



POLITECHNIKA WROCŁAWSKA
WYDZIAŁ INFORMATYKI I ZARZĄDZANIA
Wybrzeże Wyspiańskiego 27, 50-370 Wrocław

Rozprawa Doktorska

PREDYKCYJNO-SUBSTYTUCYJNA
METODA BEZSTRATNEJ KOMPRESJI DANYCH

Jakub Swacha

Promotor: prof. US dr hab. Zdzisław Szyjewski

Wrocław 2003

Streszczenie

Niniejsza praca podejmuje temat bezstratnych metod kompresji danych. Celem pracy jest zdefiniowanie oryginalnej metody kompresji danych łączącej w sobie podejście substytucyjne i predykcyjne. Przedstawiono algorytm realizujący nową metodę, którego elementy składowe zawierają szereg oryginalnych rozwiązań, poprawiających ich szybkość i efektywność działania. Nową metodę poddano weryfikacji empirycznej, wykazując że jej wysoka szybkość działania i dobra efektywność kompresji czyni z niej najlepsze dostępne rozwiązanie dla średnich prędkości transmisji.

Abstract

The thesis is devoted to lossless data compression methods. Its main aim is to define an original data compression method which is both predictive and substitutional. An algorithm for the new method has been presented, comprising many original developments improving its speed and effectiveness. Empirical verification of the new method shows that its high speed and compression effectiveness makes it the best available solution for the average range of transmission speeds.

Spis treści

WYKAZ SYMBOLI I OZNACZEŃ STOSOWANYCH W PRACY	6
WYKAZ TABEL	7
WYKAZ RYSUNKÓW	8
WSTĘP	10
<u>1 KOMPRESJA DANYCH</u>	<u>12</u>
1.1 ISTOTA KOMPRESJI DANYCH	12
1.1.1 PODSTAWOWE POJĘCIA	12
1.1.2 FAZY KOMPRESJI DANYCH	13
1.1.3 ŹRÓDŁA DANYCH I ICH MODELE	14
1.1.4 KODOWANIE	17
1.1.5 PRZETWARZANIE WSTĘPNE	18
1.1.6 TRANSFORMACJE	19
1.1.7 ROLA CZASU W MODELU	20
1.1.8 MIARY KOMPRESJI	21
1.2 PODZIAŁ METOD KOMPRESJI DANYCH	24
1.2.1 KRYTERIUM WIERNEGO ODTWORZENIA	24
1.2.2 KLASYFIKACJA METOD KOMPRESJI BEZSTRATNEJ	26
1.2.3 STRATNA KOMPRESJA DANYCH	29
1.2.4 METODY STRATNEJ KOMPRESJI DANYCH	30
1.3 ZASTOSOWANIA KOMPRESJI DANYCH	31
1.3.1 OBSZARY ZASTOSOWAŃ METOD KOMPRESJI DANYCH	31
1.3.2 PODSTAWOWY OBSZAR ZASTOSOWAŃ METOD KOMPRESJI DANYCH	32
1.3.3 INNE ZASTOSOWANIA METOD KOMPRESJI DANYCH	37
<u>2 METODY BEZSTRATNEJ KOMPRESJI DANYCH</u>	<u>39</u>
2.1 KODOWANIE STATYSTYCZNE	39
2.1.1 KODOWANIE HUFFMANA	39
2.1.2 KODOWANIE ARYTMETYCZNE	41
2.1.3 KODOWANIE PRZEDZIAŁÓW	42
2.2 METODY SUBSTYTUCYJNE	44

2.2.1	PODSTAWY KOMPRESJI SUBSTYTUCYJNEJ	44
2.2.2	KODOWANIE DŁUGOŚCI SERII	46
2.2.3	METODA LZ77 I POCHODNE	47
2.2.4	METODA LZFG	49
2.2.5	METODA LZ78 I LZW	49
2.3	METODY PREDYKCYJNE	51
2.3.1	PREDYKCJA PRZEZ CZĘŚCIOWE DOPASOWANIE (PPM)	51
2.3.2	PPM Z DZIEDZICZENIEM INFORMACJI (PPMII)	53
2.3.3	PPM O NIEOGRANICZONEJ DŁUGOŚCI KONTEKSTU (PPM*)	54
2.3.4	METODA WAŻENIA DRZEW KONTEKSTOWYCH (CTW)	55
2.3.5	DYNAMICZNA KOMPRESJA MARKOWA (DMC)	56
2.3.6	ALGORYTM PRZELĄCZAJĄCY VOLFA I WILLEMSA (VW98)	58
2.4	METODY TRANSFORMACYJNE	59
2.5	METODY HYBRYDOWE	60
2.5.1	METODY SUBSTYTUCYJNE ZE SŁOWNIKIEM KONTEKSTOWYM	60
2.5.2	METODA LZP	62
2.5.3	INNE ALGORYTMY HYBRYDOWE	63
2.6	ZESTAWIENIE METOD KOMPRESJI BEZSTRATNEJ	64
3	<u>METODA PREDYKCYJNO-SUBSTYTUCYJNA</u>	66
3.1	WPROWADZENIE DO PROBLEMATYKI PREDYKCYJNO-SUBSTYTUCYJNEJ	66
3.1.1	PODSTAWOWA TERMINOLOGIA	66
3.1.2	PREDYKCYJNO-SUBSTYTUCYJNY MODEL ŹRÓDŁA DANYCH	69
3.1.3	MINIMALNA I ZADOWALAJĄCA DŁUGOŚĆ SUBSTYTUCJI	71
3.1.4	PRZELĄCZANIE. METODY SKŁADOWE	74
3.1.5	KOD NIEPEŁNY. ZASADY WYŁĄCZANIA	76
3.2	PRZESŁANKI DO POSZUKIWANIA NOWEJ METODY	79
3.2.1	KRYTYCZNA DYSKUSJA ISTNIEJĄCYCH METOD	79
3.2.2	ZASADNOŚĆ POŁĄCZENIA PREDYKCJI I SUBSTYTUCJI	81
3.3	ZARYS METODY PREDYKCYJNO-SUBSTYTUCYJNEJ	82
3.3.1	PODSTAWY METODY PREDYKCYJNO-SUBSTYTUCYJNEJ	82
3.3.2	SCHEMATY DZIAŁANIA METODY PREDYKCYJNO-SUBSTYTUCYJNEJ	84
3.3.3	PROBLEMY BADAWCZE W OBSZARZE METODY PREDYKCYJNO-SUBSTYTUCYJNEJ	86
3.4	ALGORYTM PREDYKCYJNO-SUBSTYTUCYJNY BEZ ESTYMACJI DŁUGOŚCI ZAPISU	87
3.4.1	STRUKTURY DANYCH	87

3.4.2	ALGORYTM PRZEŁĄCZAJĄCY	91
3.4.3	KOMPONENT SUBSTYTUCYJNY	93
3.4.4	KOMPONENT PREDYKCYJNY	97
3.4.5	PARAMETRY ALGORYTMU	101
3.4.6	PODSTAWOWE WŁAŚCIWOŚCI ALGORYTMU	103
3.5	ALGORYTM PREDYKCYJNO-SUBSTYTUCYJNY Z ESTYMACJĄ DŁUGOŚCI ZAPISU	104
3.5.1	ALGORYTM PRZEŁĄCZAJĄCY	104
3.5.2	PODSTAWOWE WŁAŚCIWOŚCI ALGORYTMU	107
3.5.3	PRZYKŁAD KOMPRESJI ALGORYTMEM PREDYKCYJNO-SUBSTYTUCYJNYM	108
3.5.4	PORÓWNANIE Z ISTNIEJĄCYMI ALGORYTMAMI HYBRYDOWYMI	110
4	<u>WERYFIKACJA METODY PREDYKCYJNO-SUBSTYTUCYJNEJ</u>	<u>112</u>
4.1	METODOLOGIA BADAWCZA	112
4.1.1	TESTOWY ZESTAW PLIKÓW	112
4.1.2	SPOSÓB PROWADZENIA BADAŃ	117
4.2	EKSPERYMENTY Z RÓŻNYMI PARAMETRAMI UŻYTKOWNIKA	120
4.2.1	MODYFIKACJA RZĘDU MODELU STATYSTYCZNEGO	120
4.2.2	MODYFIKACJA WIELKOŚCI SŁOWNIKÓW I MINIMALNEJ DŁUGOŚCI SUBSTYTUCJI	122
4.2.3	DŁUGOŚĆ KOMPRESOWANEGO PLIKU A REZULTATY KOMPRESJI	125
4.3	PORÓWNANIE Z UŻYTYM ALGORYTMEM PREDYKCYJNYM	126
4.4	PORÓWNANIE Z INNYMI DOSTĘPNYMI ALGORYTMAMI	127
4.4.1	TESTOWANE ALGORYTMY	127
4.4.2	PORÓWNANIE EFEKTYWNOŚCI KOMPRESJI	130
4.4.3	PORÓWNANIE SZYBKOŚCI DZIAŁANIA	133
4.4.4	PORÓWNANIE DWUKRYTERIALNE	135
4.4.5	PORÓWNANIE PRZYSPIESZENIA TRANSMISJI DLA ZAŁOŻONEJ PRĘDKOŚCI	136
4.4.6	PORÓWNANIE DLA RÓŻNYCH PRĘDKOŚCI TRANSMISJI	139
4.4.7	PORÓWNANIE DLA RÓŻNYCH ŚRODOWISK TESTOWYCH	140
	<u>PODSUMOWANIE</u>	<u>143</u>
	<u>LITERATURA</u>	<u>147</u>

Wykaz symboli i oznaczeń stosowanych w pracy

A	alfabet źródła danych
$ A $	wielkość alfabetu
A_{ef}	alfabet efektywny zbioru danych
a	symbol alfabetu
α	współczynnik substytucyjności
BR	średnia bitowa
CST	całkowita szybkość transmisji
CCT	całkowity czas transmisji
CP	współczynnik procentowy kompresji
CR	stopień kompresji
$c(M)$	liczba niezerowych liczników w modelu M
D	słownik fraz
d	odległość pomiędzy frazami
γ	współczynnik nieefektywności kodera statystycznego
$H(X)$	entropia źródła X
$I(Z)$	informacja przyporządkowana zdarzeniu Z
K	rząd (całościowego) modelu PPM
k	długość kontekstu, rząd (częstkowego) modelu statystycznego
$L_V(U)$	długość U po zakodowaniu metodą V
l	długość frazy
l_{min}	minimalna długość substytucji
l_{suf}	wystarczająca długość substytucji
$\lg$	logarytm o podstawie 2
M^k	model statystyczny rzędu k
$P(a M)$	prawdopodobieństwo wystąpienia symbolu a według modelu M
$q(a M)$	licznik wystąpień symbolu a w modelu M
q_s	okres skalowania
$R(X)$	redundancja źródła X
$S(U)$	długość U przed kompresją
Sw	przełącznik
$T(M_i^k)$	suma liczników modelu M_i^k (bez wyłączenia symboli)
$T(M_i^k M_i^{k+1})$	suma liczników modelu M_i^k z wyłączeniem symboli
t_d	czas dekompresji
t_k	czas kompresji
t_t	czas transmisji
v_t	prędkość transmisji
WPT	współczynnik przyspieszenia transmisji
X	źródło danych
x_i	symbol źródła znajdujący się na pozycji i
x_i^{i+l-1}	sekwencja symboli (faza) o długości l i początku na pozycji i
x_{i-k}^{i-1}	kontekst długości k dla pozycji źródła i
Z	zdarzenie elementarne

Wykaz tabel

Tabela 1.1. Przekształcenia fazy przetwarzania wstępnego i fazy transformacji....	20
Tabela 1.2. Model stały a adaptacyjny – porównanie.....	26
Tabela 1.3. Wybrane zastosowania algorytmu Huffmana	38
Tabela 2.1. Przykład kompresji metodami substytucyjnymi	47
Tabela 2.2. Porównanie metod słownikowych	50
Tabela 2.3. Zestawienie klasycznych metod PPM	52
Tabela 2.4. Porównanie metod transformacyjnych.....	60
Tabela 2.5. Zestawienie efektywności wybranych metod kompresji bezstratnej	65
Tabela 2.6. Szybkość działania i użycie pamięci wybranych metod kompresji.....	65
Tabela 3.1. Porównanie rozmiaru po kompresji (L) i całkowitego czasu transmisji (CCT, przy prędkości 512 kb/s) dla Zipa (tryb domyślny) i PPMd (rzęd 8)....	80
Tabela 3.2. Porównanie rozmiaru po kompresji (L) i całkowitego czasu transmisji (CCT, przy prędkości 512 kb/s) dla PPMd, rząd modelu 2, 4 i 8	80
Tabela 3.3. Porównanie wybranych cech metody predykcyjnej i substytucyjnej ...	82
Tabela 3.4. Struktury danych dla kontekstu o $c = 256$ elementach.....	91
Tabela 3.5. Hybrydowe algorytmy kompresji – porównanie, część pierwsza	110
Tabela 3.6. Hybrydowe algorytmy kompresji – porównanie, część druga	110
Tabela 3.7. Algorytm predykcyjno-substytucyjny a VW98.....	111
Tabela 4.1. Pliki wykorzystane w testach.....	116
Tabela 4.2. Wartości przeciętne dla kolejnych 20 pomiarów szybkości kompresji (kB/s) pliku <i>TXT</i> przez programy PLZ i PPMd	119
Tabela 4.3. Średnia bitowa (w bpc) dla różnych rzędów modelu uzyskana programem PLZ na plikach tworzących <i>Calgary Corpus</i>	120
Tabela 4.4. Porównanie całosciowe dla różnych rzędów modelu.....	122
Tabela 4.5. Łączny odsetek podmienionych symboli (plik <i>Calgary Corpus</i>), w zależności od rozmiarów słowników poszczególnych rzędów	123
Tabela 4.6. Rozmiar słownika rzędu 2 a szybkość i efektywność kompresji	124
Tabela 4.7. Minimalna długość substytucji a szybkość i efektywność kompresji..	124
Tabela 4.8. Rezultaty testów dla kolejnych części pliku <i>Long</i>	125
Tabela 4.9. Porównanie z użytym algorytmem predykcyjnym	126
Tabela 4.10. Algorytmy i programy uwzględnione w testach.....	129
Tabela 4.11. Średnia bitowa (w bpc) osiągnięta przez testowane algorytmy dla poszczególnych plików	131

Tabela 4.12. Szybkość działania zmierzona dla testowanych algorytmów na poszczególnych plikach (MB/s)	133
Tabela 4.13. Współczynnik przyspieszenia transmisji testowanych algorytmów przy prędkości transmisji 512 kb/s	137
Tabela 4.14. Procentowa poprawa WPT poprzez użycie PLZ 2 w stosunku do innych testowanych algorytmów przy założonej prędkości transmisji 512 kB/s	139

Wykaz rysunków

Rys. 1.1. Cztery podstawowe fazy kompresji.....	14
Rys. 2.1. Przykład budowy drzewa kodowego w algorytmie Huffmana.....	40
Rys. 2.2. Kodowanie arytmetyczne – przykład Źródło: [DEOR03].	42
Rys. 2.3. Modele Markowa pierwszego rzędu Źródło: [BUNT96], opracowanie własne. ..	57
Rys. 2.4. Proces klonowania stanów modelu Markowa Źródło: [COHO87], opracowanie własne.	57
Rys. 2.5. Transformacja sortująca Burrowsa-Wheelera.	59
Rys. 3.1. Ogólny schemat dwuprzebiegowej realizacji metody predykcyno-substytucyjnej dla całego zbioru danych	84
Rys. 3.2. Schemat jednoprzebiegowej realizacji metody predykcyno-substytucyjnej bez estymacji długości zapisu	85
Rys. 3.3. Schemat jednoprzebiegowej realizacji metody predykcyno-substytucyjnej z estymacją długości zapisu	86
Rys. 4.1. Średnia bitowa dla pliku EXE i plików z innych korpusów	116
Rys. 4.2. Średnia bitowa osiągnięta przez testowane algorytmy dla korpusu <i>Calgary</i>	132
Rys. 4.3. Średnia bitowa osiągnięta przez testowane algorytmy dla korpusu <i>Canterbury</i>	132
Rys. 4.4. Szybkość działania zmierzona dla testowanych programów na korpusie <i>Calgary</i>	134
Rys. 4.5. Szybkość działania zmierzona dla testowanych programów na korpusie <i>Canterbury</i>	134
Rys. 4.6. Czas przetwarzania a efektywność kompresji <i>Calgary Corpus</i>	135
Rys. 4.7. Czas przetwarzania a efektywność kompresji <i>Canterbury Corpus</i>	135

Rys. 4.8. Współczynnik przyspieszenia transmisji testowanych algorytmów przy prędkości transmisji 512 kb/s dla korpusu <i>Calgary</i>.....	137
Rys. 4.9. Współczynnik przyspieszenia transmisji testowanych algorytmów przy prędkości transmisji 512 kb/s dla korpusu <i>Canterbury</i>	138
Rys. 4.10. Przyspieszenie transmisji przy różnych założonych prędkości transmisji (<i>Calgary Corpus</i>)	140
Rys. 4.11. Przyspieszenie transmisji przy różnych założonych prędkości transmisji (<i>Canterbury Corpus</i>).....	140
Rys. 4.12. Przyspieszenie transmisji w środowisku Celeron 800 (<i>Canterbury Corpus</i>)	141
Rys. 4.13. Przyspieszenie transmisji w środowisku Athlon 1600 (<i>Canterbury Corpus</i>)	141
Rys. 4.14. Przyspieszenie transmisji w środowisku Athlon 2400 (<i>Canterbury Corpus</i>)	142

Źródło rysunków, dla których nie podano innego źródła: opracowanie własne.

Wstęp

Postęp informatyczny, który nabrał impetu w ostatnich latach XX wieku, w wielkiej mierze przeobraził naszą cywilizację. Technologie cyfrowe przyniosły nowe sposoby komunikacji, umożliwiły zbieranie i przetwarzanie danych na nieznaną dotąd skalę, co zaowocowało zmianami w postrzeganiu świata i podejmowaniu decyzji.

Gromadzenie i transmisja danych przekładają się na pewne wymierne wymagania co do infrastruktury mającej za zadanie zapewnienie płynności i stabilności tych procesów. Wzrost zapotrzebowania na informację pociąga za sobą wzrost zapotrzebowania na media służące jej przechowywaniu i przesyłaniu. Ekstensywna rozbudowa tej infrastruktury jest kosztowna i prowadzi ponadto do jej komplikacji, co z kolei oznacza zwykle spadek efektywności.

