3} = 1.464 + 1.427 \cdot \bar{b}_q(3)$. Dla zdefiniowanej wyżej klasy daje to średnią $L_{2,3} = 3.72632$, co odpowiada efektywności $E_{2,3} = 99.89881\%$. Jeżeli dodatkowo użyjemy kodu dwufazowego do kompresji liczb B , to średnia długość słowa kodowego zmniejszy się do $\underline{L}_{2,3} = 3.72592$ i uzyskamy wówczas efektywność równą $\underline{E}_{2,3} = 99.90957\%$.

W przypadku klasy idealnej dla kodu binarno-ternarnego uzyskamy średnią długość słowa kodowego równą $L_{2,3} = 3.72574$ oraz teoretyczną efektywność $E_{2,3} = 99.91424\%$.

Tabela 4.3 Kody Huffmana z przykładu 4.2

p_i	Kod mieszany	Kod binarno-ternarny	Kod binarny
$p_1 = 0.015$	$0_3 2_3 0_7$	$2_3 1_2 0_2 0_3 0_2$	001100
$p_2 = 0.015$	$0_3 2_3 1_7$	$2_3 1_2 0_2 0_3 1_2$	001101
$p_3 = 0.016$	$0_3 2_3 2_7$	$2_3 1_2 0_2 1_3 0_2$	001110
$p_4 = 0.016$	$0_3 2_3 3_7$	$2_3 1_2 0_2 1_3 1_2$	001111
$p_5 = 0.016$	$0_3 2_3 4_7$	$2_3 1_2 0_2 2_3 0_2$	101000
$p_6 = 0.016$	$0_3 2_3 5_7$	$2_3 1_2 0_2 3_3 1_2$	101001
$p_7 = 0.017$	$0_3 2_3 6_7$	$0_3 0_2 1_2 1_2 0_2$	000000
$p_8 = 0.022$	$2_3 2_3 0_5$	$0_3 0_2 1_2 1_2 1_2$	000001
$p_9 = 0.022$	$2_3 2_3 1_5$	$2_3 1_2 1_2 0_2 0_2$	000010
$p_{10} = 0.022$	$2_3 2_3 2_5$	$2_3 1_2 1_2 0_2 1_2$	000011
$p_{11} = 0.023$	$2_3 2_3 3_5$	$2_3 1_2 1_2 1_2 0_2$	000100
$p_{12} = 0.023$	$2_3 2_3 4_5$	$2_3 1_2 1_2 1_2 1_2$	000101
$p_{13} = 0.037$	$2_3 0_3 0_3$	$0_3 0_2 0_2 0_2$	101010
$p_{14} = 0.037$	$2_3 0_3 1_3$	$0_3 0_2 0_2 1_2$	101011
$p_{15} = 0.037$	$2_3 0_3 2_3$	$0_3 0_2 1_2 0_2$	101111
$p_{16} = 0.110$	$0_3 0_3$	$1_3 0_3$	010
$p_{17} = 0.111$	$0_3 1_3$	$1_3 1_3$	011
$p_{18} = 0.112$	$2_3 1_3$	$1_3 2_3$	100
$p_{19} = 0.166$	$1_3 0_2$	$0_3 1_2$	110
$p_{20} = 0.167$	$1_3 1_2$	$2_3 0_2$	111



Rysunek 4.2 Drzewo mieszanego kodu Huffmana z przykładu 4.2

5. Konstrukcja optymalnych mieszanych kodów Huffmana

Algorytm budowy optymalnego drzewa kodu mieszanego w danej klasie nie jest tak prosty jak w przypadku binarnego kodu Huffmana, gdyż także elementy o wyższych prawdopodobieństwach ($p_{r+1}, \dots, p_n$) decydują o tym, czy opłaca się połączenie elementów $p_1, \dots, p_r$ jednym węzłem o r potomkach. Ponadto mamy do czynienia z wieloma różnymi wartościami r_i , które mogą być brane pod uwagę, co także zwiększa liczbę wszystkich możliwych do uzyskania mieszanych drzew Huffmana. Algorytm znalezienia optymalnego w danej klasie mieszanego kodu Huffmana wymaga porównania średniej długości słowa kodowego wszystkich drzew mieszanych możliwych do utworzenia z n elementów (*metoda porównań*). Przy czym tworząc nowy węzeł w należy zachować zasadę sortowania prawdopodobieństw (podobnie jak w algorytmie binarnego kodu Huffmana) i łączyć r_i najmniej prawdopodobnych symboli lub węzłów. Liczba porównań D jest w przybliżeniu zależnością wykładniczą o współczynniku około 1.794. Poniższy wzór na przybliżoną wartość D został wyznaczony dla $n \leq 25$:

$$D \approx 33 \cdot (1.794)^{n-8} \quad (5.1)$$

Praktyczna realizacja algorytmu poszukiwania optymalnego mieszanego kodu Huffmana nie wymaga tworzenia od początku każdego drzewa, tak aby wyliczyć średnią długość słowa kodowego. Jest to autorska propozycja tzw. szybkiej rekurencji z parametrami globalnymi, dzięki czemu nie trzeba przekazywać parametrów do wywoływanej rekurencyjnie funkcji. W ten sposób aktualnie wyznaczany wynik L_{mix} częściowo zależy od poprzednio wyliczonych wartości składowych. Program napisany w języku C testowany na komputerze klasy Pentium III 500 MHz wykonywał 406222 porównania na sekundę (pomiar wykonany dla $n = 20$). Co oznacza, że na podstawie zależności (5.1) już dla $n = 38$ symboli potrzebowalibyśmy około godziny, aby znaleźć optymalne drzewo kodowe. Jest to główna wada tej propozycji poszukiwania optymalnych mieszanych kodów Huffmana (w odniesieniu do wydajności obecnie stosowanych procesorów przy $n \leq 256$). Kod źródłowy głównych funkcji tego programu znajduje się w dodatku B rozprawy. Autor zaprezentował kilka metod pozwalających na zmniejszanie liczby drzew, dla których należy porównać średnie długości słów kodu mieszanego L_{mix} , aby wyznaczyć optymalny w danej klasie mieszany kod Huffmana.