Prostą alternatywą dla zwiększania istniejącej bazy technicznej jest lepsze jej wykorzystanie. Ulepszenia mogą dotyczyć organizacji przechowywania i przesyłania danych – tworzenia podręcznych magazynów danych lub stosowania dedykowanych protokołów transmisji. Jeszcze lepsze efekty powinno dać zmniejszenie objętości przechowywanych lub przesyłanych danych.

W praktyce dane pochodzące z większości typowych źródeł zawierają w dużej mierze informację nieistotną lub powieloną. Odjęcie tych niepotrzebnych składników może radykalnie zmniejszyć objętość zbioru danych. Taka operacja ma znaczenie nie tylko w kontekście ilościowym – zmniejszenie kosztów, przyspieszenie tempa pracy – ale również jakościowym – umożliwia łamanie barier technologicznych, wykonywanie czynności dotąd niemożliwych: cyfrową rejestrację i transmisję danych przy ograniczonej fizycznie pojemności nośnika lub przepustowości kanału komunikacyjnego. Operację zmniejszenia nadmiarowości (redundancji) zbioru danych nazywamy kompresją danych.

Choć termin *kompresja* przywołać może na myśl silniki spalinowe czy sprężarki, mimowolnie korzystamy z dobrodziejstw kompresji danych prawie na każdym kroku: rozmawiamy przez telefon komórkowy dzięki zaimplementowanej w nim kompresji mowy, robimy zdjęcia cyfrowym aparatem fotograficznym dzięki kompresji obrazów, oglądamy setki kanałów telewizji satelitarnej dzięki kompresji dźwięku i obrazów

ruchomych, a kiedy łączymy się poprzez linię telefoniczną z Internetem, dzięki kompresji, znacznie szybciej udaje się nam przeglądać strony WWW.

Powszechność zastosowań kompresji danych potwierdza wagę roli, którą spełnia ona w świecie, w którym istotną rolę odgrywają technologie cyfrowe. Nie oznacza jednak, że w dziedzinie tej powiedziano już ostatnie słowo. Próba wniesienia czegoś nowego do tego tematu jest także niniejsza praca.

Celem niniejszej pracy jest zdefiniowanie oryginalnej metody kompresji łączącej w sobie wybrane elementy istniejących metod różnych typów, opracowanie realizującego ją algorytmu i empiryczne jego zweryfikowanie. Weryfikacja pokaże, czy rozwiązanie wykorzystujące dwie metody, z których jedna jest szybka, a druga efektywna, będzie lepsze od ich obu, gdy jednocześnie uwzględni się zarówno czas działania, jak i efektywność kompresji.

Pierwsza część celu – zdefiniowanie predykcjno-substytucyjnej metody kompresji – zostanie osiągnięta drogą analityczną, poprzez:

- porównanie istniejących metod kompresji, w poszukiwaniu rozwiązań, które nadawałyby się do wkomponowania w nową metodę, czemu poświęcony będzie rozdział drugi pracy;
- określenie najważniejszych zagadnień dotyczących łączenia dwóch różnych metod kompresji, co zostanie wykonane w podrozdziale 3.1;
- uzasadnienie sensu tworzenia nowej metody, poprzez pokazanie niedoskonałości istniejących metod, oraz dokonanie wyboru komponentów składowych, wskazując na wzajemnie dopełniające się cechy podejścia predykcyjnego i substytucyjnego (zawarte w podrozdziale 3.2).

Konkluzją powyższych działań będzie ogólny zarys nowej metody, który można przyrównać do zbioru reguł, którym powinna być ona podporządkowana. Aby nadawał się on do bezpośredniego zastosowania, określona ramą definicji przestrzeń wypełniona zostanie treścią poprzez opracowanie algorytmu realizującego metodę predykcjno-substytucyjną. Wszystko to ujęte zostanie w dalszej części rozdziału trzeciego.

Empiryczna weryfikacja nowego algorytmu przeprowadzona zostanie w rozdziale czwartym. Obejmie zbadanie działania nowego algorytmu w zależności od wartości jego parametrów wejściowych oraz porównanie go z wyjściowym algorytmem predykcyjnym, aby sprawdzić czy przyjęty kierunek rozwoju metody jest właściwy.

Najważniejszym punktem weryfikacji będzie jednak zestawienie wyników osiąganych przez nowy algorytm z wynikami osiąganymi przez wybór najlepszych istniejących algorytmów.

Pracę kończy przedstawienie wniosków nasuwających się z uzyskanych wyników.

1 Kompresja danych

1.1 Istota kompresji danych

1.1.1 Podstawowe pojęcia

Niech X oznacza źródło danych, a x_i i -tą daną elementarną wygenerowaną przez to źródło. Zbiór wszystkich możliwych danych elementarnych, które mogą być wygenerowane przez źródło X , oznaczamy przez A i nazywamy **alfabetem źródła**. Elementy alfabetu nazywamy **symbolami** źródła; i -ty element alfabetu oznaczamy a_i . Liczba elementów alfabetu jest skończona¹ i wynosi $|A|$.

W niniejszej pracy zajmować się będziemy przede wszystkim źródłami danych zorientowanymi bajtowo. Alfabetem takiego źródła jest $A = \{0, 1, \dots, 255\}$, a $|A| = 256$.

Wygenerowany przez źródło X skończony ciąg danych elementarnych nazywamy **źródłowym zbiorem danych**. Zbiór wszystkich symboli, które występują w zbiorze źródłowym, nazywamy **alfabetem efektywnym** i oznaczamy A_{ef} . $A_{ef} \subset A$, $|A_{ef}| \leq |A|$.

Kompresją danych nazywa się proces przekształcenia pierwotnej reprezentacji zbioru danych w inną reprezentację o krótszej długości zapisu.

W trakcie tego procesu symbole pobierane ze zbioru wejściowego zamieniane są na **słowa kodowe** zapisywane do zbioru wyjściowego. Każdy symbol należący do alfabetu efektywnego posiada przynajmniej jedno odpowiadające mu słowo kodowe.

Proces odwrotny do kompresji nazywamy **dekompresją danych**.

Celem kompresji jest usunięcie ze zbioru danych części nadmiarowej. Duża część danych, wymienianych między ludźmi w życiu codziennym, jest nadmiarowa. W warunkach naturalnych, nadmiarowość może być cechą pożądaną, na przykład ze względu na zasady działania kanału transmisyjnego lub występowanie w nim zakłóceń [DĄDY99, s. 11]. W pozbawionym zakłóceń cyfrowym kanale komunikacji nadmiarowość jest najczęściej zbędna.

Matematyczną definicję informacji podał jako pierwszy C.E. Shannon w swojej pracy *A Mathematical Theory of Communication* [SHAN48]. Załóżmy, że dane jest zdarzenie Z , które jest zbiorem wyników pewnego losowego eksperymentu. Jeżeli $P(Z)$

¹ Źródła opisywane w pracy są źródłami dyskretnymi.

oznacza prawdopodobieństwo wystąpienia zdarzenia Z , to informacja przyporządkowana zdarzeniu Z dana jest wzorem:

$$I(Z) = \log_b \frac{1}{P(Z)} = -\log_b P(Z), \quad (1.1)$$

gdzie b zależy od wybranej jednostki informacji. Jeżeli b równe jest 2, to jednostkę taką nazywamy **bitem**. Nie należy przy tym mylić dwóch znaczeń słowa *bit*: przedstawionej właśnie jednostki informacji i sygnału elementarnego mogącego przyjmować wartości 0 i 1. Z uwagi na to, że w dalszej części pracy będziemy się wielokrotnie posługiwać bitami, dla prostszego zapisu logarytmów o podstawie 2, użyjemy postaci skróconej $\lg$:

$$\lg x = \log_2 x. \quad (1.2)$$

Miarą średniej zawartości informacji w źródle danych X jest jego **entropia** H , definiowana wzorem [SHAN48, s.11]:

$$H(X) = -\sum_{i=1}^n P(x_i) \lg P(x_i), \quad (1.3)$$

gdzie: n – liczba symboli w zbiorze, $P(x_i)$ – prawdopodobieństwo wystąpienia symbolu x_i .

Znając entropię $H(X)$, ustalić można **redundancję** $R(X)$, czyli nadmiarowość zbioru X :

$$R(X) = \sum_{i=1}^n P(x_i) L(x_i) - H(X), \quad (1.4)$$

gdzie: n – liczba symboli w zbiorze, $P(x_i)$ – prawdopodobieństwo wystąpienia symbolu x_i , $L(x_i)$ – długość słowa kodowego reprezentującego symbol x_i .

Z równań powyższych wynika, że dla symbolu x optymalna długość słowa kodowego $L(x)$ wynosi $-\lg P(x)$ bitów.

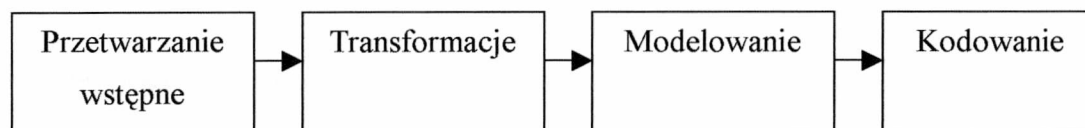
1.1.2 Fazy kompresji danych

Złożoność typowych źródeł danych wymusza pewien stopień skomplikowania metod kompresji danych. Ważne jest, by prawidłowo rozpoznać i wykorzystać zależności występujące w źródle danych. Z tego powodu, w większości metod kompresji rozróżniamy co najmniej dwie fazy:²

- **modelowanie**, mające za zadanie zbudowanie użytecznego modelu źródła,
- **kodowanie**, mające za zadanie przetworzenie danych wejściowych w oparciu o posiadany ich model.

² Wyodrębnienie fazy modelowania z fazy kodowania określa się mianem nowoczesnego paradygmatu kompresji danych [BUNT96].

Niekiedy faz tych wyróżnić można więcej. Cztery podstawowe (mogą dzielić się na etapy) fazy kompresji przedstawia rysunek 1.1.³



Rys. 1.1. Cztery podstawowe fazy kompresji

1.1.3 Źródła danych i ich modele

Modele budować można różnymi sposobami. W przypadku danych pochodzenia naturalnego można wykorzystać wiedzę o sposobie, w jaki dane te są generowane. Takie modele nazywamy fizycznymi. Z powodzeniem wykorzystuje się je na przykład w kompresji ludzkiej mowy [KRUP98, s. 57].

O wiele szersze spektrum zastosowań mają modele probabilistyczne. Swoją metodologię czerpią one z rachunku prawdopodobieństwa, a w szczególności z wywodzącej się z niego teorii procesów stochastycznych.

Podstawowym zagadnieniem przy budowie modeli probabilistycznych jest znalezienie rozkładu prawdopodobieństw symboli alfabetu źródła danych w oparciu o znany fragment tego źródła.

Modele probabilistyczne tworzy się zwykle zakładając ergodyczność i stacjonarność źródła danych. Źródło danych jest ergodyczne, jeżeli wszystkie realizacje procesu, który reprezentuje, są typowe w tym sensie, że znajomość pojedynczej realizacji tego procesu na nieskończonym odcinku czasowym pozwala wyznaczać rozkład prawdopodobieństwa w innym hipotetycznie identycznym procesie. Źródło ergodyczne jest stacjonarne, jeżeli zaobserwowane w nim związki probabilistyczne nie ulegają zmianie przy przesunięciu w czasie [RADZ97, ss. 465–466].

Charakterystyka modelu powinna odzwierciedlać charakterystykę źródła, które opisuje. Z tego powodu możemy klasyfikować źródła ze względu na rodzaje prawidłowości, które w nich występują, oraz modele ze względu na rodzaje prawidłowości, które są w stanie opisać.

³ Fazom modelowania i kodowania poświęcono dwa kolejne punkty tego podrozdziału. Dalej omówione zostaną przetwarzanie wstępne oraz transformacje. Zmieniona kolejność, w jakiej opisywane są poszczególne fazy, wynika po pierwsze z kluczowego znaczenia, jakie mają dla kompresji modelowanie i kodowanie, po drugie ze względu na to, że w opisie dwóch pierwszych faz kompresji odwołujemy się do pojęć, które wyjaśnione są w punkcie *Źródła danych i ich modele*.

Źródłem najprostszego rodzaju, które każdy symbol źródła generuje niezależnie od pozostałych z identycznym prawdopodobieństwem, jest symetryczne źródło Bernoulliego [SZPA99]. Jego model zapiszemy następująco:

$$P(a_i) = \frac{1}{|A|}, \quad i = 1, 2, \dots, |A|, \quad (1.5)$$

gdzie $|A|$ oznacza rozmiar alfabetu A .

Jeżeli źródło generuje poszczególne symbole z różnym prawdopodobieństwem, ale niezależnie od pozostałych, to określamy je jako asymetryczne źródło Bernoulliego [SZPA99]. Odpowiada mu następujący model:

$$P(a_i) = p_i, \quad i = 1, 2, \dots, |A|, \\ p_i \geq 0, \quad \sum_{i=1}^n p_i = 1. \quad (1.6)$$

W źródłach Bernoulliego nie występuje korelacja pomiędzy symbolami, dlatego nazywamy je źródłami bez pamięci (w literaturze anglojęzycznej – *memoryless sources*). W praktyce o wiele częściej spotyka się źródła z pamięcią (*conditional sources*).

Jednym z najpowszechniej stosowanych sposobów reprezentowania zależności w źródłach z pamięcią są modele Markowa. W modelu Markowa rzędu K przyjmuje się, że znajomość ostatnich K stanów jest równoważna wiedzy o całej historii procesu [SAYO02, s. 24]:

$$P(x_n | x_{n-1}, \dots, x_{n-K}) = P(x_n | x_{n-1}, \dots, x_{n-K}, \dots). \quad (1.7)$$

Entropia dla źródła Markowa X o j stanach X_i jest średnią wartości entropii dla poszczególnych stanów [SAYO02, s. 25]:

$$H(X) = \sum_{i=1}^j P(X_i) H(X_i). \quad (1.8)$$

Źródło z pamięcią można przedstawić jako automat o skończonej liczbie stanów (*finite state machine*, FSM). Jeżeli stany modelu odpowiadają sekwencjom symboli o skończonej długości, a przejścia pomiędzy stanami – symbolom alfabetu źródła, to mamy do czynienia z modelem FSM źródła danych [BUNT96, s. 26]. Sekwencję symboli źródła identyfikującą stan nazywamy **kontekstem**.

Model FSM jest markowski, jeżeli dla jego pojedynczego stanu istnieje nie więcej niż jedno przejście dla każdego symbolu alfabetu. Maksymalna liczba przejść z pojedynczego stanu jest wtedy równa rozmiarowi alfabetu źródła.

Gdy dla danego symbolu alfabetu istnieje więcej niż jedno przejście z pewnego stanu, to model taki określamy jako niemarkowski model FSM [BUNT96, s. 13].

Będzie tak na przykład wtedy, gdy wybór kolejnego stanu uzależniony jest nie tylko od aktualnego stanu modelu, ale również od stanów wcześniejszych.

Jeżeli w modelu FSM każdemu stanowi odpowiada dokładnie jeden kontekst, to model taki nazywamy modelem FSMX. W modelu FSMX, dla każdego kontekstu o niezerowej długości, istnieje kontekst nadrzędny krótszy od niego o dokładnie jeden symbol [BUNT96, s. 26]. Przykładem modelu FSMX jest model PPM, który posłuży jako jeden z modeli składowych metody predykcjno-substytucyjnej.

Przez **rzęd** modelu FSM rozumiemy maksymalną długość kontekstu, dla której istnieją stany tego modelu. Jeżeli rząd wynosi K , to nie są modelowane konteksty o długości $k > K$. Rozszerzając to pojęcie na modele Bernoulliego, modelowi asymetrycznemu przypiszemy rząd 0, natomiast symetrycznemu rząd -1 .⁴

Przedstawione do tej pory modele koncentrują się na opisie korelacji zachodzącej pomiędzy kolejnymi symbolami źródła. Nie jest to oczywiście jedyny rodzaj zależności mogący występować w źródłach danych.

Na przykład w zbiorach bazodanowych, w których poszczególne rekordy zapisane są sekwencyjnie, często silniejsza korelacja występuje pomiędzy polami tego samego typu sąsiadujących rekordów, aniżeli pomiędzy sąsiednimi polami tego samego rekordu. Kontekstem właściwym dla symbolu jest w takiej sytuacji nie sekwencja bezpośrednio poprzedzająca, ale oddalona od niego o pewną liczbę pozycji.

Inne rodzaje zależności nie mają jednak charakteru uniwersalnego – odnoszą się do jedynie do specyficznych typów danych. Dlatego ich zastosowanie w kompresji danych jest ograniczone.

Niekiedy, gdy źródło danych jest realizacją złożonych procesów, niełatwo jest opisać je przy pomocy jednego modelu. Można wtedy uznać je za źródło mieszane, będące połączeniem wielu źródeł, z których w danej chwili aktywne jest tylko jedno z nich. Opisać je można przy użyciu modelu mieszanego, nazywanego też modelem przełączanym. Istnieje w nim j cząstkowych źródeł X_i , z których każde posiada opisujący je model M_i , oraz przełącznik wybierający źródło X_i z prawdopodobieństwem P_i [SAYO02, s. 27]. Przykładem modelu mieszanego jest model użyty w metodzie predykcjno-substytucyjnej.

⁴ W starszych pozycjach literatury (np. [SHAN48]) spotkać można odmienną numerację rzędów, zaczynająca się nie od -1 , a od 0. Najprostszy symetryczny model Bernoulliego ma w niej rząd 0, asymetryczny model Bernoulliego ma rząd 1, model uwzględniający tylko konteksty o jednostkowej długości – rząd 2, itd.

1.1.4 Kodowanie

Kodowanie to proces przyporządkowania symbolom źródła słów kodowych, którymi są zwykle ciągi binarne. Aby kodowanie było efektywne, kody powinny być możliwie najkrótsze. Kod jest asymptotycznie optymalny, jeżeli dla danego rozkładu prawdopodobieństwa stosunek średniej długości kodu do entropii dąży do 1, gdy entropia dąży do nieskończoności.

Podstawowym problemem związanym z kodowaniem jest zapewnienie możliwości odtworzenia pierwotnych danych. Przede wszystkim kody powinny być unikalne. Kody są unikalne, jeżeli w alfabecie źródłowym nie ma dwóch różnych symboli o tym samym kodzie. Po drugie, kody powinny być jednoznacznie dekodowalne, to znaczy dowolna kombinacja słów kodowych powinna dać się odkodować w jednoznaczny sposób. Kody są jednoznacznie dekodowalne, gdy spełniona jest nierówność Krafta-McMillana [SAYO02, s. 33]:

$$K(C) = \sum_{i=1}^N 2^{-L_i} \leq 1. \quad (1.9)$$

gdzie C jest kodem składającym się z N słów kodowych o długościach $L_1, L_2, \dots, L_N$.

Kod, w którym do prawidłowego zdekodowania słowa kodowego nie jest potrzebna znajomość początku słowa po nim następującego, nazywamy kodem natychmiastowym [SAYO02, s. 29]. Warunkiem koniecznym i dostatecznym, aby dany kod był kodem natychmiastowym, jest by żadne słowo kodowe nie było przedrostkiem innego słowa kodowego [ABRA69, s. 66].

Ze względu na liczbę symboli na które przypada jedno słowo kodowe i długości słów kodowych, rozróżniamy cztery rodzaje kodów [LEHI87, s. 5]:

- o stałej długości słów kodowych dla stałej liczby symboli, np. kod ASCII,
- o stałej długości słów kodowych dla zmiennej liczby symboli, np. kod Tunstalla,
- o zmiennej długości słów kodowych dla stałej liczby symboli, np. kod Huffmana,
- o zmiennej długości słów kodowych dla zmiennej liczby symboli, np. kod Lempela-Ziva.

Kody mogą być tworzone dla konkretnej informacji (kiedy znane są konkretne prawdopodobieństwa), lub dla ogólnych typów informacji (kiedy znany jest jedynie przybliżony rozkład prawdopodobieństwa). Do pierwszej grupy zaliczamy na przykład kody Shannona-Fano [FANO49] i Huffmana [HUFF52]. Do drugiej, nazywanej kodami uniwersalnymi, zaliczamy między innymi kod unarny, kod Eliasa [ELIA75], kod Fibonacciego [APFR85], kod Golomba [GOLO66] i Rice'a [RICE79]. Ich podstawowe zastosowanie to kodowanie liczb całkowitych.

1.1.5 Przetwarzanie wstępne

Przetwarzanie wstępne obejmuje dwa rodzaje operacji: analizę danych źródłowych i ich motywowane semantycznie przekształcenie.

Celem analizy danych jest określenie pewnych właściwości źródła, które posłużą jako parametry usprawniające proces modelowania, lub zostaną wykorzystane do wyboru jednego z kilku dostępnych algorytmów dla dalszych faz. Przykładowo, analiza stosowana w programie Zip ma za zadanie zaklasyfikowanie zbioru danych do jednej z dwóch grup: tekstu albo danych binarnych [INFO99]. W zależności od ustalonej grupy, wykonywana jest kompresja o różnych parametrach pracy.

Analiza ma duże znaczenie dla programów wykorzystujących metody transformacyjne. Na przykład w programie SDC [DEOR03], w zależności od wykrytych własności źródła stosuje się różne algorytmy sortowania (najszybsze algorytmy dla przypadków typowych mają bardzo długie czasy działania w najgorszym przypadku), i modelowania danych po transformacji (inne dla tekstu i danych binarnych).

Najbardziej jednak oczywistym jest użycie analizy danych w połączeniu z motywowanym semantycznie przekształceniem. Rolą analizy jest tutaj określenie rodzaju przetwarzanych danych i wybranie odpowiadającego mu filtra.

Motywowane semantycznie przekształcenie, nazywane też filtrowaniem, wykorzystuje wiedzę o charakterystyce wskazanego typu danych w taki sposób, by poprawić efektywność modelowania go metodami uniwersalnymi, które nie uwzględniają semantyki źródła. Z tej racji, najbardziej użyteczne jest dla typów danych zawierających zależności specyficznego rodzaju, które nie mogą być w pełni wychwycone przez model uniwersalny.

Najbardziej rozpowszechnionym rodzajem motywowanego semantycznie przekształcenia jest filtrowanie danych sygnałowych (takich jak na przykład dźwięk i obrazy pochodzenia naturalnego). W ich przypadku, kolejne próbki zwykle różnią się między sobą minimalnie, co przekłada się na niską efektywność modelowania przy użyciu uniwersalnych modeli. Wystarczy jednak zastąpić absolutne wartości ich różnicami z wartością poprzednią (technikę tę nazywa się kodowaniem różnic, ang. *delta coding*), by uzyskiwany stopień kompresji uległ znacznej poprawie. Technikę tę stosuje większość nowych programów kompresujących (np. Rar [ROSZ03], GRZip [GREB03] czy UHARC [HERK02]).

Istotną poprawę efektywności kompresji poprzez przekształcenie wstępne uzyskać można także dla plików wykonywalnych (również stosowane w programach takich jak np. Rar, GRZip czy UHARC). Najprostszą stosowaną techniką jest

zastąpienie występujących w skompilowanym programie adresów względnych na bezwzględne. Bardziej skomplikowane techniki wykorzystują wiedzę o specyfice poszczególnych części, z których składa się plik wykonywalny danego systemu.

Podatnym na przekształcenia wstępne typem danych jest również tekst w języku naturalnym [GRAB99]. Do typowych przekształceń tekstowych zaliczamy konwersję znaków białych (np. zamiana tekstu w postaci „Ala ma kota” na postać „AlaMaKota”), konwersję wielkości liter (np. „Dzień Pierwszy” na „_dzień _pierwszy”), poszerzenie alfabetu o najczęściej spotykane wieloznaki, czy też zmianę kolejności symboli w alfabecie (na przykład przemieszczenie polskich znaków narodowych pomiędzy litery alfabetu łacińskiego, rozdzielenie samogłosek i spółgłosek, albo przemieszanie w alfabecie dużych i małych liter). Analogiczne reguły konwersji tworzyć można dla dowolnych, nie tylko naturalnych języków, pod warunkiem, że znana jest ich charakterystyka.

Przekształcenia wstępne są także wysoce skuteczne w przypadku zbiorów bazodanowych. Dla danych w postaci tabelarycznej (np. relacyjnych baz danych), podstawową zasadą działania jest zamiana wierszowego układu danych na kolumnowy [ABEL03]. W przypadku dokumentów XML stosuje się natomiast zamianę postaci kanonicznej na model zdarzeniowy ESAX [CHEN01].

1.1.6 Transformacje

Faza transformacji obejmuje przekształcenia danych źródłowych będące integralnym składnikiem określonej metody kompresji, których wykonanie jest konieczne przed przejściem do faz modelowania i kodowania.

Rezultatem przeprowadzenia tej fazy jest pewien pośredni zbiór danych. Niekiedy jest on krótszy od zbioru oryginalnego (co jest wymiernym dowodem, że kompresja nastąpiła), niekiedy zaś ma tę samą długość, lub nawet jest dłuższy, jednak zawarte w nim dane lepiej poddają się modelowaniu i są przez to efektywniej kodowane.

Faza transformacji jest typowa dla metod kompresji stratnej (np. dyskretne przekształcenie kosinusowe w JPEG [JPEG93]). Wśród metod bezstratnych występuje przede wszystkim w metodach opartych na transformacji blokowej Burrowsa-Wheelera [BUWH94]. Nie jest jednak do nich ograniczona. Na przykład dotyczy również metod substytucyjnych wykorzystujących algorytmy dwuprzebiegowe, w których da się wyodrębnić etap dokonywania substytucji i etap kodowania statystycznego zmodyfikowanych już danych.

Istotną kwestią jest rozgraniczenie właściwych transformacji od przekształceń przetwarzania wstępnego. Można go dokonać w oparciu o kryteria zamieszczone

w tabeli 1.1. Stosując je nie zawsze można jednoznacznie zakwalifikować dany etap algorytmu do określonej fazy; niekiedy przyporządkowanie takie ma więc charakter czysto umowny.

Tabela 1.1. Przekształcenia fazy przetwarzania wstępnego i fazy transformacji

Faza przetwarzania wstępnego	Faza transformacji
Wykonanie przekształcenia jest fakultatywne.	Niewykonanie przekształcenia czyni bezzasadnym wykonywanie dalszych etapów algorytmu.
To samo przekształcenie wstępne można łączyć z różnymi algorytmami kompresji, choć niekoniecznie z tym samym skutkiem.	Przekształcenie stanowi integralny składnik określonego algorytmu kompresji.
Faza modelowania wykorzystuje model uniwersalny.	Model użyty w fazie modelowania dostosowany jest do specyfiki rezultatu transformacji.
Przekształcenie oparte jest na wiedzy o semantyce określonego typu danych.	Przekształcenie może mieć uniwersalne zastosowanie (niezależne od typu danych).

Źródło: opracowanie własne.

1.1.7 Rola czasu w modelu

Mimo iż większość metod kompresji oparta jest na założeniu o stacjonarności źródła, w praktyce nader często obserwuje się przewagę korelacji lokalnej nad globalną. Zjawisko to, nazywane lokalnością referencji, można w uproszczeniu zapisać następująco:

$$\lim_{i \rightarrow \infty} P(x_j = x_{j+i} \mid x_j \notin \{x_{j+1}, \dots, x_{j+i-1}\}) = 0. \quad (1.10)$$

Zdaniem amerykańskiego badacza M. Mahoneya, o powszechności jego występowania świadczy choćby to, że jest ono uwzględnione w procesie uczenia u ludzi i zwierząt. Gdyby nie dotyczyło ono wielu rodzajów źródeł danych występujących w przyrodzie, mózg ludzki nie wyewoluował by w ten sposób [MAHO02].

Niektóre metody kompresji uwzględniają to zjawisko już w samym swym mechanizmie działania (na przykład Deflate [DEUT96a]). W innych wymaga to wykonywania dodatkowych operacji, takich jak [HOWA93]:

- częsty restart modelu (wada: za każdym razem traci się zbyt wiele statystyk);
- preferowanie statystyk pochodzących z ostatnio przetworzonej części danych (wada: zbyt skomplikowane obliczeniowo);
- kodowanie rang (wada: obniża efektywność kompresji);
- sukcesywne zwiększanie wartości o które powiększa się statystyki (tak zwane ekspotencjalne starzenie, wada: trudne w implementacji);
- okresowe skalowanie statystyk.

Ostatni z wymienionych sposobów jest prosty w implementacji i szybki w działaniu. Skalowanie jest procedurą polegającą na podzieleniu wszystkich liczników należących do skalowanego modelu o pewną stałą nazywaną **współczynnikiem skalowania**.

Skalowanie inicjowane jest w momencie przekroczenia przez inkrementowany licznik pewnej ustalonej wartości oznaczanej q_s , którą nazywamy **okresem skalowania**, a jej odwrotność ($1/q_s$) **częstotliwością skalowania**.

W metodach dynamicznych, częstotliwość i współczynnik skalowania są parametrami sterującymi szybkością aktualizacji modelu, to jest tym, jak szybko model dostosowuje się do modelowanego źródła. Trzecim takim parametrem jest **stała inkrementacji**. Jest to wartość, o którą zwiększa się licznik po każdorazowym wystąpieniu odpowiadającego mu symbolu. W jednym modelu można stosować różne stałe inkrementacji – przykładowo w PPMD ich wartość różni się w zależności od tego czy symbol pojawia się w danym modelu po raz pierwszy czy kolejny [HOWA93].

Oprócz występowania lokalności referencji, szybkość aktualizacji modelu ma szczególne znaczenie jeszcze w dwóch przypadkach:

- początkowej części źródła, gdy model jest pusty;
- zmiany charakterystyki źródła, gdy model staje się nieadekwatny.

Dobór parametrów aktualizacji modelu zależeć powinien od charakterystyki kodowanego źródła danych. Przykładowo:

- źródło stacjonarne bez lokalności referencji – niska częstotliwość skalowania, mała stała inkrementacji;
- źródło niestacjonarne z bardzo dużą lokalnością referencji – wysoka częstotliwość skalowania, duża stała inkrementacji;
- źródło heterogeniczne w którym naprzemiennie występują bloki o spójnej charakterystyce – częstotliwość skalowania uzależniona od wielkości bloków, mała stała inkrementacji, duży współczynnik skalowania.

1.1.8 Miary kompresji

Podstawowym kryterium oceny metody kompresji danych jest **efektywność kompresji**. Efektywność kompresji informuje jak bardzo udało się zmniejszyć objętość kodowanych danych; nie uwzględnia więc szybkości, z jaką kompresja jest wykonywana.

Do najczęściej używanych miar efektywności kompresji zaliczamy stopień kompresji, współczynnik procentowy kompresji i średnią bitową. **Stopień kompresji**

(ang. *compression ratio*, *CR*) definiuje się jako stosunek długości źródłowego zbioru danych (S) do długości zbioru skompresowanego (L) [CAPP85]:

$$CR = \frac{S}{L}. \quad (1.11)$$

Im wyższa wartość CR , tym lepszą efektywność kompresji osiągnięto.

Współczynnik procentowy kompresji (*compression percentage*, CP) zaproponował, choć w nieco odmienniej od przedstawionej tu postaci, Rubin w pracy [RUBI76]:

$$CP = \frac{S - L}{S} \cdot 100\%. \quad (1.12)$$

Im wyższa wartość CP , tym lepszą efektywność kompresji osiągnięto. CP przyjmuje wartości ujemne w przypadku, gdy długość zbioru wynikowego przekracza długość zbioru źródłowego, a wraz ze wzrostem efektywności kompresji zbliża się do wartości 100%, której wszak nigdy nie osiąga.

Średnią bitową (*bitrate*, BR) nazywamy średnią ilość bitów przypadającą na pojedynczy symbol (lub inną jednostkę) oryginalnego zbioru danych. W zależności od rodzaju kompresowanych danych, za jednostkę BR przyjmuje się bity na znak (bpc), bity na próbkę (bps), czy bity na punkt (bpp). W niniejszej pracy posługiwać będziemy się pierwszą z wymienionych jednostek, przyjmując 8 bitów na znak:

$$BR = \frac{8 \cdot L}{S}. \quad (1.13)$$

Im niższa wartość BR , tym lepszą efektywność kompresji osiągnięto.

Większość cytowanych pozycji literaturowych, zawiera pomiary wyłącznie efektywności kompresji, mierzonej najczęściej średnią bitową. Czas kompresji, jeżeli w ogóle jest podawany, to zwykle w postaci osobnego zestawienia, co uniemożliwia proste porównanie wyników z uwzględnieniem obu kryteriów.

Gdy chodzi o znalezienie najbardziej efektywnego algorytmu kompresji, takie podejście jest uzasadnione. Skoro jednak celem ma być ocena praktycznej przydatności algorytmu, takie podejście należy uznać za niewystarczające.

Możemy wyróżnić cztery najważniejsze cele użycia kompresji:

- związane z przesyłaniem danych:
 - 1) wymiana danych,
 - 2) dystrybucja danych,
- związane z przechowywaniem danych:
 - 3) archiwizacja,
 - 4) tworzenie kopii zapasowych.

W zależności od założonego celu, należałoby zatem użyć innej miary wydajności algorytmu uwzględniającej czas.

W przypadku transmisji danych, kompresja wykonywana jest zbliżoną liczbę razy do dekompresji. Nadawca kompresuje dane, odbiorca je dekompresuje. Motywem użycia kompresji danych jest zmniejszenie łącznego czasu transmisji. Dobrą miarą wydajności algorytmu kompresji będzie w tym przypadku całkowity czas transmisji, to jest suma czasów: kompresji, transmisji i dekompresji zbioru danych.

Przy dystrybucji danych, dekompresja wykonywana jest znacznie częściej aniżeli kompresja. Dane kompresowane są jednokrotnie przez nadawcę, natomiast dekompresowane wielokrotnie – przez każdego z odbiorców. Miarą wydajności algorytmu kompresji może być w tym przypadku całkowity czas dostarczenia odbiorcy zbioru danych, to jest suma czasów: transmisji i dekompresji. Czas kompresji nie jest całkowicie bez znaczenia, ale ma charakter warunku ograniczającego: jeżeli tylko nie przekracza założonych wartości skrajnych, jest akceptowalny.

W przypadku archiwizacji, dane raz skompresowane będą dekompresowane nieznana liczbę razy. Miara wydajności algorytmu kompresji w tym przypadku powinna obejmować cztery parametry: wielkość zbioru danych po archiwizacji, czasy jego kompresji i dekompresji, oraz zakładany stosunek pomiędzy liczbą wykonywanych operacji kompresji a liczbą wykonywanych operacji dekompresji.

Przy tworzeniu kopii zapasowych dokonuje się kompresji danych, które niekoniecznie będą kiedykolwiek dekompresowane. Miara wydajności algorytmu kompresji w tym przypadku powinna obejmować dwa parametry: wielkość kopii zapasowej oraz czas jej tworzenia (kompresji). Czas dekompresji nie jest całkowicie bez znaczenia, ale ma charakter warunku ograniczającego: jeżeli tylko nie przekracza założonych wartości skrajnych, jest akceptowalny.

Jak widać, efektywność kompresji nie wystarcza do pełnej oceny wydajności algorytmu kompresji. W praktyce powinno się brać ją pod uwagę w połączeniu z czasem wykonania.

Jeżeli prowadzone badania mają naturę ogólną, a więc nie da się jednoznacznie wskazać celu użycia kompresji, można posłużyć się miarami transmisji danych. Ich zalety są następujące:

- uwzględniają zarówno czas kompresji, jak i dekompresji danych;
- posiadają naturalną interpretację;
- nie wymagają wprowadzania dodatkowych założeń (np. stosunku pomiędzy liczbą wykonywanych operacji kompresji i dekompresji).

W odróżnieniu od efektywności kompresji, wyniki pomiarów czasu mogą być interpretowane wyłącznie z uwzględnieniem środowiska testowego. Co więcej, chcąc uzyskać miarę uwzględniającą i czas, i efektywność, trzeba dodatkowo znać proporcję, w jakiej te dwie wielkości połączyć. W przypadku miar transmisji, nośnikiem takiej proporcji jest szybkość, z jaką dane mogą być transmitowane.