5.1. Metoda integralnych poddrzew

Twierdzenie 5.1 Jeżeli suma prawdopodobieństw k najmniej prawdopodobnych symboli jest nie większa od prawdopodobieństwa każdego z pozostałych symboli, to tych k symboli może być traktowane jako osobne *integralne poddrzewo*, dla którego rozkład prawdopodobieństw występujących w nim symboli nie ma wpływu na kształt pozostałej części mieszanego kodu Huffmana.

Dla takiego poddrzewa należy znaleźć optymalny kod mieszany. Następnie oznaczamy korzeń tego integralnego poddrzewa jako nowy symbol o prawdopodobieństwie p_1' , co daje nam nowy zestaw symboli o niemalejących prawdopodobieństwach $p_1', p_{k+1}, p_{k+2}, \dots, p_n$. Dla tych symboli ponownie próbujemy zastosować metodę integralnych poddrzew. Jeśli nie znajdziemy więcej integralnych poddrzew, to z pozostałych na liście symboli tworzymy optymalny mieszany kod Huffmana, który następnie łączymy z integralnymi poddrzewami.

5.2. Poddrzewa integralne binarnie

W rozprawie znajduje się szeroki opis warunków podziału na *poddrzewa integralne binarnie*. Metoda ta powinna być stosowana łącznie z metodą przedstawioną w punkcie 5.1. Ich skuteczność jest zależna od rozkładu źródła S .

5.3. Kod binarno-ternarny

Inną propozycją zmniejszenia złożoności obliczeniowej poszukiwania optymalnego kodu mieszanego jest użycie wyłącznie kodu binarno-ternarnego (dwubuforowego), który jest podzbiorem kodów mieszanych. Może się to wiązać ze zmniejszeniem efektywności kompresji, lecz taki kod ma wiele zalet. Efektywność kodu binarno-ternarnego jest najczęściej wyższa od binarnego kodu Huffmana, a praktyczna realizacja wymaga użycia tylko dwóch buforów: binarnego i ternarnego (patrz przykład 4.2). Liczba porównań D zmniejsza się wówczas do:

$$D \approx (1.618)^{n-1} \quad (5.2)$$

i stanowi wartość wyrazu ciągu Fibonacciego. Omawiany na początku punktu 5 program potrzebował około godziny (na komputerze klasy Pentium III 500 MHz), aby wyznaczyć optymalny kod mieszany dla źródła

dla $n = 38$ symbolach. W przypadku kodu binarno-ternarnego jest to niecałe 97 sekund. Ale już dla $n = 44$ symboli czas zwiększa się do prawie 30 minut. Zatem dla źródeł o dużej liczbie symboli nawet algorytm poszukiwania optymalnego kodu binarno-ternarnego okazuje się mało praktyczny. W takich przypadkach należy stosować suboptymalne mieszane kody Huffmana.

6. Suboptymalne mieszane kody Huffmana o krótkim czasie konstrukcji drzewa kodowego

Ze względu na dużą liczbę iteracji potrzebnych do utworzenia optymalnego drzewa kodu mieszanego dla większych wartości n może pojawić się problem, jeśli użycie metod podziału na poddrzewa integralne i poddrzewa integralne binarnie nie daje podziału ma wystarczająco małe poddrzewa o np. $n_i < 20$ elementach. Można wówczas poszukiwać suboptymalnych drzew. Pierwszą propozycją jest tzw. *kod dwupoziomowy*. Tworzymy binarny kod Huffmana do momentu, aż pozostanie tylko $N < n$ elementów (węzłów lub symboli źródła), które stają się zbiorem symboli źródła $S_0 = \{w_1, w_2, \dots, w_N\}$ o prawdopodobieństwach $P_{w_1}, P_{w_2}, \dots, P_{w_N}$. Z elementów w_i konstruuje się tzw. *poddrzewo główne* (poddrzewo pierwszego poziomu), czyli optymalny w danej klasie kod mieszany o N symbolach. Jeśli dany symbol w_i posiada potomków (w utworzonym wcześniej binarnym kodzie Huffmana), to jest on korzeniem *poddrzewa drugiego poziomu*, które jednak nie może posiadać więcej niż N liści, czyli symboli pierwotnego źródła. Dla każdego z tych $k \leq N$ poddrzew należy znaleźć optymalny mieszany kod Huffmana. Dzięki rozbiciu na dwa poziomy drzew możliwe jest szybkie uzyskanie suboptymalnego kodu. Minimalnie może on kodować $n_{\min} = 2 \cdot N - 1$ symboli dla drzewa Huffmana będącego odpowiednikiem kodu unarnego. Maksymalnie kod dwupoziomowy może kodować $n_{\max} = N^2$ symboli, gdy są one równoprawdopodobne i jednocześnie N jest potęgą dwójki. Jeśli np. $N = 13$, to wystarczy maksymalnie $D' = (N+1) \cdot D_{(N)} = 14 \cdot 616 = 8624$ iteracji poszukiwania suboptymalnego drzewa mieszanego. Jednocześnie $n_{\min} = 2 \cdot 13 - 1 = 25$, zatem w przypadku poszukiwania optymalnego kodu mieszanego metodą porównań potrzebowalibyśmy co najmniej $D_{(25)} = 676764$ iteracji. Drugą ważną zaletą kodu dwupoziomowego jest mała liczba buforów r -narnych, ponieważ $r \leq N$.

6.1. Metoda kolejnych uściśleń

Metoda kolejnych uściśleń wykorzystuje zasadę podobną do kodu Shannona-Fano [33], gdzie rekurencyjnie dokonuje się podziału zbioru prawdopodobieństw na dwie prawie równe części. W tym przypadku możliwy jest podział na r prawie równych części. Dzięki wykorzystaniu algorytmu binarnego kodu Huffmana do budowy poddrzewa o $n - r$ liściach uzyskujemy najszybszą z przedstawianych w pracy metod uzyskania kodu mieszanego i nawet dla źródeł o 256 symbolach czas utworzenia drzewa jest mierzony w milisekundach (dla procesora Intel Pentium III 500 MHz). Niestety najczęściej wiąże się to z jedynie niewielkim wzrostem efektywności w porównaniu do binarnego kodu Huffmana.