Najprostszą miarą jakości uwzględniającą czas i efektywność kompresji jest wspomniany już całkowity czas transmisji [GILC02; JUKI03]:

$$CCT = t_k + t_t + t_d, \quad (1.14)$$

gdzie: t_k – czas kompresji, t_t – czas transmisji skompresowanego pliku, t_d – czas dekompresji.

Jednostką miary CCT jest sekunda (s). Wartość CCT interpretuje się: jak długo trwa łącznie transmisja danych z wykorzystaniem kompresji.

Wadą tej miary jest konieczność interpretacji wyników z uwzględnieniem wielkości pliku. Można tego uniknąć korzystając z miary jaką jest całkowita szybkość transmisji:

$$CST = \frac{S}{CCT}, \quad (1.15)$$

gdzie: S – długość pliku przed kompresją.

Jednostką miary CST jest bajt przez sekundę (B/s). Wartość CST interpretuje się: jak szybko można przesyłać dane z wykorzystaniem kompresji.

Jakkolwiek CST ułatwia porównywanie wyników dla plików o różnych długościach, nadal jednak niełatwe jest porównywanie wyników uzyskanych dla różnych prędkości transmisji. Miarą wolną od tej wady jest współczynnik przyspieszenia transmisji:

$$WPT = \frac{CST}{v_t} = \frac{t_{nt}}{CCT}, \quad (1.16)$$

gdzie: v_t – szybkość transmisji nieskompresowanego pliku, t_{nt} – czas transmisji nieskompresowanego pliku.

Jednostka miary WPT jest niemianowana (1). Wartość WPT interpretuje się: ile razy szybciej dane przesyłane są z wykorzystaniem kompresji, niż byłyby bez niej.

1.2 Podział metod kompresji danych

1.2.1 Kryterium wiernego odtworzenia

Podstawowym kryterium podziału metod kompresji jest możliwość wiernego odtworzenia danych oryginalnych z postaci skompresowanej.

Jak zostało to już napisane wcześniej, typowe dane zawierają w sobie informację nieistotną (nadmiarowość semantyczną) i powieloną (nadmiarowość stochastyczną). Właśnie usunięcie tych zbędnych składników ma za zadanie kompresja danych.

Eliminacja informacji powielonej, o ile pozostawia informację o fakcie jej dokonania, pozwala na późniejsze odtworzenie danych w ich oryginalnej postaci. Metody, które to umożliwiają, nazywane są bezstratnymi (ang. *lossless*), gdyż mimo ich zastosowania nie traci się nic ze zbioru danych wejściowych. Ich oczywistą zaletą jest uniwersalność – można nimi kompresować dane bez potrzeby wcześniejszej interpretacji tychże.

Kiedy celem kompresji ma być usunięcie informacji nieistotnej, pojawia się problem ustalenia znaczenia, jakie wskazane dane niosą dla odbiorcy. Z uwagi na mnogość możliwych interpretacji tych samych danych, eliminacji informacji nieistotnej można dokonać tylko wtedy, gdy znane jest zastosowanie kompresowanych danych. Może to być jednostkowa aplikacja, częściej jednak tworzy się rozwiązania dla danych o typowych zastosowaniach – takich jak na przykład multimedia, obrazy, dźwięk. Jako że elementarnym składnikiem takiej operacji jest bezpowrotne usunięcie części informacji – uznanej za nieistotną, nazywamy ją kompresją stratną (ang. *lossy*).

Dane skompresowane metodami kompresji stratnej nie wystarczają do odtworzenia danych oryginalnych. Nie stanowi to jednak przeszkody, o ile informacja uzyskana ze skompresowanych danych spełnia wymogi zastosowania. Przykładowo, usunięcie z pamiątkowej fotografii trudno dostrzegalnych szczegółów nie ma zwykle jakiegokolwiek znaczenia. Ale usunięcie takich szczegółów ze zdjęć rentgenowskich mogłoby doprowadzić do postawienia przez lekarza błędnej diagnozy. Odwrotnie, usunięcie ze zdjęcia rentgenowskiego wszystkiego poza okolicami obszaru chorobowo zmienionego, nie wpłynie na jego przydatność diagnostyczną. Z kolei pozostawienie na pamiątkowej fotografii wyłącznie twarzy przedstawionych osób, zrujnowałoby jej estetykę i uczyniłoby ją praktycznie bezwartościową.

Aby dokonać usunięcia obu typów zbędnej informacji, to jest zarówno nieistotnej jak i powielonej, metody stratne łączy się zazwyczaj z metodami bezstratnymi.⁵ Znane z popularnych zastosowań algorytmy kompresji stratnej mają przez to zarówno etapy stratne jak i bezstratne (patrz np. algorytm kompresji obrazów JPEG, który przedstawimy w punkcie 1.2.4).

Kompresja stratna usuwa ze zbioru danych źródłowych więcej niż czyni to jej bezstratna odpowiedniczka. Oczywistą konsekwencją tego stanu rzeczy jest wyższa

⁵ Metoda będąca rezultatem takiego połączenia pozostaje oczywiście stratna.

efektywność metod kompresji stratnej. Należy tu jednak przypomnieć, iż metody kompresji stratnej nie mają charakteru uniwersalnego – poddać im można tylko niektóre typy danych.

1.2.2 Klasyfikacja metod kompresji bezstratnej

Kryterium ogólności stosowania. W zależności od tego, czy przeznaczeniem metody jest kompresja danych dowolnego rodzaju, czy tylko jednego wybranego typu danych, rozróżniamy metody uniwersalne i dedykowane. Metody uniwersalne opierają się w swym działaniu wyłącznie na modelu statystycznym źródła danych, który określa jednolitą zasadę budowy modelu dla każdego typu danych generowanego przez takie źródło. Metody dedykowane uwzględniają także semantykę, która jest specyficzna dla poszczególnych typów danych.

Metodami łączonymi nazwiemy metody uniwersalne, które dysponują pewną liczbą modeli dedykowanych, użytecznych w przypadku napotkania odpowiadających im danych. Zwykle, choć nie zawsze, modele dedykowane wykorzystywane są w nich poprzez przetwarzanie wstępne.

Kryterium adaptacyjności modelu. Ze względu na to, czy rozkład prawdopodobieństw symboli, w używanym modelu źródła, jest z góry ustalony, czy też dostosowywany jest do przetwarzanych danych, rozróżniamy metody z modelem stałym i adaptacyjnym.

Metody z modelem stałym posługują się zawsze takim samym modelem, niezależnie od rzeczywistej zawartości zbioru danych. Model taki może być zbudowany na przykład w oparciu o znaną charakterystykę dużej liczby zbiorów pewnego typu danych.

Tabela 1.2. Model stały a adaptacyjny – porównanie

Model stały	Model adaptacyjny
Wąski zakres stosowania – wyłącznie dla źródeł zgodnych z posiadanym modelem.	Szeroki zakres stosowania.
Do rekonstrukcji danych źródłowych potrzebny jest model użyty do kompresji.	Do rekonstrukcji danych źródłowych wystarcza ich zakodowana postać.
Wersja modelu dekodera musi być zgodna z wersją modelu kodera.	Nie występuje problem wersji modelu.
Kompresja jest prosta, gdyż nie wymaga modelowania	Kompresja jest skomplikowana, gdyż wymaga modelowania

Źródło: opracowanie własne.

Metody z modelem adaptacyjnym posługują się osobnym modelem dla poszczególnych zbiorów danych. Budują go w oparciu o zaobserwowaną charakterystykę przetwarzanych danych. Wady i zalety obu podejść prezentuje tabela 1.2.

Klasyczne metody z modelem adaptacyjnym rozpoczynają kompresję z modelem pustym, który staje się efektywny dopiero po przetworzeniu pewnej liczby danych. Najwyższą efektywność kompresji zaobserwowano jednak [JUKI03] dla programów (takich jak np. Durilca [SZKA03]), w których już model początkowy zawiera pewną przybliżoną charakterystykę źródła danego typu. Taki pośredni – pomiędzy modelem stałym a adaptacyjnym – przypadek określimy jako metodę z modelem adaptacyjnym o niezerowym stanie początkowym. Modele takie osiągają dobrą efektywność już od samego początku przetwarzania danych, pod warunkiem że model początkowy dobrze te dane opisuje.

Kryterium czasu budowy modelu. Rozróżniamy trzy rodzaje metod z modelem adaptacyjnym, ze względu na czas, w którym budują one modele [LEHI87, ss. 7–8]:

- metody statyczne⁶ (określane też jako *off-line* [BUNT96]),
- metody dynamiczne (określane też jako *on-line* [BUNT96]),
- metody semidynamiczne.

Metody statyczne wykonują kompresję w dwóch przebiegach, z których pierwszy poświęcony jest budowie modelu, a drugi zakodowaniu danych w oparciu o gotowy już model. W trakcie kodowania model jest niezmienny, co oznacza, że model nie odzwierciedla zmian zachodzących w charakterystyce źródła. Dane zakodowane poprzedzone są zapisem samego modelu, gdyż od samego początku dekompresji musi być on znany dekodownikowi. Przykładem implementacji takiej metody jest algorytm Huffmana zastosowany w programie PKZip do kodowania danych po wykonaniu substytucji [PKWA02].

Metody dynamiczne wykonują kompresję w jednym przebiegu, w którym fazy modelowania i kodowania występują naprzemiennie. Modyfikacja modelu następuje wraz z każdym przetworzonym symbolem źródła, a więc model odzwierciedla zmiany zachodzące w charakterystyce źródła. Dekoder rekonstruuje model w sposób analogiczny do kodera, tylko w oparciu o już przetworzone dane, co oznacza że nie ma potrzeby zapisu samego modelu. Do tego rodzaju zaliczymy na przykład algorytm PPMII zaimplementowany w programie PPMd [SZKA02b].

⁶ W literaturze, terminy statyczny oraz dynamiczny/adaptacyjny stosuje się jako równoprawne zarówno przy podziale ze względu na kryterium adaptacyjności, jak i czasu budowy modelu. W celu poprawienia przejrzystości omawianej klasyfikacji, w niniejszej pracy posługujemy się parą określeń (stały, adaptacyjny) dla rozróżnienia metod statycznych opartych na gotowym modelu i budujących model, oraz parą określeń (statyczny, dynamiczny) dla rozróżnienia metod budujących model w trakcie lub przed rozpoczęciem kodowania.

Metody semidynamiczne wykorzystują jednocześnie kilka modeli statycznych. Przetwarzane dane dzielą na fragmenty, z których każdemu przyporządkowują ten model, dla którego osiąga się najlepszą efektywność kompresji. Kompresja takimi metodami wymaga zwykle co najmniej kilku przebiegów, w których modele są budowane, oraz jednego przebiegu finalnego, w którym następuje dobór gotowych modeli do poszczególnych fragmentów oraz kodowanie. Przykładem implementacji takiej metody jest algorytm Huffmana z wieloma drzewami zastosowany w programie Bzip2 do kodowania danych po transformacjach BWT-MTF [SEWA00].

Kryterium sposobu działania. Kluczowe dla efektywnej kompresji jest wykorzystanie różnego rodzaju korelacji występującej tak pomiędzy poszczególnymi symbolami źródła (skala mikro), jak jego jednostkami strukturalnymi różnego stopnia uogólnienia (skala makro). Obecnie, cechy strukturalne źródła wykorzystuje się w kompresji przede wszystkim na trzy sposoby: predykcyjny, substytucyjny oraz transformacyjny.

Intuicyjnie najbardziej oczywisty sposób to tworzenie na tyle złożonych modeli, by były one w stanie odwzorować zależności występujące między symbolami. Takie podejście cechuje metody predykcyjne (np. PPM). Budowa wiernych, a przez to obszernych modeli probabilistycznych źródła oznacza duże wymagania co do zasobów systemowych. Dużą popularność zdobyły sobie dlatego rozwiązania alternatywne.

Rozbudowa alfabetu stosowana jest w substytucyjnych metodach kompresji (np. LZW). W dużym uproszczeniu polega ona na powiększeniu alfabetu źródła o nowe symbole, zastępujące często spotykane sekwencje symboli oryginalnego alfabetu.

Z kolei ideę metod transformacyjnych można generalnie przedstawić następująco: zamiast dopasowywać model do danych, dopasujmy dane do modelu. Aby efektywnie kompresować dane z zastosowaniem prostego modelu statystycznego rzędu 0, wystarczy je wpierw zdekorelować. Odbywa się to dwuetapowo. Wpierw, w oparciu o transformację sortującą (np. BWT), zmienia się kolejność symboli źródła, tak by symbole identyczne trafiły blisko siebie tworząc skupiska. W ten sposób zamienia się korelację globalną na lokalną. Następnie stosuje się prosty algorytm (np. MTF), który usuwa korelację lokalną.

Wymienione wyżej trzy sposoby nie są jedynymi spotykanymi. Na przykład metody oparte na anty-słownikach posługują się „odwrotnym” modelem źródła, *anty-słownikiem* zawierającym sekwencje symboli, które nigdy (lub prawie nigdy) się w nim nie pojawiają [CROC00]. Z kolei metody gramatyczne dla danego zbioru danych generują bezkontekstową gramatykę zawierającą zestaw reguł potrzebnych do jego pełnego odtworzenia, która jest następnie kodowana [KIYA02]. Zamiast klasycznego

modelu probabilistycznego stosowano też sieci neuronowe, uzyskując porównywalną efektywność kompresji, jednak przy znacznie dłuższym czasie działania [MAHO00]. Innymi metodami nie będziemy się dalej zajmować ze względu na ich marginalne znaczenie i brak bezpośredniego związku z nową metodą.

1.2.3 Stratna kompresja danych

Gdy operuje się metodami kompresji stratnej, pojawia się niezwykle ważny aspekt jakościowy. Dotyczy on granicy, którą oddzielamy istotną część danych od części nieistotnej. Dla wielu zastosowań, a szczególnie tam, gdzie końcowym odbiorcą informacji jest człowiek, granica pomiędzy tymi dwoma częściami jest rozmyta. Przesuwanie tej linii podziału pozwala na wybór priorytetu kompresji, pomiędzy wiernością oryginałowi a efektywnością kompresji. Zwykle możliwa jest płynna regulacja jakości, co pozwala dostosować objętość danych do posiadanych warunków technicznych. Dzięki temu dysponując nawet łączem o niskiej przepustowości lub nośnikiem o niewielkiej pojemności można nadal korzystać z technologii cyfrowych, tracąc jedynie na uzyskiwanej jakości.

W niektórych zastosowaniach dla typowych rodzajów danych znaleźć można pewną naturalną linię podziału, pozwalającą na usunięcie części danych obiektywnie (z uwzględnieniem charakteru zastosowania) uznanej za zbędną. Mówi się wtedy o kompresji praktycznie bezstratnej. Na przykład, gdy zniekształcenia obrazu wynikające z zastosowania stratnej metody kompresji są niedostrzegalne dla ludzkiego oka, mówi się o kompresji wizualnie bezstratnej. Korzystamy z takiego podejścia, gdy priorytetem jest jednoznacznie jakość, a jednocześnie dopuszczalne są pewne zniekształcenia w stosunku do oryginału.

Problemem stopnia zniekształceń źródła informacji zajmuje się specjalna gałąź teorii informacji (nazywana po angielsku *rate-distortion theory*). Podwaliny pod nią położył C. Shannon. W swojej pracy [SHAN48, s. 48] wykazał on, że dla danego źródła (o znanych własnościach statystycznych) i danej miary zniekształceń istnieje funkcja stopnia zniekształcenia $BR(D)$ (ang. *rate-distortion function*). Jeżeli za D przyjmiemy tolerowany poziom zniekształceń, to $BR(D)$ wyznacza najniższą możliwą do osiągnięcia średnią bitową. Kiedy kompresja jest bezstratna (nie ma zniekształceń, $D = 0$), to najniższa osiągalna średnia bitowa równa jest entropii źródła:

$$BR(D) = H. \quad (1.17)$$

W praktyce stosuje się wiele różnych miar zniekształceń [PRZE98, ss. 118–120]. Obok miar obliczeniowo obiektywnych stosuje się także miary subiektywne, w których oceny jakości zrekonstruowanej informacji dokonują ludzie.

1.2.4 Metody stratnej kompresji danych

Do podstawowych metod kompresji stratnej zaliczamy: metody predykcyjne, kwantowanie, transformacje, metody wykorzystujące fraktale oraz kodowanie percepcyjne.

Metody predykcyjne opierają swoje działanie na zastąpieniu rzeczywistych danych współczynnikami funkcji najlepiej te dane opisujących. Metody takie sprawdziły się między innymi w przypadku kompresji mowy ludzkiej (np. metoda predykcji liniowej LPC) [KRUP98, s. 57].

Kwantowanie to technika usuwania części informacji poprzez reprezentację określonego podzbioru wartości jednym z elementów tego podzbioru (kwantowanie skalarowe) lub podzbiorem tego podzbioru (kwantowanie wektorowe). W większości technik kompresji stratnej to właśnie kwantowanie przesądza o redukcji wielkości zbioru danych. Przykładowo, w metodzie LPC poddaje się kwantowaniu parametry modelu kanału głosowego [KRUP98, s. 59], a w algorytmie kompresji obrazów JPEG – dane po transformacji DCT [JPEG93].

Transformacje mają na celu takie przekształcenie danych wejściowych, aby lepiej poddawały się następującemu po nich kwantowaniu, to znaczy by przy zadanym stopniu kwantyzacji tracona była mniejsza część informacji istotnej. Transformacje stosowane są na przykład w kompresji obrazu, w postaci dyskretnego przekształcenia kosinusowego (DCT) [SAYO02, s. 398] czy dyskretniej transformaty falkowej (DWT) [SAYO02, s. 476]. Ich zadaniem jest tam rozłożenie sygnału wejściowego na podpasma, co umożliwia późniejsze zredukowanie mniej zauważalnej dla odbiorcy części widma w stopniu większym od części bardziej zauważalnej.

Metody nazywane fraktalnymi [SKAR98, s. 175] opierają się na wykorzystaniu samopodobieństwa występującego w niektórych rodzajach danych, szczególnie pochodzenia naturalnego. Podstawowym zastosowaniem metody fraktalnej jest kompresja obrazów (także ruchomych).

Kodowanie percepcyjne dotyczy tych typów danych, których docelowym odbiorcą jest człowiek. Posługując się wiedzą o niedoskonałości ludzkiego systemu percepcji, usunąć można tę część informacji, która jest przezeń ignorowana. W przypadku obrazów wykorzystuje się niewrażliwość oka odbiorcy na usunięcie obiektów bardzo małych (poprzez zmniejszenie rozdzielczości, ang. *downsampling*) i w niewielkim stopniu różniących się od swojego otoczenia (poprzez kwantowanie). W przypadku dźwięku szerokopasmowego usuwa się dźwięki zagłuszane przez inne występujące równocześnie lub sąsiadujące w czasie.

Pojedynczy algorytm kompresji stratnej łączy w sobie często kilka metod kompresji, tak stratnych, jak i bezstratnych. Przykładowo, popularny algorytm kompresji obrazów JPEG składa się z następujących etapów [JPEG93]:

- 1) Bezstratna⁷ transformacja wstępna: Przekształcenie z formatu zapisu barw RGB na format z wyodrębnioną składową luminancji.
- 2) Stratne kodowanie percepcyjne: Oko ludzkie jest bardziej czułe na zmiany jasności niż koloru, dlatego zmniejsza się rozdzielczość map chrominancji.
- 3) Bezstratna transformacja główna: DCT dla bloków punktów obrazu o wymiarach 8x8. Rezultatem jest macierz zawierająca współczynniki DCT odpowiadające energii sygnału w poszczególnych pasmach częstotliwości.
- 4) Stratne kwantowanie. Każdy z elementów macierzy dzielony jest przez odpowiadającą mu liczbę z tablicy kwantyzacji. Wyższym częstotliwościom odpowiadają wyższe współczynniki kwantyzacji (kodowanie percepcyjne).
- 5) Bezstratne kodowanie. Algorytmem Huffmana lub arytmetycznie.

1.3 Zastosowania kompresji danych

1.3.1 Obszary zastosowań metod kompresji danych

Obszar zastosowań metod kompresji danych podzielić można na dwie dziedziny:

- Zastosowania związane z podstawowym zagadnieniem kompresji danych, to jest minimalizacją objętości danych. Ich celem jest przyspieszenie dostępu do danych, zwiększenie prędkości transmisji i lepsze wykorzystanie nośników danych poprzez redukcję objętości danych.
- Zastosowania nie związane bezpośrednio z minimalizacją objętości danych, w których metody kompresji danych wykorzystuje się w zupełnie odrębnych dziedzinach nauki.

Pewnym wspólnym obszarem dla obu tych grup są te zastosowania, w których użycie metod kompresji z przesłanek związanych z minimalizacją objętości danych przynosi bardzo korzystne efekty uboczne.

Na przykład w kryptologii, wcześniejsze poddanie szyfrowanych danych kompresji utrudnia złamanie szyfru [WILS94]. Po pierwsze, w wyniku kompresji danych, uzyskuje się zbiór danych o rozkładzie prawdopodobieństw poszczególnych symboli zbliżonym do jednostajnego, co czyni nieskutecznymi proste ataki statystyczne. Po drugie, dzięki podwójnemu kodowaniu (szyfrowanie poprzedzone kompresją),

⁷ Jeżeli pominiemy niedokładność biorącą się z przybliżania wyników obliczeń.

nawet znajomość odszyfrowanego fragmentu przekazu nie przyda się w łamaniu szyfru, o ile nie jest znana skompresowana postać tego fragmentu.

Kompresję w połączeniu z szyfrowaniem wykonuje się zawsze jako pierwszą. Dane zaszyfrowane dobrym algorytmem kryptograficznym nie dziedziczą własności statystycznych źródła, co czyni je słabo kompresowalnymi. Właściwość tę wykorzystać można do oceny jakości algorytmu szyfrującego – jeżeli dane zaszyfrowane poddają się efektywnej kompresji, dyskwalifikuje to użyty algorytm szyfrujący.

Typowy przykład korzystnych efektów ubocznych to przyspieszenie procesu przetwarzania danych. Wynika ono z szeregu przyczyn, między innymi przyspieszenia operacji wejścia-wyjścia (zbiory danych są krótsze, a więc maleje czas wyszukiwania, odczytu i zapisu), zmniejszenia liczby wykonywanych operacji wejścia-wyjścia (bufor może pomieścić więcej danych w postaci skompresowanej), standardowe operacje (wyszukiwanie, kopiowanie) na skompresowanych danych ze względu na ich długość wykonują się szybciej (porównanie dwubajtowego identyfikatora ze wzorcem odbywa się znacznie szybciej niż ciągu kilkudziesięciu znaków). Efekt przyspieszenia procesu przetwarzania danych w szczególności dotyczy zapytań w systemach bazodanowych, zarówno relacyjnych [RAY95], jak i opartych na XML, gdzie uzyskano kilkukrotne przyspieszenie wykonywania zapytań [TOHA01].

1.3.2 Podstawowy obszar zastosowań metod kompresji danych

Zamieszczony poniżej przegląd najważniejszych zastosowań kompresji, w których chodzi o minimalizację objętości danych, z pewnością nie wyczerpuje tego tematu, daje jednak wyobrażenie o skali intensywności z jaką metody kompresji danych weszły do powszechnego użytku.

Najbardziej oczywisty obszar zastosowań kompresji to technologie informatyczne. Zacząć tu należy od uniwersalnych narzędzi kompresji danych nazywanych archiwizatorami [PIND98]. Programy takie jak Zip [INFO99] czy Rar [ROSZ03] są powszechnie używane, gdyż pozwalają zmniejszyć przestrzeń dyskową zajmowaną przez dane, jak również oszczędzić czas związany z przesłaniem danych poprzez Internet. Ich dodatkowe zalety to:

- ułatwienie wymiany i operacji na kolekcji zbiorów danych: poprzez umieszczenie grupy plików w jednym archiwum łatwo odseparowujemy je od innych grup plików i możemy w łatwy sposób przenosić je i kopiować odwołując się do pliku zbiorczego;
- dodatkowa (oprócz wynikającej z efektywności kompresji) oszczędność miejsca na dysku: zwykle pliki zajmują faktycznie więcej miejsca niż wynosi ich długość,

ze względu na granulację przestrzeni dyskowej (rozmiar jednostki alokacji – klastra dyskowego) i ich opis w tablicy alokacji plików;

- naprawa niedostatków systemu operacyjnego: w przypadku wielu archiwizatorów można przechowywać w jednym archiwum wiele wersji pliku o tej samej nazwie, czy też opatrywać zbiory danych komentarzami.

Dzięki dostępności bezpłatnych bibliotek przeznaczonych do obsługi popularnych typów archiwów (np. [GAAD95]), wiele programów użytkowych w sposób przezroczysty dla użytkownika obsługuje archiwa. Na przykład projekty w języku Java zapisywać można w plikach JAR, będących w istocie archiwami w formacie Zlib.

Użytkownicy mogą korzystać z tego typu narzędzi również w sposób dla nich niezauważalny, korzystając z takich programów jak na przykład DriveSpace Microsoftu, które kompresują wszystkie dane zapisywane na dysku. Skompresowana partycja dysku pomieścić może nawet kilkakrotnie więcej danych, niż wynosi jej fizyczna pojemność. Co więcej, o ile suma czasu odczytu z dysku skompresowanych danych i czasu ich dekompresji nie przekracza ich czasu odczytu bez kompresji, osiąga się przyspieszenie dostępu do zapisanych w ten sposób danych.

Również wiele popularnych formatów wymiany danych wykorzystuje algorytmy kompresji, w przezroczysty dla użytkownika sposób pozwalając na zmniejszenie wielkości dokumentów, na przykład pliki pomocy *on-line* (CHM), animacje Macromedia Flash (SWF), czy grafika wektorowa Corel Draw (CDR).

Duża część danych, w szczególności w systemach korporacyjnych, przechowywana jest w systemach bazodanowych. Pomysły użycia kompresji do redukcji rozmiaru baz danych mają historię sięgającą wczesnych lat 70-tych XX wieku [RUKE72]. Już P. A. Alsberg w swojej pracy [ALSB75] zwrócił uwagę na przyspieszenie szybkości dostępu do danych w systemach bazodanowych, które można osiągnąć dzięki kompresji. Przykłady praktycznej implementacji kompresji w systemach bazodanowych przedstawili Lynch i Browning [LYBR81] i D.G. Severance [SEVE83].

W 1985 roku G. V. Cormack przedstawił metodę kompresji poszczególnych rekordów [CORM85], która została zaimplementowana w systemie zarządzania informacją (IMS) firmy IBM. W późniejszych latach zwrócono uwagę na to, że lepsze efekty daje kompresja poszczególnych pól rekordów [GRSH91]. W ten sposób możliwe jest wykonywanie zapytań bazodanowych na skompresowanych danych, bez potrzeby ich dekompresji. W pracy [RAY95] wykazano, że oszczędność czasu wynikająca ze zmniejszenia liczby operacji dyskowych przewyższa wydłużenie czasu przetwarzania (ze względu na konieczność dekompresji).

Ze względu na obszerność baz danych, złożoność wykonywanych zapytań i wymagany szybki czas reakcji, bardzo odpowiednim miejscem dla implementacji metod kompresji są hurtownie danych [GOYA99].

Kompresować można nie tylko dane przechowywane na pamięciach masowych, ale również te znajdujące się w pamięci operacyjnej [DOUG93]. Część danych, która nie jest aktualnie przetwarzana może być poddana kompresji. Usprawnia to obróbkę wielkich zbiorów danych, które inaczej z trudem mieściłyby się w pamięci operacyjnej, a także zwalnia część pamięci na potrzeby innych aplikacji. Stwarza to pozory powiększenia rozmiaru pamięci fizycznej komputera. Redukcji podlega liczba odwołań do pamięci wirtualnej, co wpływa na ogólne przyspieszenie i zwiększenie komfortu pracy.

Transmisja danych poprzez sieci komputerowe bardzo często wiąże się z wykorzystaniem kompresji. Jest ona uwzględniona w specyfikacjach protokołów różnych warstw – począwszy od Point-to-Point Protocol (PPP) [SIMP94], poprzez Internet Protocol (IP) [POST81], a skończywszy na HyperText Transfer Protocol (HTTP) [FIEL97].

Ze względu na jego specyfikę i zakres użycia, pierwotnie wybranym dla implementacji kompresji protokołem był PPP. W protokole PPP Compression Control Protocol (CCP) [RAND96] określono zasady negocjacji metody kompresji pakietów PPP. Do najbardziej popularnych metod zaliczamy Stac LZS [FRSI96], BSD (oparte na uniksowym programie Compress) [SCHR96] i Microsoft Point-To-Point Compression (MPPC) [PALL97].

Wraz z rozpowszechnieniem się szyfrowanej transmisji danych wystąpił opisany wcześniej problem nieefektywności kompresji zaszyfrowanych danych. Pakiety PPP zawierały dane już zaszyfrowane, co czyniło próbę ich kompresji bezcelową. W związku z tym postanowiono umożliwić kompresję na poziomie datagramów IP. Zasady jej działania określa protokół IP Payload Compression Protocol (IPComp) [SHAC98]. Jej dwie podstawowe metody to Deflate i LZS.

Standard HTTP 1.1 określa zasady kodowania zawartości, definiując jako trzy podstawowe metody: Gzip, Compress i Deflate [FIEL97]. Korzystanie z zakodowanej zawartości możliwe jest dzięki określeniu standardowych formatów dla skompresowanych danych: Zlib [DEGA96] (używany przez program Zip), Gzip [DEUT96b] (używany przez program Gzip) i Deflate [DEUT96a] – wewnętrzny format opisu danych skompresowanych metodą Deflate, wykorzystywaną zarówno w Gzipie jak i w Zipie.

Kompresji poddaje się nie tylko dane przenoszone w pakietach, ale również same nagłówki tych pakietów. Najczęściej wykorzystuje się tu fakt niewykorzystywania całej szerokości pól przeznaczonych na zapis pewnych wartości opisujących pakiet. Standard PPP definiuje dwa rodzaje takiej kompresji: Address-and-Control-Field-Compression (ACFC) i Protocol-Field-Compression (PFC) [SIMP94]. Analogiczne rozwiązania opracowano dla IP (IP Header Compression, IPHC), TCP/IP oraz IPX.

Komutowany dostęp do Internetu poprzez linię telefoniczną również wiąże się z wykorzystaniem kompresji danych. Będące w powszechnym użytku standardy kompresji dla komunikacji modemowej to CCITT V42bis oraz Microcom MNP-5 i MNP-7 [IDG99, s. 22].

Innym pokrewnym punktem zastosowania są systemy *cache*, gdzie użycie kompresji znacznie przyspiesza dostęp do dokumentów poprzez jednoczesne zmniejszenie średniej wielkości dokumentu i zwiększenie liczby dokumentów mieszczących się w pamięci podręcznej (co wprost przekłada się na wzrost prawdopodobieństwa trafień *cache*) [VEVE98]. Przykładowo, jeżeli stopień kompresji wynosi 3:1, to w przypadku dokumentu, który już wcześniej znajdował się w *cache*'u prędkość dostępu wzrosłaby po kompresji trzykrotnie (przyjmując, że czas dekompresji jest pomijalny). Ze względu na kompresję, pojemność *cache*'u wzrosła by również trzykrotnie, a z nią liczba dokumentów, które są w nim przechowywane. Jeżeli stosunek czasu dostępu do dokumentu, do czasu dostępu do jego kopii na serwerze *cache* wynosi na przykład 5:1, to dla każdego takiego dokumentu uzyskano by 15-krotny wzrost prędkości dostępu.

Rozwój biologii molekularnej stworzył duże zapotrzebowanie na algorytmy umożliwiające kompresję łańcuchów RNA i DNA przechowywanych w bazach danych ośrodków badawczych. Wynikało ono z dużej objętości tych danych – przykładowo genom człowieka zawiera około trzy miliardy znaków (oznaczanych 'A', 'C', 'G', 'T') zapisane w 23 parach chromosomów. Z uwagi na to, że uniwersalne metody kompresji danych wykazywały na tym polu słabą efektywność, opracowano nowe techniki pozwalające na lepszą kompresję danych tego typu [GRTA94].

Obecnie dużą część danych cyfrowych stanowią dane sygnałowe pochodzenia naturalnego, takie jak mowa ludzka, dźwięk, obrazy i film (obrazy ruchome i dźwięk). Zbiory danych tego typu mają często bardzo dużą objętość, co czyni ich kompresowanie nieomal koniecznością [SWAC02b, s. 412].

Różnego rodzaju metody kompresji mowy [KRUP98, ss. 55–63] stosowane są w telefonii komórkowej (GSM), cyfrowej telefonii stacjonarnej (ISDN), telefonii

internetowej (Voice over IP), dyktafonach, czy automatach informacyjnych (np. w środkach masowej komunikacji).

Metody kompresji dźwięku szerokopasmowego [KRUP98, ss. 43–54] znalazły zastosowanie w rozpowszechnianiu muzyki poprzez Internet (MPEG-1 Layer 3 – MP3), internetowym radiu (RealAudio), radiu cyfrowym, kodowaniu ścieżek dźwiękowych filmów (MP3, AC-3), cyfrowych kasetach (DCC) i płytach (MP3-CD).

Kompresja obrazów to przede wszystkim powszechnie spotykane w Internecie formaty wymiany grafiki – GIF, JPEG i w mniejszym stopniu PNG [SWAC01]. Wyszły one poza sferę zastosowań związanych z komputerami, trafiając między innymi do cyfrowych aparatów fotograficznych. Kompresja wykorzystywana jest także przy kodowaniu danych przesyłanych faksem (standardy CCITT, w tym JBIG) i archiwach dokumentów (TIFF, DjVu [BOTT98]). Bezstratne algorytmy kompresji znajdują zastosowanie w przypadku zdjęć satelitarnych i różnego typu obrazów medycznych.

Specyficzne zastosowania wymagają dedykowanych algorytmów kompresji obrazu. Na przykład wprowadzenie kompresji opartej na kodowaniu falkowym do baz odcisków palców amerykańskiego Federalnego Biura Śledczego (FBI) pozwoliło na znaczną poprawę ich parametrów użytkowych (wyeliminowano w ten sposób z przechowywanych obrazów charakterystyczne dla JPEG zniekształcenia, które utrudniały porównywanie linii papilarnych) [BRIS96].

W przypadku kart graficznych renderujących grafikę trójwymiarową istotną kwestią jest szybkość ich pracy. Z uwagi na niewystarczającą pamięć znajdującą się bezpośrednio na karcie, część tekstur wczytywana jest z pamięci operacyjnej komputera, co jest procesem czasochłonnym. Rozwiązaniem jest tu kompresja tekstur zarówno przyspieszająca transmisję danych z kartą graficzną, jak i dzięki lepszemu wykorzystaniu pamięci karty ograniczająca liczbę sytuacji, w których doładowanie tekstur jest potrzebne. Dodatkowo, możliwe staje się użycie tekstur o niedopuszczalnym dotąd rozmiarze. Algorytm przedstawiony w pracy [DUMO01] zwiększył prędkość renderowania scen z dużą ilością tekstur nawet kilkunastokrotnie. Większość sprzedawanych obecnie kart graficznych posiada sprzętową implementację kompresji tekstur w standardzie DirectX/S3TC [S3I98].

Początki kompresji obrazów ruchomych to technologie dla wideotelefonii i wideokonferencji (H261, H263) [SKAR98, ss. 283–354]. Później rozwinięto je do zastosowań wymagających lepszej jakości obrazu. Najpierw Video-CD (MPEG-1), a potem DVD (MPEG-2) wprowadziły ją do powszechnego użytku. Obecnie wśród najważniejszych jej zastosowań wymienić należy telewizję cyfrową, telewizję

internetową, amatorskie i profesjonalne kamery cyfrowe, telefonię cyfrową trzeciej generacji (UMTS) [IDG99, s. 111].

1.3.3 Inne zastosowania metod kompresji danych

Druga grupa zastosowań uświadamia, że metody pierwotnie pomyślane do realizacji kompresji danych znajdują liczne zastosowania poza tą dziedziną nauki, niekiedy bardzo daleko od niej odbiegające.

W pracy *Text mining: A new frontier for lossless compression* [WITT99] wskazano na wydobywanie informacji z tekstu (*text mining*) jako ważny obszar zastosowań metod bezstratnej kompresji danych. Mianem *text mining* określa się proces analizy tekstu mającej na celu wydobywanie z niego informacji z jakichś powodów użytecznej. Użyteczne może być na przykład wyodrębnienie z tekstu obcojęzycznych wtrętów, skrótów, nazw, nazwisk, czy adresów. Daje to także możliwość klasyfikacji tekstu ze względu na wybrane kryterium (np. autora czy język).