6.2. Kod dwupoziomowy z wykorzystaniem metody kolejnych uściśleń

Propozycja użycia kodu dwupoziomowego może okazać się nieskuteczna, gdy źródło ma zbyt dużą liczbę symboli n i wówczas część poddrzew drugiego poziomu może zawierać większą liczbę symboli niż dopuszczalne N . Można zaproponować połączenie dwóch przedstawionych wcześniej metod. Najpierw tworzymy binarny kod Huffmana do momentu, aż pozostanie tylko $N < n$ elementów (węzłów lub symboli źródła). Stają się one zbiorem symboli źródła $S_0 = \{w_1, w_2, \dots, w_N\}$ o prawdopodobieństwach $P_{w_1}, P_{w_2}, \dots, P_{w_N}$, z których tworzymy poddrzewo główne (kod optymalny w danej klasie kodów mieszanych). Podobnie jak w przypadku kodu dwupoziomowego symbole $w_1, w_2, \dots, w_N$ stanowią zestaw N elementów, na które mogą składać się korzenie poddrzew drugiego poziomu lub symbole pierwotnego źródła o odpowiednio wysokich prawdopodobieństwach. Dla każdego z tych poddrzew budujemy suboptymalny kod w oparciu o metodę kolejnych uściśleń. Połączenie tych propozycji możemy określić mianem *rozszerzonego kodu dwupoziomowego*. Autor rozprawy proponuje, aby ustalić wartość $N = 23$ z buforami r -narnymi ze zbioru $r_i = \{2, 3, 5, 7, 11, 13, 17, 19, 23\}$. Dla takiej wartości N do utworzenia suboptymalnego drzewa kodu mieszanego dla źródła o $n = 256$ symbolach wystarczy niecała sekunda (w przypadku procesora Pentium III 500 MHz). W tabeli 6.1 znajduje się porównanie średnich długości słów kodowych dla binarnego kodu Huffmana L_2 , metody kolejnych uściśleń L_{mku} oraz dla rozszerzonego kodu dwupoziomowego L_{rkd} . Wyniki uzyskano na podstawie badań dwunastu plików z danymi różnego typu. Średnia długość słowa kodowego nie uwzględnia nagłówka pliku, tak aby wielkość kompresowanego pliku nie miała wpływu na porównywanie efektywności. Na początku testowane były pliki multimedialne zarówno te bez wcześniejszej kompresji (BMP, WAV) jak i skompresowane (JPG, MP3). Druga część testów to różne odmiany dokumentów tekstowych: DOC, PS, PDF, CPP, DJVU – (format DeJaVu wykorzystuje kod entropijny do kompresji danych, stąd zastosowanie binarnego kodu Huffmana nie spowodowało zmniejszenia

średniej długości słowa kodowego). W ostatniej części testów znalazły się: EXE – program wykonywalny realizujący kompresję metodą Huffmana, PPT – prezentacja PowerPoint, a także XLS – arkusz Excela, dla którego niska efektywność kompresji wynika z dużej wartości najbardziej prawdopodobnego symbolu $p_{\max} = 0.62469$. Metoda kolejnych uściśleń okazała się lepsza od binarnego kodu Huffmana jedynie w czterech przypadkach. Natomiast dla wszystkich dwunastu testów najlepszy rezultat uzyskiwała metoda rozszerzonego kodu dwupoziomowego, dla której uzyskano średni wzrost procentowej efektywności o 0.30868%. Oznacza to średni zysk 2552 bajtów na każdy 1MB skompresowanych danych testowych.

Tabela 6.1 Wyniki testów dla suboptymalnych kodów mieszanych

Typ pliku	Rozmiar [B]	L_2	E_2 [%]	L_{mku}	E_{mku} [%]	L_{rkd}	E_{rkd} [%]
BMP	263222	7.47832	99.70188	7.47832	99.70188	7.45944	99.95415
JPG	896708	7.99791	99.78994	7.99791	99.78994	7.98449	99.95762
MP3	1489609	7.94294	99.78752	7.94294	99.78752	7.92852	99.96896
WAV	572118	7.67210	99.64351	7.67210	99.64351	7.64720	99.96802
PDF	366296	7.92118	99.53184	7.91032	99.66852	7.88599	99.97603
PS	1257608	5.26470	99.32028	5.25391	99.52423	5.23260	99.92957
DJVU	103525	8.00000	99.97188	8.00000	99.97188	7.99908	99.98334
CPP	12328	5.33396	99.31449	5.32028	99.56980	5.30119	99.92832
DOC	505856	4.38683	99.64436	4.38683	99.64436	4.37557	99.90065
XLS	28672	3.45710	98.38101	3.45710	98.38101	3.44751	98.65461
EXE	229376	6.54894	99.59602	6.54894	99.59602	6.52536	99.95584
PPT	1261056	7.91335	99.76010	7.90816	99.82546	7.89674	99.96982
Średnia:		6.65978	99.53690	6.65640	99.59201	6.64031	99.84558