Predykcyjna metoda PPM jest z powodzeniem stosowana w przetwarzaniu języka naturalnego [TECL97]. Wymienić tu można automatyczną identyfikację języka, poprawianie ortograficzne tekstów [TEAH98], inteligentne dzielenie wyrazów, czy też podpowiadanie i automatyczne kończenie słów.

Innym zastosowaniem PPM jest wstępne pobieranie dokumentów webowych (ang. *web prefetching*) [PALP98]. Działanie takie odbywa się w chwilach mniejszego obciążenia sieci i polega na ładowaniu do systemu *cache* dokumentów, których z dużym prawdopodobieństwem może wkrótce zażądać użytkownik. W momencie, kiedy rzeczywiście dany dokument zostanie zażądany przez użytkownika, będzie się on znajdował już w *cache*'u, co ograniczy czas oczekiwania użytkownika na dostęp do dokumentu. Rolą PPM jest odwzorowanie sposobu zachowania użytkownika – na podstawie historii dokumentów, których dotąd żądał, określa się, które dokumenty zażąda jako następne.

Zastosowanie algorytmów kompresji bezstratnej (PPM, CTW, LZ77) w mikroprocesorach do predykcji wykonywanej gałęzi instrukcji znacznie poprawiło jej skuteczność, a co za tym idzie wydajność mikroprocesorów [DEBO99].

Również algorytm Huffmana znalazł liczne zastosowania poza sferą kompresji danych. Ich przegląd zawiera tabela 1.3.

Tabela 1.3. Wybrane zastosowania algorytmu Huffmana

Zastosowanie	Autor	Rok
Konstrukcja optymalnych drzew poszukiwań	Zimmerman	1959
	Hu i Tucker	1971
	Itai	1976
Poszukiwanie przeskoków w funkcji monotonicznej jednej zmiennej	Glassey i Karp	1976
Poszukiwanie źródeł skażenia rzeki		
Poszukiwanie wycieków z rurociągu		
Łączenie list	Brent i Kung	1978
Przetwarzanie wyrażeń	Parker	1980

Źródło: opracowanie własne.

W pracy *Text Compression as a Test for Artificial Intelligence* [MAHO99] zaproponowano wykorzystanie kompresji jako testu na sztuczną inteligencję, w miejsce obarczonego licznymi wadami testu Turinga. Test polega na wbudowaniu modelu językowego testowanego automatu do kodera predykcyjnego i skompresowaniu, przy pomocy tak otrzymanego programu, tekstu w języku naturalnym. Jeżeli uzyskana efektywność predykcji (mierzona średnią bitową dla tak skompresowanego tekstu) przewyższa dolny próg efektywności predykcji wykonywanej przez ludzi (zmierzonej doświadczalnie), to testowany automat uznać można za sztuczną inteligencję. Dla tekstu w języku angielskim, dolny próg efektywności predykcji wykonywanej przez ludzi oszacowano na poziomie 1,3 bpc. Najlepszy z testowanych przez Mahoneya programów uzyskał wynik 1,87 bpc.

Rozwój technologii cyfrowych wciąż trwa. Można zatem z dużym prawdopodobieństwem przypuszczać, że w przyszłości pojawiać się będą nowe zastosowania metod kompresji danych.

2 Metody bezstratnej kompresji danych

2.1 Kodowanie statystyczne

2.1.1 Kodowanie Huffmana

Zadaniem kodowania statystycznego jest dysponując modelem probabilistycznym źródła, zapisać jego symbole w taki sposób, by osiągnąć jak najmniejszą długość wyjściowego zbioru danych. Metody kodowania statystycznego są rzadko stosowane jako samodzielne metody kompresji. Pełnią jednak wielką, niejako usługową rolę dla innych metod, wykorzystujących kodowanie statystyczne jako ostatni etap przetwarzania danych.

Najbardziej znaną metodą kodowania statystycznego jest kodowanie Huffmana [HUFF52]. Do prezentacji zarówno struktur danych, jak i algorytmów, będzie od tej pory służył pseudokod oparty na notacji języka Pascal. Zapisany w ten sposób algorytm Huffmana ma następującą postać:

```
Zbuduj_drzewo (LicznikSymboli):
// LicznikSymboli:
// Tablica zawierająca liczbę wystąpień poszczególnych symboli
// alfabetu w źródle
ListaDrzew.Wyczyść
FOR i:=0 TO Rozmiar_alfabetu_źródła-1 DO BEGIN
    NEW(NoweDrzewo)
    WITH NoweDrzewo DO BEGIN
        Symbol:=i
        Licznik:=LicznikSymboli[i]
        Syn:=NIL;    END
    ListaDrzew.Dodaj (NoweDrzewo)
END
WHILE ListaDrzew.Licznik > 1 DO BEGIN
    ListaDrzew.PosortujWgLicznikówRosnąco
    NEW(NoweDrzewo)
    WITH NoweDrzewo DO BEGIN
        Syn:=ListaDrzew.Drzewo[1]
        Córka:=ListaDrzew.Drzewo[2]
        Licznik:=Syn.Licznik+Córka.Licznik
        ListaDrzew.Usuń (Syn)
        ListaDrzew.Usuń (Córka)
    END
    ListaDrzew.Dodaj (NoweDrzewo)
END
IF ListaDrzew.Licznik > 0 THEN
    WygenerujSłowaKodowe (ListaDrzew.Drzewo[1], 0, 0)
```

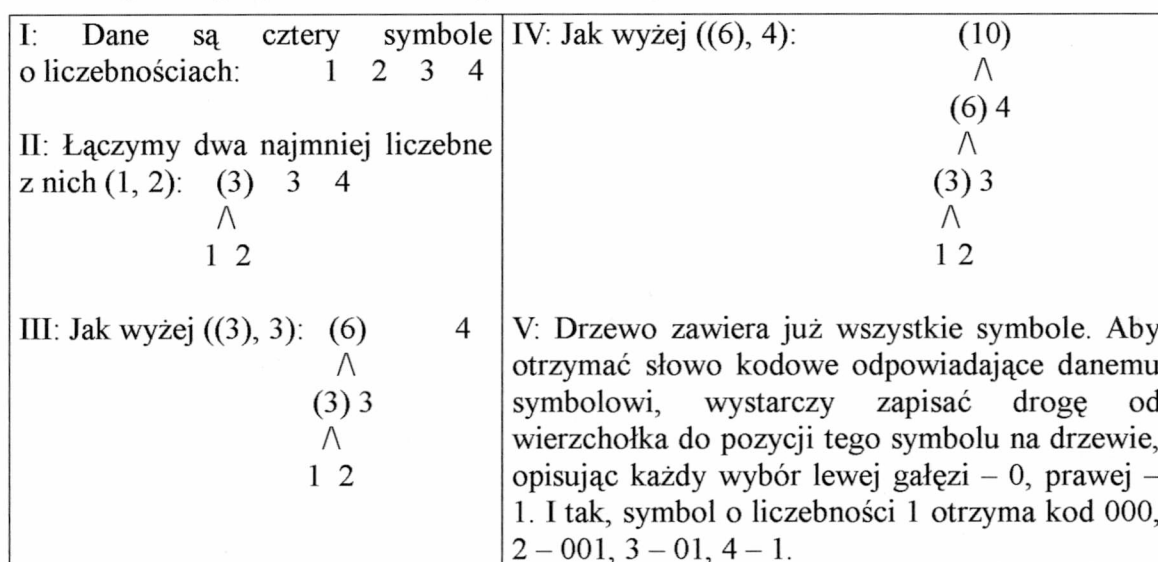
Gdzie funkcja WygenerujSłowaKodowe ma następującą postać:

```

WygenerujSłowaKodowe(Drzewo, Prefiks, Długość):
  WITH Drzewo DO
    IF Syn=NIL THEN BEGIN
      Kod[Symbol].SłowoKodowe:=Prefiks
      Kod[Symbol].DługośćSłowaKodowego:=Długość
    END ELSE BEGIN
      WygenerujSłowaKodowe(Syn, Prefiks*2, Długość+1):
      WygenerujSłowaKodowe(Córka, Prefiks*2+1, Długość+1):
    END
  DELETE(Drzewo)

```

Otrzymana w rezultacie tablica Kod, dla każdego symbolu alfabetu zawiera przyporządkowane mu słowo kodowe i jego długość. Przykład budowy drzewa kodowego w algorytmie Huffmana przedstawia rysunek 2.1.



Rys. 2.1. Przykład budowy drzewa kodowego w algorytmie Huffmana

Algorytm Huffmana wymaga znajomości prawdopodobieństw występowania poszczególnych symboli w źródle danych. Kiedy prawdopodobieństwa te nie są znane, można je przybliżyć w oparciu o statystykę tej części zbioru, która została do tej pory przetworzona. Wykorzystanie klasycznego algorytmu Huffmana do budowy nowego kodu po napotkaniu każdego kolejnego symbolu wejściowego nie nadaje się do praktycznego zastosowania, ze względu na zbyt dużą czasochłonność. Problem ten rozwiązuje dynamiczne kodowanie Huffmana (ang. *adaptive Huffman coding*), które jako pierwsi przedstawili niezależnie od siebie Faller [FALL73] i Gallagher [GALL78]. Ze względu na to, że w dedykowanym zastosowaniu algorytm ten jest obecnie wypierany przez kodowanie arytmetyczne, nie będziemy przedstawiać tu szczegółów jego realizacji.

Kod Huffmana jest optymalny, gdyż generuje najkrótsze możliwe słowa kodowe dla danego alfabetu symboli. Jego redundancja zbliża się do zera, gdy liczebności poszczególnych symboli mają się do siebie tak, jak kolejne potęgi liczby 2. W praktyce rzadko kiedy zależności pomiędzy liczebnościami poszczególnych symboli są tak

regularne. Na przykład, dla symbolu o liczebności 1 z rysunku 2.1, optymalna długość słowa kodowego wyliczona w oparciu o wzór 1.1 powinna wynosić około 3,32 bity, a nie 3 bity jak wyznaczono to algorytmem Huffmana. Słowo kodowe nie może mieć jednak niecałkowitej długości ze względu na barierę granulacji pamięci.

Barierę tę przełamuje kodowanie arytmetyczne, poprzez odstępianie od koncepcji słów kodowych przypisywanych osobno poszczególnym symbolom źródła.

2.1.2 Kodowanie arytmetyczne

W kodowaniu arytmetycznym nie przydziela się symbolom słów kodowych, ale przedziały prawdopodobieństwa. Cały zbiór wejściowy kodowany jest w postaci pojedynczej liczby (zwanej znacznikiem bądź identyfikatorem), której wartość odpowiada skumulowanemu prawdopodobieństwu wystąpienia kodowanego ciągu symboli [SAYO02, s. 83].

Wykorzystanie skumulowanej funkcji rozkładu (dystrybucyjnej) do kodowania danych, choć wzmiankowane już przez Shannona [SHAN48], stało się możliwe dopiero po rozwiązaniu problemu stałej precyzji, czego dokonali niezależnie Pasco [PASC76] i Rissanen [RISS76].

Dystrybucję $F_X(i)$ zmiennej losowej X można zdefiniować następująco:

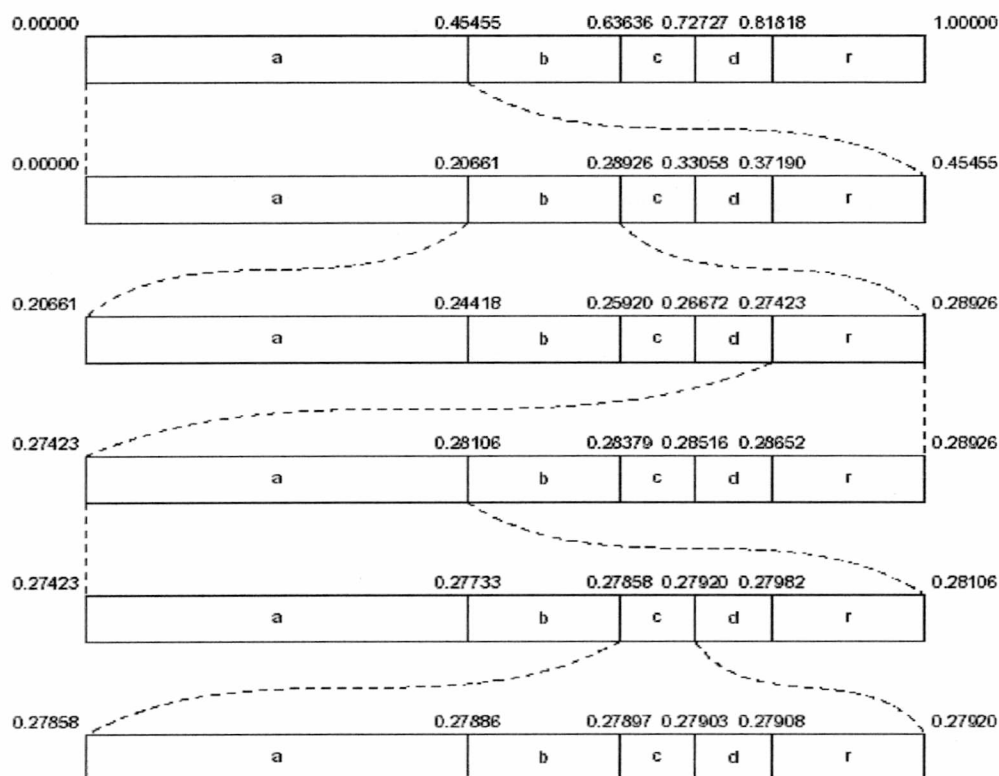
$$F_X(i) = \sum_{k=1}^i P(X = k) \quad (2.1)$$

gdzie i jest wartością zmiennej losowej X w przypadku pojawienia się w źródle symbolu a_i należącego do alfabetu symboli źródłowych A o prawdopodobieństwie $P(a_i)$:

$$\begin{aligned} X(a_i) &= i \\ a_i &\in A, \quad A = \{a_1, a_2, \dots, a_n\} \\ P(X = i) &= P(a_i) \end{aligned}$$

Funkcja ta przyjmuje wartości z zakresu $[0, 1)$. Jako że wartości tych jest nieskończenie wiele, można zatem przy jej pomocy zakodować dowolny zbiór danych.

Procedura obliczania znacznika polega na krokowym zmniejszaniu rozmiaru przedziału wraz z kodowaniem kolejnych symboli źródła. Początkowy przedział $[0, 1)$ po zakodowaniu pierwszego symbolu źródła a_i zostanie ograniczony do $[F_X(i-1), F_X(i))$. Tak otrzymany przedział zostanie proporcjonalnie zredukowany przy kodowaniu następnego symbolu. Redukcja będzie powtarzana aż do wyczerpania symboli źródła. Dowolna liczba należąca do otrzymanego w ten sposób przedziału jest identyfikatorem całego przetworzonego ciągu symboli. Przykład kodowania arytmetycznego dla ciągu symboli 'abrac' przedstawia rysunek 2.2.



Rys. 2.2. Kodowanie arytmetyczne – przykład

Źródło: [DEOR03].

Podstawowy problem kodowania arytmetycznego dotyczy sposobu w jaki zapisywany jest identyfikator. Klasyczne kodowanie arytmetyczne posługuje się kodem binarnym [SAYO02, s. 92]. Identyfikator jest w nim reprezentowany w postaci ciągu bitów oznaczających odpowiednio usunięcie górnej (0) lub dolnej (1) połowy dotychczasowego przedziału. Pozwala to na przyrostowe generowanie identyfikatora – tworzony jest on równoległe z przetwarzaniem danych źródłowych, bez potrzeby oczekiwania na ostatni symbol źródła przed rozpoczęciem zapisu.

Kod arytmetyczny jest kodem efektywnym. Jego redundancja dąży do zera, gdy długość ciągu wejściowego (S) dąży do nieskończoności. W przypadku kodu binarnego:

$$H(X) \leq L \leq H(X) + \frac{2}{S}. \quad (2.2)$$

Pominięto tu sam algorytm generujący kod binarny, w następnym punkcie przedstawiony zostanie bowiem algorytm kodowania przedziałów, o tyle bardziej istotny, że wykorzystany w proponowanej metodzie predykcyjno-substytucyjnej.

2.1.3 Kodowanie przedziałów

Klasyczne kodowanie arytmetyczne jest powolne, gdyż generuje identyfikator bit po bicie. Znaczne przyspieszenie procesu kodowania można uzyskać tworząc identyfikator z większych jednostek, na przykład bajtów.

Kodowanie przedziałów, nazywane też kodowaniem zakresów (ang. *range coding*), zaproponował jako pierwszy G. N. N. Martin [MART79]. Spopularyzował je M. Schindler, implementując w swoim programie Szip [SCHI97].

Kodowanie przedziałów jest uogólnioną postacią klasycznego kodowania arytmetycznego. Identyfikator jest w nim reprezentowany przez ciąg liczb całkowitych z zakresu $[0, n)$, z których każdej przyporządkowany jest pewien przedział prawdopodobieństwa. Łatwo zauważyć, że dla $n = 2$ otrzymujemy klasyczne kodowanie arytmetyczne, natomiast dla $n = 256$ otrzymujemy kodowanie przedziałów zorientowane bajtowo.

Poniżej przedstawiono jeden z najszybszych znanych algorytmów tego typu – koder przedziałów D. Subbotina, pozbawiony charakterystycznego dla kodowania arytmetycznego problemu przeniesienia⁸ (*carryless rangecoder*) [SUBB99].

Funkcja `RC.Zakoduj(Qsk, Qs, Qt)` koduje symbol o liczniku Qs , sumie liczników symboli poprzedzających kodowany Qsk i sumie liczników wszystkich możliwych symboli Qt . Dekoder realizowany jest poprzez dwie funkcje. Najpierw `RC.PobierzQ(Qt)` zwraca wartość z przedziału $[Qsk, Qsk+Qs)$, w oparciu o którą możliwe jest ustalenie właściwego symbolu. Następnie `RC.Odkoduj(Qsk, Qs)` ustawia stan dekodera na identyczny z odpowiadającym mu stanem kodera.

```
// Stałe
// Min_Zakres - najmniejsza dopuszczalna wielkość zakresu,
// większa lub równa największej możliwej wartości Qt
// Przesunięcie - wyznaczamy ze wzoru:
// lg(Maksymalna_dopuszczalna_wielkość_zakresu+1)-8
// Zmienne
// Zakres - aktualna wielkość zakresu,
// na początku Maksymalna_dopuszczalna_wielkość_zakresu
// Początek_zakresu - aktualna dolna granica zakresu, na początku 0
// Znacznik - (dekoder) aktualnie przetwarzany fragment znacznika
// o długości Przesunięcie+8, na początku wczytywany

RC.Zakoduj(Qsk, Qs, Qt):
Zakres:= Zakres DIV Qt
Początek_zakresu:= Początek_zakresu + Qsk * Zakres
Zakres:= Zakres * Qs
REPEAT
  IF (Początek_zakresu XOR (Początek_zakresu+Zakres))
    >= (1 SHL Przesunięcie) THEN
    IF Zakres >= Min_Zakres THEN BREAK
    ELSE Zakres:= -Początek_zakresu AND (Min_Zakres-1)
  Zapisz_bajt(Początek_zakresu SHR Przesunięcie)
  Zakres:= Zakres SHL 8
  Początek_zakresu:=Początek_zakresu SHL 8
UNTIL FALSE
```

⁸ Problem przeniesienia dotyczy sytuacji, w której zakres staje się zbyt mały, by zakodować kolejny symbol, a jednocześnie nie można jednoznacznie zakwalifikować go do pojedynczego przedziału.

```

RC.PobierzQ(Qt):
Zakres:= Zakres DIV Qt
PobierzQ:= (Znacznik-Poczatek_zakresu) DIV Zakres

RC.Odkoduj(Qsk,Qs):
Poczatek_zakresu:= Poczatek_zakresu + Qsk * Zakres
Zakres:= Zakres * Qs
REPEAT
  IF (Poczatek_zakresu XOR (Poczatek_zakresu+Zakres))
    >= Przesunięcie THEN
    IF Zakres >= Min_Zakres THEN BREAK
    ELSE Zakres:= -Poczatek_zakresu AND (Min_Zakres-1)
    Znacznik:= Znacznik SHL 8 OR Wczytaj_bajt()
    Zakres:= Zakres SHL 8
    Poczatek_zakresu:= Poczatek_zakresu SHL 8
UNTIL FALSE

```

Dzielenie liczb całkowitych powoduje niewykorzystywanie części zakresu (reprezentowanej przez resztę z dzielenia). Oznacza to, że powyższy algorytm ustępuje efektywnością klasycznemu kodowaniu arytmetycznemu. W praktyce, wydłużenie identyfikatora jest niewielkie (znacznie poniżej 0,5%).

2.2 Metody substytucyjne

2.2.1 Podstawy kompresji substytucyjnej

Generalna idea na której opierają się substytucyjne metody kompresji danych jest następująca: skoro w źródle danych pewne sekwencje symboli występują ze znacznie większym prawdopodobieństwem od innych, można powiększyć alfabet o nowe symbole, które będą reprezentować te sekwencje. W ten sposób ciąg symboli zostanie zastąpiony pojedynczym nowym symbolem, co zmniejsza liczbę symboli źródła.

Sekwencję źródła zakwalifikować można do podmiany, gdy występuje ona w:

- istniejącym słowniku typowych fraz;
- określonym kontekście semantycznym;
- wcześniejszej części zbioru wejściowego.

Przypadek pierwszy dotyczy sytuacji, gdy w momencie kompresji posiada się wiedzę *a priori* o charakterystycznych dla danego typu źródła frazach. Na przykład, mając do dyspozycji katalog biblioteki i chcąc skompresować listę zawierającą kilka pozycji z tej biblioteki, można zastąpić pełen opis bibliograficzny poszczególnych tytułów, ich numerami katalogowymi.

Metody realizujące takie podejście nazywamy statycznymi metodami słownikowymi [SAYO02, s. 124]. Są zwykle wysoce efektywne, ale dotyczą ich wszystkie mankamenty metod kompresji z modelem stałym (patrz 1.2.2), w szczególności możliwość użycia tylko dla danych, do których jest dostępny słownik.

Typowe zastosowania to kompresja tekstu w językach naturalnych (w oparciu o słownik danego języka), kodu źródłowego programów w określonym języku programowania, czy formularzy z polami zawierającymi opis słowny o kilku możliwych wartościach.

W drugim przypadku również potrzebna jest wstępna wiedza o zawartości źródła. Tym razem jednak nie jest już wymagana znajomość konkretnych fraz, które należy podmienić, wystarczy znajomość zasad budowy kompresowanego zbioru danych, pozwalająca określić początek i koniec frazy do podmiany. Zastąpienie ciągu znaków identyfikatorem nazywane jest tokenizacją napisów. Oryginalne napisy przechowywane są razem z zakodowanymi danymi, a więc metoda taka nie ma wad metod statycznych. Nadal ma natomiast wady metod dedykowanych – kompresować w ten sposób można tylko te typy danych, w których takie dające się wyodrębnić sekwencje symboli występują i wiadomo jak je zidentyfikować.

Zwykle takie podejście wykorzystywane jest w różnego rodzaju binarnych formatach wymiany danych [SWAC02a]. Najbardziej znanym przykładem takiego formatu jest binarny XML stworzony na potrzeby WAP [W3C99].

Prawdziwie uniwersalne jest dopiero podejście trzecie. Stosując je podmienić można każdą sekwencję symboli o której wiadomo, że wystąpiła we wcześniejszej części danych źródłowych. Słowa „wiadomo” użyto w poprzednim zdaniu nieprzypadkowo – większość stosowanych w praktyce metod ma niepełną wiedzę o historii źródła. Nie wystarczy więc, by pewna fraza powtarzała się – to powtarzanie musi być jeszcze zauważalne przez użyty algorytm kompresji. Niewidzialność powtórzeń dla algorytmu może być konsekwencją jednej z trzech przyczyn:

- zasady działania algorytmu – na przykład część algorytmów nie rozpoznaje powtórzeń będących fragmentem bądź połączeniem innych fraz;
- ograniczenia liczby porównywanych fraz – podyktowane zwykle chęcią skrócenia czasu działania algorytmu;
- ograniczenia rozpatrywanej historii źródła – najczęściej z powodu oszczędności pamięci. Poszukiwanie powtórzeń jest wtedy ograniczone do ostatnio przetworzonej części danych. Część tę nazywa się buforem słownikowym (ang. *sliding window*) [SAYO02, s. 126]. Część danych aktualnie oczekującą na zakodowanie nazywamy zaś buforem kodowania (ang. *look-ahead buffer*) [SAYO02, s. 128].

2.2.2 Kodowanie długości serii

Minimalna długość bufora słownikowego wynosi 1. Bufora o takiej długości używa najprostsza metoda substytucyjna – metoda kodowania długości serii (RLE od angielskiego *run-length encoding*).

W metodzie tej przesyłane na wyjście są tylko te symbole źródła, które różnią się od poprzedzających. Natomiast znajdująca się w buforze kodowania sekwencja symboli identyczna z symbolem w buforze słownikowym kodowana jest pod postacią liczby naturalnej oznaczającej długość tej sekwencji.

W przypadku typów danych, w których takie sekwencje zdarzają się często, metoda kodowania długości serii wykazuje bardzo wysoką efektywność. Najbardziej oczywistym typem danych o takich właściwościach jest prosta grafika generowana komputerowo. Właśnie na potrzeby kompresji grafiki w latach 80-tych XX wieku powstały takie wykorzystujące RLE standardy jak Amiga IFF, Apple Macintosh Packbits (używany między innymi w formacie wymiany danych TIFF), PCX, czy Microsoft RLE (używany w formacie zapisu Windows BMP) [SWAC01].

Grafika komputerowa nie wyczerpuje obszaru zastosowań RLE. Często wykorzystuje się ją jako jeden z etapów bardziej złożonych metod, w szczególności do redukcji sekwencji zer pojawiających się w pośredniej reprezentacji danych po wykonaniu kwantowania (np. JPEG [JPEG93]) lub transformacji (BWT zaimplementowanej w Bzip2) [SEWA00].

Podstawowym problemem, z którym boryka się metoda kodowania długości serii, jest co zrobić, gdy serie symboli nie występują. W klasycznej wersji algorytmu powodowałoby to, zamiast skrócenia, dwukrotne wydłużenie danych (po każdym symbolu następowało by '0' oznaczające brak powtórzeń). Rozwiązanie stanowią kody sterujące, które włączają i wyłączają tryb RLE w zależności od napotykaných danych. Kody sterujące mogą być przypisane specjalnym symbolom dodanym do alfabetu, lub umieszczone w i tak zapisywanych danych, na przykład [SWAC01]:

- w najwyższym bicie długości (np. Amiga IFF i Apple Macintosh Packbits),
- w postaci nieużywanych normalnie wartości długości (np. Microsoft RLE),
- w dwóch najwyższych bitach danych (np. PCX).

Reprezentacja kodów jako specjalnych symboli dodanych do alfabetu również może mieć wiele form. Standardowo, dodaje się do alfabetu nowy symbol, który sygnalizuje użycie RLE, dodatkowo kodując długość sekwencji korzystając z kodów uniwersalnych. Niekiedy sięga się po bardziej oryginalne rozwiązania. Na przykład do redukcji długości sekwencji zer powstałych po transformacjach BWT i MTF stosuje się zapis przy pomocy dwóch symboli zera [BAKU00]. Możliwość kombinacji

wzajemnego położenia tych dwóch symboli (0_a i 0_b) pozwala na zapis sekwencji zer o długości l w postaci sekwencji symboli 0_a i 0_b o długości $\lg(l+1)$.

Metoda kodowania długości serii sprawdza się w przypadku danych zawierających liczne ciągi takich samych symboli. W przypadku danych takich jak na przykład tekst, gdzie takie ciągi należą do rzadkości, ograniczenie historii źródła do jednego symbolu uniemożliwia uzyskanie istotnej kompresji. Kompresję takich danych umożliwiają algorytmy operujące na znacznie większym buforze słownikowym.

2.2.3 Metoda LZ77 i pochodne

W metodzie kodowania długości serii, sekwencjami są powtórzenia jednego symbolu. Znajomość tego symbolu i długości sekwencji wystarcza do odkodowania danych. Gdy bufor słownikowy jest większy, możliwe jest rozpoznanie jako wtórnej, sekwencji składającej się z różnych symboli. Potrzebny jest zatem słownik zawierający znane sekwencje – stąd nazwa: metody słownikowe [SAYO02, s. 123]. Znalezione sekwencje zamieniają one na indeks identyfikujący je w słowniku. Tabela 2.1 pokazuje przykładowy wynik kompresji metodami kodowania długości serii i słownikowymi.

Tabela 2.1. Przykład kompresji metodami substytucyjnymi

Dane źródłowe	ALAAAA MAAAA KOTAAA
RLE	1A 1L 4A 1 1M 4A 1 1K 1O 1T 1A 4A
RLE PackBits	2 AL -4 A 2 M -4 A 4 KOT -4 A
LZSS	A L A (1, 3) M (6, 4) K O T (8, 4)
LZW	A L A [AA] A M [AAA] A K O T [AAAA]

Źródło: opracowanie własne.

Dwie podstawowe metody słownikowe opublikowali w latach 1977 i 1978 Ziv i Lempel [ZILE77; ZILE78]. Od inicjałów autorów nazywa się je metodami LZ77 i LZ78. Podstawową różnicą pomiędzy nimi jest koncepcja słownika: w LZ77 słownikiem jest ostatnio przetworzona część danych (bufor słownikowy), natomiast w LZ78 słownik jest specjalną strukturą tworzoną przez koder równoległe z kodowaniem danych. Obie metody znalazły liczne zastosowania praktyczne, które przedstawiono w podrozdziale 1.3. Poniżej przedstawione zostaną ich zasady działania.

Algorytm kompresji metodą LZ77 można w uproszczeniu zapisać następująco:

```
i:=1
WHILE i <= Rozmiar_danych_wejściowych DO BEGIN
  Długość:=0
  Odległość:=0
  FOR j:=i-1 DOWNT0 i-Rozmiar_bufora_słownikowego DO BEGIN
    Tmp:=Długość_frazy_zgodnej(i,j)
    IF Tmp > Długość THEN BEGIN
      Długość:=Tmp
      Odległość:=i-j;   END
    END
  END
  Zapisz(Odległość)
```



```
Zapisz(Długość)
INC(i, Długość)
Zapisz(Symbol_źródłowy[i])
INC(i)
END
```

W procesie dekompresji, pary (*odległość*, *długość*) zastępowane są kopią ciągu pobraną z miejsca na które wskazują. Ze względu na to, że nie odbywa się wtedy wyszukiwanie, dekompresja wykonywana jest znacznie szybciej od kompresji, co oznacza, że algorytm LZ77 jest asymetryczny. Jest to cenna właściwość w przypadku danych częściej czytanych niż modyfikowanych.

Oryginalny algorytm LZ77 jest niepraktyczny ze względu na jego rozrzutność (w przypadku braku podmiany dla pojedynczego symbolu źródła zapisywane są aż trzy wartości: 0, 0, *nie podmieniony symbol*) i powolność (wyszukiwanie fraz zgodnych algorytmem typu siłowego). W 1982 roku J. A. Storer i T. G. Szymanski przedstawili algorytm LZSS, w którym zaproponowali [STSZ82]:

- wykorzystanie struktur drzewiastych do szybkiego wyszukiwania fraz zgodnych z kodowaną,
- użycie bitowego przełącznika pomiędzy trybem podmieniania a kopiowania oryginalnych symboli źródłowych.

O ile pierwsza modyfikacja czyni z LZSS bardzo szybki algorytm kompresji (szybkość można przy tym regulować poprzez zmianę wielkości bufora słownikowego), o tyle druga istotnie poprawia jego efektywność w porównaniu z LZ77.

Istnieje bardzo duża liczba modyfikacji algorytmu LZ77/LZSS. Najliczniejszą ich grupę stanowią algorytmy dwuprzebiegowe, łączące kompresję substytucyjną z kodowaniem statystycznym. Powoduje to znaczne spowolnienie kompresji, przy jednoczesnym wyraźnym wzroście efektywności.

Przykładem takiego algorytmu jest Deflate [DEUT96a]. W algorytmie tym, szybkie odnalezienie w buforze słownikowym frazy zgodnej możliwe jest dzięki przechowywaniu odwołań do każdej kolejnej kodowanej frazy na jednokierunkowej liście. Wartość funkcji mieszającej (wyliczonej w oparciu o pierwsze trzy symbole frazy) dla wszystkich elementów takiej listy jest taka sama. Dzięki tablicy zawierającej adresy list, uszeregowane według wartości funkcji mieszającej, możliwe jest szybkie przejście do właściwej listy, pod warunkiem wyliczenia wartości funkcji mieszającej dla sekwencji symboli znajdującej się w buforze kodowania.

Podmianę w algorytmie Deflate wykonuje się wyłącznie wtedy, gdy długość frazy zgodnej przekracza pewną ustaloną wartość minimalną. W przeciwnym razie, na wyjście kopiowany jest oryginalny symbol. Samą substytucję koduje się dzięki rozszerzeniu alfabetu o dodatkowe symbole odpowiadające przedziałom długości. Po

zasygnalizowaniu wykonania podmiany użyciem takiego symbolu, konieczne jest jeszcze zapisanie dokładnej (zawierającej się w znanym już przedziale) wartości długości, przedziału odległości i jej dokładnej wartości. Kolejne przedziały długości i odległości mają rosnącą wielkość. Dalszym odwołaniom odpowiadają zatem dłuższe kody, co dobrze odpowiada modelowi źródła z lokalnością referencji. W drugim przebiegu, tak zapisane dane poddaje się kodowaniu algorytmem Huffmana.

Z innych algorytmów substytucyjnych należy tu wspomnieć o obecnie najszybszym LZO [OBER03], i najefektywniejszym LZMA [PAWŁ03] (według [JUKI03]).

2.2.4 Metoda LZFG

Jedną z przyczyn obniżających efektywność kompresji metodami typu LZ77 jest tak zwana nadmiarowość wskaźników (ang. *pointer redundancy*). Bierze się ona stąd, że najbardziej typowe frazy występują w buforze słownikowym wielokrotnie. Tym samym istnieje wiele sposobów zakodowania tej samej frazy (używając różnych odległości), co w rezultacie przekłada się na niepotrzebne wydłużenie zapisu.

W roku 1989 E. R. Fiala i D. H. Greene przedstawili algorytm, który eliminuje tę słabość, znany obecnie jako LZFG [FIGR89]. W LZFG wyeliminowano problem nadmiarowości wskaźników w ten sposób, że w celu zidentyfikowania podmienianego ciągu symboli zamiast odległości do ciągu z nim zgodnego zapisuje się wskaźnik do *m*-arnego drzewa (nazywanego *trie*) zawierającego podciągi, które dotąd wystąpiły w źródle.

LZFG wykorzystuje drzewa typu PATRICIA [MORR68]. W drzewach takich każda gałąź odpowiada pojedynczemu symbolowi alfabetu, przy czym sekwencja gałęzi zawierających tylko jeden element zapisywana jest jako pojedyncza gałąź zawierająca wskaźnik do bufora, w którym znajdują się odpowiadające kolejnym gałęziom symbole. Przejście od korzenia do liścia drzewa pozwala odtworzyć oryginalną frazę. Struktury te użyte są w LZFG zarówno przy kompresji (do wyszukania zgodnego ciągu, a także do odpowiedniego zakodowania go), jak i dekompresji (do prawidłowego zdekodowania ciągu). Z powyższego faktu wynika zbliżona złożoność obliczeniowa obu procedur (symetryczność), co odróżnia tę metodę od LZ77.

2.2.5 Metoda LZ78 i LZW

W metodzie LZ78, w odróżnieniu od LZ77, słownika nie stanowi ostatnio przetworzona część danych źródłowych, ale jest on niezależnie przechowywaną strukturą danych, budowaną przez koder równoległe z przetwarzaniem danych. Zamiast pary (*odległość*, *długość*) opisującej położenie frazy zgodnej w buforze słownikowym,

do zastąpienia frazy podmienianej wykorzystuje się tu indeks określający odpowiadającą jej pozycję słownika.

W tabeli 2.2 zamieszczono porównanie opartej na LZ78 metody LZW z metodami LZSS i LZFG. Uwaga: zawarte w tej i kolejnych tabelach oceny słowne (szybkości, efektywności) są względne i odnoszą się do pozostałych metod ujętych w danej tabeli.