6.3. Suboptymalny kod mieszany ze stałym opóźnieniem

W tej pracy kod mieszany był dotychczas rozważany jako kod z podziałem na wiele buforów r -narych. Najpierw poddawany był kompresji cały ciąg danych, a następnie w pliku wynikowym po nagłówku umieszczano kolejne zakodowane bufory r -narne. Istnieje możliwość przeplatania danych binarnych i r -narych w strumieniu wynikowym, co może być przydatne w przypadku transmisji danych czasu rzeczywistego. Użycie liczb B wprowadza opóźnienie transmisji, gdyż dekodery już w momencie pojawienia się pierwszego słowa kodu zawierającego choć jedną cyfrę r_i -narną (w ogólności f cyfr r_i -narych), powinien mieć dostęp do pierwszej liczby B_i (czyli także do przyszłych $q - f$ cyfr r_i -narych), tak aby móc odczytać (po zdekodowaniu liczby B_i) niezbędnych f cyfr r_i -narych. Ponieważ wraz z każdym słowem kodowym pojawia się średnio $g(r)$ cyfr r -narych, to w sytuacji gdy $q > g(r)$ koder wprowadza opóźnienie, gdyż musi zapisać q cyframi najwolniej zapełniający się bufor r -narny i dopiero po zakodowaniu tych cyfr w liczbę B , wysłać ją do odbiorcy. Opóźnienie oznacza średnią liczbę symboli wejściowych, które należy wczytać, aby zapełnić r_i -narny bufor liczby B_i :

$$T(r) = \frac{q}{g(r)} - 1 \tag{6.1}$$

Aby uprościć zasadę opisaną powyżej transmisji i zmniejszyć wprowadzane średnie opóźnienie można utworzyć suboptymalny kod dwubuforowy, gdzie oprócz strumienia binarnego będzie tylko jeden dodatkowy bufor r -narny. Przykładem takiego kodu jest kod binarno-ternarny opisywany w punkcie 5 tego referatu. W przypadku kodu suboptymalnego możemy zbudować kod o stałym opóźnieniu wynoszącym $q - 1$ symboli wejściowych:

Definicja 6.1 Kod mieszany pierwszego rzędu to taki kod, w którym korzeń drzewa kodowego łączy ze sobą $r > 2$ potomków (korzeń niebinarny), natomiast pozostałe węzły drzewa kodowego posiadają wyłącznie dwóch potomków. Wówczas $g(r) = 1$.

6.4. Optymalny i suboptymalny kod mieszany z małym opóźnieniem

Tworząc suboptymalny kod mieszany pierwszego rzędu o r potomkach w korzeniu można także sprawdzić, czy użycie elementu r -narnego dla pozostałych węzłów pozwoli na dalsze skrócenie średniej długości słowa kodowego. Można w tym celu skorzystać z uproszczenia metody kolejnych uściśleń ograniczając test do dwóch możliwości: węzeł może być binarny lub r -narny (gdzie r zostało wcześniej ustalone dla korzenia). Kod mieszany pierwszego rzędu, który wzbogacimy o kolejne węzły r -narne, charakteryzuje

się zmiennym średnim opóźnieniem kodera, lecz jest ono nie większe niż $q - 1$ symboli wejściowych. Zatem dodatkowe węzły r -narne nie tylko zwiększają efektywność kompresji, ale zmniejszają także średnie opóźnienie kodera. Taki dwubuforowy kod o małym opóźnieniu może być stosowany w transmisji danych czasu rzeczywistego, gdzie dane pojawiają się na wejściu kodera na bieżąco. W podobny sposób można analizować średnie opóźnienie dla kodu mieszanego o wielu buforach r -narych. Wówczas średnie opóźnienie wynosi $T_{sr} = \max \{T(r_1), T(r_2), \dots, T(r_{m-1})\}$. Największa spośród wartości $T(r)$ określa średnie opóźnienie T_{sr} kodu mieszanego. Dla źródła z przykładu 4.2 otrzymano kod binarno-ternarny, dla którego $g(3) = 1.427$, czyli $T_{sr} = 27.7316$, a także kod mieszany, dla którego $g(3) = 1.778$, $g(5) = 0.112$ oraz $g(7) = 0.111$, wówczas $T_{sr} = T(7) = 98.0991$.

7. Dynamiczne mieszane kody Huffmana

Działanie dynamicznego (adaptacyjnego) kodu Huffmana polega na tworzeniu drzewa kodowego na bieżąco, co oznacza, że po każdym odczytaniem z wejścia symbolu następuje jego zakodowanie i aktualizacja drzewa. W odróżnieniu od statycznego kodu Huffmana, nie musimy przesyłać razem z plikiem dodatkowo opisu drzewa kodowego (lub opisu liczności symboli źródła). Ponadto istnieje możliwość kompresji strumienia danych, którego nie znamy w całości w momencie rozpoczęcia kodowania. Ma to miejsce np. przy transmisji danych pojawiających się na wejściu kodera na bieżąco (w czasie rzeczywistym). W przypadku kodera dynamicznego musimy się liczyć z tym, że na początku nie wiemy, jakie symbole będą kodowane, więc musimy zarezerwować dodatkowy symbol nazywany *znacznikiem* [12]. Kod znacznika informuje dekodera, że pojawił się nowy symbol nie występujący wcześniej w drzewie kodowym. Konieczność istnienia dodatkowego symbolu powoduje zmniejszenie efektywności kompresji. Drugą wadą kodera dynamicznego jest większa złożoność obliczeniowa spowodowana częstą potrzebą aktualizacji kształtu drzewa kodowego. Dla binarnego kodu Huffmana istnieją algorytmy szybkiej aktualizacji drzewa [12, 29]. Jednak dla kodów mieszanych stanowi to duży problem ze względu na dużą złożoność obliczeniową tworzenia optymalnego kodu mieszanego. W rozprawie przedstawiono trzy metody dynamicznego kodu mieszanego. Oprócz podstawowej zaprezentowano też wersję, dla której niezbędna jest znajomość całego strumienia danych przed rozpoczęciem kodowania (*metoda druga*), a także *metodę trzecią*, w której kod mieszany zawężono do kodu binarno-ternarnego. Na podstawie wyników testów zamieszczonych w rozprawie można uznać dynamiczny kod binarno-ternarny jako najlepszy kompromis między wydajnością a złożonością obliczeniową i wielkością opóźnienia występującego podczas kompresji (oraz ewentualnej transmisji). Wyniki przykładowego testu zawiera tabela 7.1. Kodowaniu został poddany obraz Lena o 16 poziomach szarości.

Tabela 7.1 Wyniki testów dla obrazu Lena

Typ kompresji	Długość pliku [B]
Mieszany kod dynamiczny	121951
Mieszany kod dynamiczny (metoda druga)	121951
Binarno-ternarny kod dynamiczny (metoda trzecia)	121952
Binarny kod dynamiczny	122831
Statyczny mieszany kod Huffmana	121987
Statyczny binarny kod Huffmana	122868
Statyczny kod arytmetyczny	121367

8. Zastosowania mieszanych kodów Huffmana

8.1. Kompresja elementów DC w standardzie JPEG

Jednym z najbardziej popularnych standardów stratnej kompresji obrazów jest JPEG. Ostatnim etapem kompresji w algorytmie JPEG jest kodowanie współczynników DCT [5]. W przypadku elementów DC koder zapisuje tylko błąd predykcji, czyli różnicę aktualnej i poprzednio kodowanej wartości współczynnika DC. Następnie błędy te koduje się przy pomocy modyfikacji kodu Huffmana. Symbol kodowy składa się z przedrostka (słowa kodu Huffmana) określającego numer grupy oraz dodatkowych bitów, za pomocą których można zapisać reprezentację współczynnika DC w danej grupie. Grupa o numerze n posiada n dodatkowych bitów i umożliwia zakodowanie liczb z przedziału $(-2^n + 1; -2^{n-1})$ oraz $(2^{n-1}; 2^n - 1)$. Przy takiej strukturze danych dopuszcza się tabelę kodową zawierającą maksymalnie 16 grup. Przykładowa uniwersalna tabela kodowa oferowana przez twórców schematu JPEG, powstała na podstawie pomiarów

wielu obrazów testowych. Podczas badań autor rozprawy przygotował do testów własny zestaw 12 obrazów (256 odcieni szarości), na podstawie których zbudowano binarny i mieszany kod Huffmana. Ponieważ uniwersalna tabela (JPEG) nie może być dla każdego obrazu optymalna, więc przy jej użyciu podczas testów otrzymano średnią długość słowa kodowego $L_2 = 2.91054$, co dało efektywność kompresji zaledwie $E_2 = 92.47508\%$. Dla wyznaczonego binarnego kodu Huffmana otrzymano średnią $L_2 = 2.78306$, co dało efektywność kompresji $E_2 = 96.71087\%$. Dla kodu binarno-ternarnego (klasa $r_0 = 2, r_1 = 3, q_1 = 65$) otrzymano $L_{2,3} = 2.75455$, co dało procentową efektywność $E_{2,3} = 97.71196\%$. Dodatkowo po zastosowaniu kodu dwufazowego do kompresji bufora ternarnego uzyskano zmniejszenie średniej długości słowa kodowego do $L_{2,3} = 2.75440$, co dało dalszy wzrost efektywności do $E_{2,3} = 97.77173\%$. Uzyskanie większych efektywności jest trudne ze względu na małą liczbę symboli źródła (zaledwie 8 grup kodowych). Powyższe wyniki zostały opublikowane w pracy [37] wraz z propozycją algorytmu szybkiej konwersji z liczby B do bloku q cyfr ternarnych. Algorytm zastępuje wielokrotne dzielenie liczby B przez 3 zestawem odejmowań kolejnych potęg trójki i porównań znaku wyniku odejmowania. Liczba operacji arytmetycznych jest zależna od wartości konkretnych cyfr ternarnych, ale maksymalnie potrzeba $2 \cdot (q - 1)$ odejmowań, a także $2 \cdot (q - 1) + 1$ przypisań liczby dwubitowej. Istnienie tego algorytmu uzasadnia stwierdzenie, że zaproponowane w rozprawie metody podnoszenia efektywności kompresji w przeciwieństwie np. do techniki rozszerzonych binarnych kodów Huffmana nie wymagają dużych nakładów obliczeniowych i sprzętowych.

8.2. Zastosowanie kodu binarno-ternarnego do bezstratnej kompresji obrazów

Często zachodzi potrzeba bezstratnej archiwizacji obrazów (bez jakiegokolwiek utraty jakości), np. dla obrazów medycznych (badania USG itp.) [32]. W takich przypadkach stosuje się metody modelowania danych takie jak predykcja liniowa, pozwalające uzyskać wzrost efektywności kompresji [21]. Dzięki zastosowaniu predykcji liniowej można usunąć część zależności między sąsiednimi pikselami otrzymując źródło o rozkładzie Laplace’a, co powoduje wzrost wydajności kompresji uzyskanej metodami entropijnymi [33]. Predykcja liniowa zaledwie trzeciego rzędu w połączeniu z suboptymalnym kodem binarno-ternarnym (BT-Huff – klasa kodu $r_0 = 2, r_1 = 3, q_1 = 5$) pozwoliła uzyskać lepsze rezultaty, niż w przypadku użycia takich popularnych formatów graficznych jak GIF oraz PNG. Można się o tym przekonać na podstawie wyników z tabeli 8.1. Jeszcze wyższy stopień kompresji możemy uzyskać dzięki zwiększeniu rzędu predykcji i stopnia skomplikowania algorytmu, co ma miejsce w przypadku metody CALIC.

Tabela 8.1 Średnia liczba bitów na piksel dla czterech standardowych obrazów testowych

Testowany plik	GIF	PNG	Predyktor + BT-Huff	CALIC
<i>Lena</i>	8.077	4.611	4.483	4.121
<i>Airpane</i>	6.547	4.275	4.258	3.743
<i>Peppers</i>	8.236	4.908	4.829	4.425
<i>Woman1</i>	6.938	4.983	4.987	4.555

8.3. Zastosowanie kodów mieszanych do zwiększenia efektywności kodu Golomba

W 1966 roku Solomon W. Golomb w pracy [20] przedstawił podstawowe założenia i przykłady rodziny kodów, które przyjęto określać mianem kodów Golomba. Kod ten służy do reprezentacji nieujemnych liczb całkowitych n , których prawdopodobieństwo jest zgodne z rozkładem geometrycznym $G(n)$:

$$G(n) = (1 - p) \cdot p^n \tag{8.1}$$

Jest to kod przedrostkowy dla źródła o nieskończonym alfabecie symboli (wartości n). W praktyce służy on do kodowania źródła binarnego, dla którego p oznacza prawdopodobieństwo wystąpienia bitu bardziej prawdopodobnego: $p = \max \{p(0), p(1)\} \geq 0.5$. Przyjmijmy, że $p = p(0)$, wówczas słowo kodu Golomba dla wartości n reprezentuje ciąg (symbol) składający się z n zer i jednej jedynki. Główną zaletą tego kodu jest to, że nie wymaga on używania tablic kodowych i jest stosunkowo prosty do implementacji sprzętowej. Słowo kodu Golomba składa się z dwóch części: przedrostka będącego numerem grupy u (w postaci kodu unarnego) oraz z kodu dwufazowego, za pomocą którego koduje się numer elementu v w danej grupie. Grupa o numerze u składa się z m kolejnych elementów alfabetu źródła:

$$m = \left\lceil - \frac{\log_{10}(1 + p)}{\log_{10} p} \right\rceil \tag{8.2}$$

Dla niektórych wartości p możliwe jest utworzenie *zmodyfikowanego kodu Golomba* cechującego się wyższą efektywnością kompresji źródeł binarnych w porównaniu do podstawowego kodu Golomba. Zamiast kodu dwufazowego można wykorzystać kod mieszany dla m symboli równoprawdopodobnych [42]. Np. dla $p = 0.8191$ procentowa efektywność dla kodu podstawowego wynosi $E_G = 98.73897\%$, natomiast dla zmodyfikowanego kodu Golomba uzyskujemy $E_{\underline{G}} = 99.08443\%$ oraz wagę zysku $W = 27.64803\%$.

9. Podsumowanie

Analiza aktualnego stanu wiedzy na temat kodów entropijnych cechujących się wyższą efektywnością od binarnego kodu Huffmana wykazała ich następujące wady:

- wysokie zapotrzebowanie na pamięć niezbędną do przechowywania dużych tablic kodowych (przez koder i dekoder rozszerzonego kodu Huffmana czy kodu Tunstalla);
- większy stopień skomplikowania algorytmu (zwłaszcza dekodera) dla kompresji arytmetycznej, co oznacza dłuższy czas kompresji i dekompresji względem binarnego kodu Huffmana;
- niska efektywność zapisu wielowartościowych słów kodu Huffmana (w odniesieniu do aktualnie użytkowanych architektur sprzętowych opartych o binarną reprezentację liczb) nie pozwalająca na praktyczne ich wykorzystanie.

Z drugiej strony zaproponowany przez autora algorytm kompresji wykorzystujący mieszany kod Huffmana ma następujące własności:

- efektywność mieszanego kodu Huffmana jest w zdecydowanej większości przypadków wyższa od efektywności binarnego kodu Huffmana (w najgorszym przypadku oba te kody są równoważne);
- algorytm charakteryzuje się niską złożonością implementacyjną i małymi wymaganiami pamięciowymi, podobnie jak ma to miejsce w przypadku binarnego kodu Huffmana;
- dzięki odpowiedniej reprezentacji danych możliwe jest użycie wielu różnych elementów wielowartościowych (r -narych) w jednym kodzie, z zachowaniem wysokiej efektywności zapisu cyfr r -narych.

Własności te są zgodne z tezą niniejszej rozprawy. Poniżej przedstawiono klasyfikację najwydajniejszych kodów entropijnych pod względem wymagań sprzętowych niezbędnych do prawidłowego działania algorytmu kompresji (w kolejności od najbardziej do najmniej wymagających). Po lewej stronie znajduje się klasyfikacja dotychczasowa, natomiast po prawej klasyfikacja uwzględniająca zaproponowany w rozprawie algorytm, dzięki któremu kolejność złożoności implementacyjnej pokrywa się z kolejnością efektywności kompresji dla poszczególnych algorytmów:

Dotychczasowe wymagania sprzętowe:

- rozszerzony kod Huffmana
- kod arytmetyczny
- binarny kod Huffmana

Aktualne wymagania sprzętowe (oraz efektywność):

- kod arytmetyczny
- binarno-ternarny (mieszany) kod Huffmana
- binarny kod Huffmana

Oryginalny **wkład własny** autora w pracy stanowi:

- propozycja konstrukcji kodów mieszanych będących uogólnieniem kodu Huffmana;
- rozwiązanie problemu efektywnego zapisu cyfr r -narych (nie będących potęgą dwójki), co jest podstawą praktycznego użycia r -narych kodów Huffmana (łączenie w bloki po q cyfr r -narych, które następnie kompresowane są przy użyciu kodu dwufazowego);
- propozycja efektywnej realizacji mieszanego kodu Huffmana, to jest algorytmu pozwalającego na łączenie wielu różnych symboli r -narych w jednym kodzie w taki sposób, aby uzyskać kod o większej efektywności niż binarny kod Huffmana;
- opisanie podstawowych zasad konstruowania drzewa optymalnego kodu mieszanego (przy danych ograniczeniach);
- wykonanie badań testowych potwierdzających wzrost efektywności uzyskany dzięki zastosowaniu kodów mieszanych (w porównaniu do binarnego kodu Huffmana) zarówno dla zbiorów generowanych w sposób pseudolosowy jak i dla rzeczywistych plików z danymi o różnych rozkładach prawdopodobieństw;
- wyznaczenie wykładniczej złożoności konstrukcji podstawowego algorytmu poszukiwania optymalnego mieszanego kodu Huffmana (metoda porównań) wraz z metodą analitycznego wyznaczania dokładnej liczby iteracji dla metody porównań;

- zdefiniowanie warunków, dla których możliwe jest zmniejszenie liczby iteracji potrzebnych do wyznaczenia optymalnego kodu Huffmana (metoda integralnych poddrzew oraz propozycja poszukiwania poddrzew integralnych binarnie);
- analiza optymalnego binarno-ternarnego kodu Huffmana będącego propozycją zmniejszenia złożoności implementacyjnej kodu mieszanego;
- trzy propozycje użycia suboptymalnych kodów mieszanych o krótkim czasie konstrukcji drzewa kodowego:
 - suboptymalny kod dwupoziomowy;
 - metoda kolejnych uściśleń;
 - rozszerzony kod dwupoziomowy;
- analiza opóźnień wprowadzanych przez mieszany kod Huffmana i propozycja użycia suboptymalnych kodów o stałym opóźnieniu;
- propozycja użycia optymalnych i suboptymalnych kodów o małym opóźnieniu;
- przedstawienie trzech metod kodowania wykorzystujących kody mieszane w celu podniesienia efektywności dynamicznego kodu Huffmana;
- praktyczne zastosowanie kodu binarno-ternarnego jako modyfikacja przedrostkowego kodu Huffmana używanego przy kompresji elementów *DC* w algorytmie JPEG;
- opracowanie algorytmu szybkiego konwertera binarno-ternarnego, który nie wymaga wykonywania operacji dzielenia;
- propozycja użycia kodów mieszanych w celu podniesienia efektywności kodu Golomba;
- analiza efektywności kodu dwufazowego wraz z przedstawieniem prostych algorytmów kodera i dekodera;
- opracowanie algorytmu doboru parametrów wzorcowych q zapewniających wysoką efektywność zapisu cyfr r -narnych;
- przedstawienie dwóch algorytmów tworzenia krótkich informacji nagłówkowych dla statycznych metod entropijnych (opracowanych na podstawie odpowiednio dobranych informacji znanych z literatury);
- wykonanie oprogramowania w języku C opartego na poszczególnych zaproponowanych w pracy algorytmach, a także oprogramowania testującego działanie i umożliwiającego generowanie wykresów zaprezentowanych w pracy (w tym program generujący źródła o założonym przez użytkownika rozkładzie).

W rozprawie na 161 stronach przedstawiono szczegółowy opis powyższych zagadnień poparty ponad dwudziestoma przykładami, wynikami testów oraz analizą matematyczną. Wyniki badań zostały zaprezentowane w 17 referatach na konferencjach o zasięgu zarówno krajowym jak i międzynarodowym. Oprócz samego zdefiniowania zasad działania algorytmu kompresji opartej na mieszanym kodzie Huffmana (zarówno w wersji statycznej jak i dynamicznej), autor przeanalizował także problemy związane ze znalezieniem optymalnego kodu mieszanego wraz z propozycjami ich rozwiązania. Jednak dla źródeł o dużej liczbie symboli (pomijając kody o z góry zdefiniowanych drzewach) przy obecnych możliwościach technicznych wprowadzenie kodu optymalnego do zastosowań czasu rzeczywistego może okazać się niepraktyczne. Dlatego też autor przedstawił kilka propozycji konstruowania drzew suboptymalnego kodu mieszanego o krótkim czasie konstrukcji drzewa kodowego.

Dalsze kierunki badań:

Jak już wspomniano, istotnym problemem jest szybka generacja mieszanych kodów Huffmana dla bardzo dużych źródeł. Konieczne są tu poszukiwania dalszych algorytmów konstrukcji kodów suboptymalnych, a w miarę możliwości także optymalnych. Z praktycznego punktu widzenia duże znaczenie mają powiązania algorytmów z ich realizacją sprzętową, gdyż często od tego zależy, czy dana metoda będzie w przyszłości wykorzystywana w praktyce (np. w układach o małych wymaganiach energetycznych).

Literatura

1. **J. Abrahams, M. J. Lipman** „Zero-redundancy coding for unequal code symbol costs”, IEEE Transactions on Information Theory, vol. 38, no. 5, s. 1583-1586, wrzesień 1992
2. **N. Abramson** „Teoria informacji i kodowania”, Państwowe Wydawnictwo Naukowe, Warszawa 1969
3. **A. V. Aho, J. E. Hopcroft, J. D. Ullman** „Projektowanie i analiza algorytmów”, Wydawnictwo Helion, Gliwice 2003
4. **A. V. Aho, J. D. Ullman** „Wykłady z informatyki z przykładami w języku C”, Wydawnictwo Helion, Gliwice 2003
5. **V. Bhaskaran, K. Konstantinides** „Image and video compression standards – algorithms and architectures” S.Ed., Kluwer Academic Publishers, 1997
6. **A. Bookstein, S. T. Klein, T. Raita** „Is Huffman coding dead?”, Research and Development in Information Retrieval, s. 80-87, 1993
7. **M. Buro** „On the maximum length of Huffman codes”, Information Processing Letters, no. 45, s. 219-223, 1993
8. **R. M. Capocelli, A. De Santis** „A note on D-ary Huffman codes”, IEEE Transactions on Information Theory, vol. 37, no. 1, s. 174-179, styczeń 1991
9. **R. M. Capocelli, A. De Santis** „New bounds on the redundancy of Huffman codes”, IEEE Transactions on Information Theory, vol. 37, no. 4, s. 1095-1104, lipiec 1991
10. **Y. Choueka, S. T. Klein, Y. Perl** „Efficient variant of Huffman codes in high level languages”, Proc. 8-th ACM-SIGIR Conf., s. 122-130, Montreal 1985
11. **T. H. Cormen, C. E. Leiserson, R. L. Rivest** „Wprowadzenie do algorytmów”, Wyd. 5, Wydawnictwa Naukowo-Techniczne, Warszawa 2001
12. **A. Drozdek** „Wprowadzenie do kompresji danych”, Wydawnictwa Naukowo-Techniczne, Warszawa 1999
13. **F. Fabris** „Variable-Length -to- Variable-Length source coding: A greedy step-by-step algorithm”, IEEE Transactions on Information Theory, vol. 38, s. 1609-1617, wrzesień 1992
14. **L. Gajek, M. Kałuszka** „Wnioskowanie statystyczne – modele i metody”, Wyd. 2 Wydawnictwa Naukowo-Techniczne, Warszawa 1996
15. **R. G. Gallager** „Optimal source codes for geometrically distributed integer alphabets”, IEEE Transactions on Information Theory, s. 228-230, marzec 1975
16. **R. G. Gallager** „Variations on a theme by Huffman”, IEEE Transactions on Information Theory, vol. 24, no. 6, s. 668-674, listopad 1978
17. **N. C. Geçkinli** „Two corollaries to the Huffman coding procedure”, IEEE Transactions on Information Theory, s. 342-344, maj 1975
18. **E. N. Gilbert** „Coding with digits of unequal cost”, IEEE Transactions on Information Theory, vol. 41, no. 2, s. 596-600, marzec 1995
19. **M. J. Golin, G. Rote** „A dynamic programming algorithm for constructing optimal prefix-free codes with unequal letter costs”, IEEE Transactions on Information Theory, vol. 44, no. 5, s. 1770-1781, wrzesień 1998
20. **S. W. Golomb** „Run-length encoding”, IEEE Transactions on Information Theory, vol. 12, s. 399-401, lipiec 1966
21. **K. Heim** „Metody kompresji danych”, Wydawnictwo MIKOM, Warszawa 2000
22. **D. A. Huffman** „A method for the construction of minimum-redundancy codes”, Proc. IRE, vol. 40, s. 1098-1101, 1952
23. **A. Kato, Te Sun Han, H. Nagaoka** „A D-ary Huffman code for a class of sources with countably infinite alphabets”, Proc. 1995 IEEE Int. Symp. on Information Theory, p. 324, 1995
24. **G. O. H. Katona, T. O. H. Nemetz** „Huffman codes and self-information”, IEEE Transactions on Information Theory, vol. 22, no. 3, s. 337-340, maj 1976
25. **T. Linder, V. Tarokh, K. Zeger** „Existence of optimal prefix codes for infinite source alphabets”, IEEE Transactions on Information Theory, vol. 43, no. 6, s. 2026-2028, listopad 1997
26. **D. J. C. MacKay** „Information Theory, Inference, and Learning Algorithms”, Version 6.0, Cambridge University Press, czerwiec 2003
27. **D. Manstetten** „Tight bounds on the redundancy of Huffman codes”, IEEE Transactions on Information Theory, vol. 38, no. 1, s. 144-151, styczeń 1992
28. **J. L. Massey** „Applied digital information theory I”, script only for personal use, kopia elektroniczna
29. **K. Sayood** „Kompresja danych – wprowadzenie”, Wyd. 2, Wydawnictwo RM, Warszawa 2002

30. **K. Sayood (editor)** „*Lossless Compression Handbook*”, Academic Press USA, 2003
31. **C. E. Shannon** „*A mathematical theory of communication*”, The Bell System Technical Journal, vol. 27, s. 379-423, 623-656, lipiec, październik 1948
32. **W. Skarbek i inni** „*Multimedia i standardy kompresji danych*”, Akademicka Oficyna Wydawnicza PLJ, Warszawa 1998
33. **W. Skarbek** „*Metody reprezentacji obrazów cyfrowych*”, Akademicka Oficyna Wydawnicza PLJ, Warszawa 1993
34. **R. Stasiński, G. Ulacha** „*Huffman codes revisited*”, Proceedings of 24th Symposium of Information Theory in The Benelux, Veldhoven 2003
35. **W. Szpankowski** „*Asymptotic average redundancy of Huffman (and other) block codes*”, IEEE Transactions on Information Theory, vol. 46, no. 7, s. 2434-2443, listopad 2000
36. **B. P. Tunstall** „*Synthesis of Noiseless Compression Codes*”, Ph. D. thesis, Georgia Institute of Technology, Atlanta, GA, wrzesień 1967
37. **G. Ulacha** „*Wzrost efektywności kompresji JPEG – algorytm szybkiego dekodera liczb ternarnych*”, Materiały VI konferencji Reprogramowalne Układy Cyfrowe, Szczecin 2003
38. **G. Ulacha** „*Podstawowe warunki stosowania mieszanych kodów Huffmana*”, Materiały V Ogólnopolskich Warsztatów Doktoranckich, Gliwice 2003
39. **G. Ulacha, R. Stasiński** „*Source binary coding using non-binary Huffman codes*”, Materiały konferencyjne Krajowego Sympozjum Nauk Radiowych URSI, Poznań 2002
40. **G. Ulacha, R. Stasiński** „*Mixed Huffman codes*”, Proceedings of 9th Advanced Computer Systems, Międzyzdroje 2002
41. **G. Ulacha, R. Stasiński** „*Efektywne kodowanie binarne elementów r-narnych dla potrzeb kodowania źródła*”, Materiały konferencyjne VII Poznańskich Warsztatów Telekomunikacyjnych, Poznań 2002
42. **G. Ulacha, R. Stasiński** „*On the optimization of Golomb codes using block coding of m-nary elements*”, Proceedings of 10th Advanced Computer Systems, Międzyzdroje 2003
43. **J. S. Vitter** „*Dynamic Huffman coding*”, ACM Transactions on Mathematical Software, vol. 15, s. 158-167, 1989
44. **F. Willems, Y. Shtakov, T. Tjalkens** „*Reflections on the prize paper: 'The context-tree weighting method: basic properties'*”, IEEE Information Theory Society Newsletter, vol. 47, no. 1, s. 1, 20-27, marzec 1997
45. **I. H. Witten, R. M. Neal, J. G. Cleary** „*Arithmetic coding for data compression*”, Communications of the ACM, vol. 30, no. 6, s. 520-540, czerwiec 1987
46. **J. Ziv, A. Lempel** „*A universal algorithm for sequential data compression*”, IEEE Transactions on Information Theory, vol. 23, no. 3, s. 337-343, maj 1977
47. **J. Ziv, A. Lempel** „*Compression of individual sequences via variable-rate coding*”, IEEE Transactions on Information Theory, vol. 24, no. 5, s. 530-536, wrzesień 1978

Biblioteka Wydziału Informatyki
Politechniki Szczecińskiej

ARF-33