Tabela 2.2. Porównanie metod słownikowych

Metoda	LZSS	LZW	LZFG
Efektywność kompresji	Średnia	Niska	Wysoka
Szybkość działania	Szybka	Bardzo szybka	Wolna
Słownik	Bufor	Osobny	Osobny
Słownik zawiera wszystkie frazy	Tak	Nie	Nie
Możliwość przepełnienia słownika	Nie	Tak	Nie
Wykorzystanie lokalności referencji	Tak	Nie	Częściowe
Nadmiarowe wskaźniki	Tak	Częściowo	Nie

Źródło: opracowanie własne.

Kluczowe zagadnienia algorytmu LZ78 to:

1. W jaki sposób kodować symbole, których jeszcze w słowniku nie ma?
2. Według jakich zasad dodawać nowe frazy do słownika?
3. Co się dzieje, gdy wolne miejsce w słowniku zostaje wyczerpane?

Ad. 1. Oryginalny LZ78 rozwiązywał ten problem w ten sposób, że po każdym indeksie zapisywany był symbol kończący frazę. Dla symboli nowych używano jako poprzednika specjalnego indeksu 0 wskazującego na ciąg pusty. W roku 1984 T. Welch zaproponował modyfikację LZ78 nazwaną LZW [WELC84]. W metodzie tej słownik rozpoczyna działanie z kompletem symboli alfabetu, przez co niemożliwe staje się napotkanie symbolu, który nie posiadałby odpowiadającego mu indeksu w słowniku. Dzięki temu wystarczy kodować sam indeks. W LZW zapisuje się go na minimalnej możliwej liczbie $k+1$ bitów, dla wielkości słownika równej 2^k+2 . Ponadto, Welch zaproponował wykorzystanie tablicy mieszającej do szybkiego odnajdywania właściwych indeksów dla aktualnie kodowanej frazy.

Ad. 2. Słownik powiększa się po zakodowaniu każdej kolejnej frazy. Nowy element słownika tworzy się wydłużając właśnie zakodowaną frazę o pierwszy następujący po niej symbol. Z tego powodu, inaczej niż w LZ77, a podobnie jak w LZFG, nie wszystkie frazy, które wystąpiły dotąd w źródle mają swoją reprezentację w słowniku. Dodatkowo, wydłużanie nowych elementów słownika tylko o jeden symbol powoduje niemożność wydajnego zagospodarowania przez algorytm powtarzających się długich fraz. Są to dwie przyczyny rzutujące negatywnie na

efektywność kompresji omawianej metody. Trzecią jest użycie słów kodowych o tej samej długości do reprezentacji każdego elementu słownika.

Ad. 3. Przepełnienie słownika (bo tak określa się opisaną sytuację) powoduje niemożność dodawania do niego nowych wpisów. W metodzie LZW kodowanie może być kontynuowane w oparciu o istniejący stan słownika, lub słownik może zostać wyczyszczony (dekoder powiadamiany jest o tym poprzez wygenerowanie specjalnego słowa kodowego 'CLEAR').

2.3 Metody predykcyjne

2.3.1 Predykcja przez częściowe dopasowanie (PPM)

Istota predykcyjnych metod kompresji tkwi w przewidywaniu nieznanego dalszego ciągu źródła. Cel ich działania ująć można następująco: znając dotychczasowy fragment źródła X zbudować model M_i^k , który dla danej sekwencji symboli x_1^{i-1} pozwoli wyliczyć wartość prawdopodobieństwa $P(x_i | x_1^{i-1})$ tak, aby wartość oczekiwana wyrażenia $-\log P(x_i | x_1^{i-1})$ była jak najmniejsza.

Najbardziej rozpowszechnioną obecnie metodą predykcyjną jest opublikowana w 1984 roku przez J. G. Cleary'ego i I. H. Wittena predykcja przez częściowe dopasowanie (*Prediction by Partial Match*, PPM) [CLWI84]. Jej podstawowym pomysłem jest by tworzyć modele statystyczne jedynie dla tych kontekstów, które wystąpiły dotąd w źródle. W ten sposób zapotrzebowanie na pamięć zostaje zredukowane do wielkości pozwalającej na praktyczną realizację algorytmu.

Kodowanie symbolu rozpoczyna się zatem od znalezienia najdłuższego zgodnego z aktualnym kontekstem, w którym ten symbol już wystąpił. Pierwszym rozpatrywanym jest kontekst o długości równej rzędowi modelu. Jeżeli w tym kontekście poszukiwany symbol dotąd nie wystąpił, następuje przejście do kontekstu o długości krótszej o 1. Aby umożliwić dekodownikowi powtórzenie tej operacji, zapisuje się w tym momencie specjalny symbol zmiany kontekstu, nazywany **symbolem wyjścia** lub **symbolem ucieczki**. Jeżeli w coraz to krótszych kontekstach nadal nie będzie poszukiwanego symbolu, operacja skracania kontekstu doprowadzi ostatecznie do długości -1 i odpowiadającego jej symetrycznego modelu Bernoulliego, w którym występują wszystkie możliwe symbole.

Ustaliwszy dla symbolu x_i właściwą długość kontekstu k , przystąpić można do wyliczenia prawdopodobieństwa $P(x_i | M_i^k)$:

$$P(x_i | M_i^k) = \frac{q(x_i | M_i^k)}{T(M_i^k)} \quad (2.3)$$

gdzie: $q(x_i | M_i^k)$ oznacza licznik dotychczasowych wystąpień symbolu x_i w modelu M_i^k , zaś $T(M_i^k)$ to suma liczników wszystkich symboli, które wystąpiły dotychczas w modelu M_i^k . $T(M_i^k)$ musi uwzględniać licznik symbolu wyjścia, ale nie musi zawierać w sobie liczników tych symboli, które występują w sprawdzonych wcześniej kontekstach dłuższych od aktualnego. Ta druga poprawka nosi nazwę **zasady wyłączenia symboli** (ang. *symbol exclusions*), a symbole pomijane przy wyliczaniu sumy noszą nazwę **symboli maskowanych**.

Prawdopodobieństwa poszczególnych symboli aktualizuje się po każdym przetworzonym symbolu źródła. Istotny jest sposób realizacji aktualizacji. W oryginalnej pracy [CLWI84] zaproponowano dwie metody: PPMA i PPMB. W PPMA każde (łącznie z pierwszym) wystąpienie symbolu powoduje zwiększenie jego licznika o 1. Licznik symbolu wyjścia pozostaje zaś od początku do końca niezmienny i wynosi 1. W metodzie PPMB zwiększenie licznika symbolu o 1 wykonywane jest tylko dla wystąpień wtórnych. Pierwsze wystąpienie symbolu powoduje jedynie zwiększenie licznika symbolu wyjścia o 1. W metodach PPMA i PPMB aktualizacji poddaje się zarówno kontekst faktycznie użyty do zakodowania symbolu, jak i wszystkie zgodne konteksty od niego krótsze. Mechanizm ten nazywamy **aktualizacją bez wyłączeń** (ang. *full updates*).

W roku 1990 A. Moffat przedstawił ulepszoną w stosunku do PPMB metodę, którą nazwał PPMC [MOFF90]. Modyfikacje były dwie:

- zwiększenie licznika symbolu o 1 wykonywane jest także przy pierwszym wystąpieniu;
- aktualizacji poddaje się wyłącznie kontekst faktycznie użyty do zakodowania symbolu; mechanizm ten nazywamy **zasadą wyłączenia aktualizacji** (ang. *update exclusions*).

Tabela 2.3. Zestawienie klasycznych metod PPM

Metoda	PPMA	PPMB	PPMC	PPMD
Wartość licznika nowego symbolu	1	0	1	0,5
Zwiększenie licznika istniejącego symbolu	1	1	1	1
Zwiększenie licznika symbolu wyjścia	0	1	1	0,5
Aktualizacja krótszych kontekstów	Tak	Tak	Nie	Nie

Źródło: opracowanie własne.

W roku 1993 P. G. Howard przedstawił metodę PPMD [HOWA93]. Ulepszenie w stosunku do metody PPMC polega w niej na tym, że pierwsze wystąpienie symbolu

powoduje zwiększenie zarówno licznika symbolu jak i licznika symbolu wyjścia zaledwie o 0,5. Zestawienie wymienionych metod zawiera tabela 2.3.

Wraz z przetwarzaniem kolejnych danych, model PPM coraz lepiej dopasowuje się do źródła. Negatywną stroną tego zjawiska jest:

- nieuwzględnianie lokalności referencji (świeżo zebrane statystyki mają taką samą wagę jak te pochodzące na przykład z początku źródła);
- bardzo słaba sprawność modelu w przypadku zmiany charakterystyki źródła (gdy źródło jest heterogeniczne).

Aby zaradzić tym niedogodnościom, w nowszych implementacjach PPM stosuje się okresowe skalowanie liczników. Ma to także inne korzystne konsekwencje:

- utrzymuje liczniki symboli w przedziale $[0, q_s]$; istnienie górnej granicy wartości liczników pozwala na przechowywanie ich w jednostkach pamięci o skończonej długości;
- utrzymuje sumę liczników symboli w przedziale $[1, cq_s - q_s - c + 2]$ (gdzie c oznacza liczbę niezerowych liczników symboli w modelu); istnienie górnej granicy sumy liczników jest zwykle wymagane przez koder statystyczny.
- ogranicza minimalne możliwe prawdopodobieństwo symbolu do

$$P_{\min} = \frac{1}{cq_s - q_s - c + 2}, \quad (2.4)$$

co pozwala wyznaczyć dolną granicę efektywności modelu.

2.3.2 PPM z dziedziczeniem informacji (PPMII)

Najefektywniejszą odmianą metody PPM jest obecnie PPMII – PPM z dziedziczeniem informacji (*PPM with Information Inheritance*) D. Szkarina [SZKA02a]. Najważniejsza z wprowadzonych w niej modyfikacji to inicjowanie prawdopodobieństw nowych w danym kontekście symboli w oparciu o prawdopodobieństwa wystąpienia tych samych symboli w kontekstach krótszych (to właśnie nazywamy dziedziczeniem informacji).

Odwoływanie się do danych statystycznych zebranych w modelach krótszych kontekstów wymusza modyfikację mechanizmu aktualizacji. Rozwiązanie użyte w PPMII określić można jako pośrednie pomiędzy pełnymi aktualizacjami a wyłączeniem aktualizacji. Oprócz aktualnego kontekstu, zwiększany jest licznik symbolu również w kontekście krótszym, ale:

- dotyczy to tylko kontekstu bezpośrednio nadrzędnego (krótszego o 1 symbol);
- licznik zwiększa się tam zaledwie o 0,5;

- operację tę wykonuje się tylko dopóki licznik symbolu w aktualnym kontekście nie przekroczy 8.

Druga ważna modyfikacja PPMII w stosunku do PPMD dotyczy zasad ustalania prawdopodobieństwa symbolu wyjścia. W PPMII wykorzystuje się drugorzędną estymację ucieczki (*Secondary Escape Estimation*, SEE) oryginalnie opracowaną przez C. Blooma dla metody PPMZ [BLOO98]. Idea SEE polega na stworzeniu osobnego modelu statystycznego wyłącznie na potrzeby szacowania prawdopodobieństwa symbolu wyjścia. W metodzie PPMII sposób przeprowadzenia estymacji zależy od rodzaju kontekstu. Rozróżnia się [SZKA02a]:

- konteksty binarne, w których występuje tylko jeden symbol alfabetu oraz symbol wyjścia, gdzie parametrami modelu SEE jest sześć właściwości kontekstu;
- niebinarne konteksty nie zawierające symboli maskowanych, w których prawdopodobieństwo symbolu wyjścia wylicza się w oparciu o specjalną formułę;
- niebinarne konteksty zawierające symbole maskowane, gdzie parametrami modelu SEE jest pięć właściwości kontekstu.

D. Szkarin w swej pracy [SZKA02a] przedstawił także modyfikację swojej metody o wyższej złożoności obliczeniowej, którą nazwał „skomplikowanym PPMII” (*Complicated PPMII*, cPPMII). Jej ulepszenia dotyczą między innymi poprawy estymacji prawdopodobieństw symboli (poprzez jej późniejszą korektę w oparciu o kontekst nadrzędny), wykorzystanie mechanizmu analogicznego do SEE przy estymacji prawdopodobieństw bardziej prawdopodobnych symboli (drugorzędna estymacja prawdopodobieństw symboli – *Secondary Symbol Probability Estimation*, SSE), ujęcie w SEE dodatkowych parametrów, oraz użycie SEE także dla kontekstów nie zawierających symboli maskowanych.

2.3.3 PPM o nieograniczonej długości kontekstu (PPM*)

Klasyczny PPM ogranicza maksymalną długość kontekstu do pewnej ustalonej z góry wielkości (rzędu modelu). Choć pomaga to zachować zapotrzebowanie na czas i pamięć w rozsądnych granicach, nie do końca odpowiada właściwościom typowych źródeł danych. Rzeczywiste konteksty mogą mieć bowiem nieograniczoną długość.

W metodzie PPM* (czytaj *ppm-star*) [CLEA95] zniesiono górną granicę kontekstu, a także zaakcentowano rolę kontekstów deterministycznych (w których występuje tylko jeden symbol alfabetu; w opisie PPMII występowały one pod nazwą kontekstów binarnych). PPM* dla każdego symbolu źródła stara się znaleźć najkrótszy kontekst deterministyczny. Jeżeli takiego kontekstu nie znaleziono, przechodzi się do najdłuższego istniejącego kontekstu i kontynuuje kodowanie standardową metodą PPM.

Algorytm PPM* jest powolny i ma bardzo duże wymagania pamięciowe, jednak uzyskane przez niego wyniki są tylko nieznacznie lepsze od klasycznego PPM. Znacznie lepszą efektywność kompresji osiągnęła dopiero przedstawiona przez C. Blooma w roku 1998 metoda PPMZ [BLOO98].

Algorytm przetwarzania pojedynczego symbolu źródła jest w niej następujący:

```
Kontekst:=Najdłuższy_deterministyczny_kontekst
IF Kontekst = NIL THEN
    Kontekst:=Lokalna_estymacja_rzędu
PPM.Zakoduj(Kontekst, Symbol)
```

Istotą wkładu C. Blooma są dwie nowatorskie techniki. Pierwszą z nich jest użyta do kodowania symboli wyjścia drugorzędna estymacja ucieczki (SEE, omówiona już wcześniej przy okazji PPMII).

Drugą zaś stanowi lokalna estymacja rzędu (*Local Order Estimation*, LOE). Jest to heurystyka wykorzystywana do ustalania najwłaściwszej w danym momencie długości kontekstu, z przedziału 0–12. Wykorzystywaną w niej miarą pewności kontekstu jest prawdopodobieństwo wystąpienia najbardziej prawdopodobnego symbolu powiększone o odwrotność liczby wszystkich symboli występujących w rozpatrywanym kontekście.

2.3.4 Metoda ważenia drzew kontekstowych (CTW)

PPM, choć zdecydowanie najpopularniejsza, nie jest jedyną metodą predykcyjną stosowaną w uniwersalnej kompresji danych. Do liczących się zaliczamy także metody CTW i DMC.

Metoda ważenia drzew kontekstowych (*Context Tree Weighting*, CTW) została opracowana w roku 1993 przez Willemsa, Shtarkova i Tjalkensa [WILL95]. Choć już wcześniej istniały metody uwzględniające ważenie modeli (na przykład [RYAB84]), były one jednak niepraktyczne.

Dla każdego symbolu źródła, w modelu rzędu K , znaleźć można $K+1$ kontekstów o długości nie mniejszej niż 0. Szacowanie prawdopodobieństwa symbolu będzie najlepsze, jeżeli uwzględni się każdy z tych kontekstów. Potrzebny jest zatem specjalny mechanizm, który będzie mieszał statystyki pochodzące z kontekstów o różnej długości w celu ustalenia jednego ważnego prawdopodobieństwa, potrzebnego do zakodowania symbolu (ang. *context blending* [BUNT96]). W PPM rolę tę pełnią symbole wyjścia, które w danym kontekście reprezentują wszystkie konteksty krótsze od niego. Innym sposobem – wykorzystanym właśnie w CTW – jest ważenie poszczególnych prawdopodobieństw.

W przypadku modeli operujących na alfabecie o dużej liczbie symboli, operacja taka byłaby bardzo czasochłonna. CTW posługuje się jednak binarnym modelem źródła. Występują w nim tylko dwa symbole: '0' i '1'.

W metodzie CTW posłużono się estymatorem KT [KRTR81]. Przybliża on prawdopodobieństwo $P_e^k(q_0, q_1)$ wystąpienia q_0 zer, i q_1 jedynek w kontekście o długości k . Początkowo $P_e^k(0, 0) = 1$, a w przypadku zwiększenia q_0 :

$$P_e^k(q_0 + 1, q_1) = \frac{q_0 + \frac{1}{2}}{q_0 + q_1 + 1}, \quad (2.5)$$

Prawdopodobieństwo ważone wyznaczyć można następująco:

$$P_w^k = \begin{cases} P_e^k, & k = K \\ \frac{P_w^k + P_e^{k-1(0)} \cdot P_e^{k-1(1)}}{2}, & k < K \end{cases} \quad (2.6)$$

gdzie $k-1(0)$ oznacza kontekst o długości $k-1$, poprzedzony symbolem '0', natomiast $k-1(1)$ oznacza kontekst o długości $k-1$, poprzedzony symbolem '1'.

Stosując tę formułę rekursywnie do coraz krótszych kontekstów, przechodzi się do korzenia drzewa. Prawdopodobieństwo ważone wyliczone w tym punkcie można wykorzystać już w kodowaniu arytmetycznym.

Najprostszym sposobem efektywnego użycia CTW do kompresji tekstu (czy też innych danych niebinarnych) jest dekompozycja takiego źródła na składniki binarne [TJAL97]. W przypadku tekstu w siedmiobitowym kodzie ASCII będzie to rozbiecie każdego znaku na siedem binarnych cyfr. Do kodowania cyfry z każdej z siedmiu możliwych pozycji stosuje się osobne drzewo kontekstowe. Dla cyfry o numerze j kontekst składa się z cyfr $1..j-1$ aktualnego znaku, poprzedzonych przez siedem cyfr wcześniejszego znaku, poprzedzonych przez cyfry jeszcze wcześniejszych znaków (aż do pewnej ustalonej wartości). Z uwagi na to, że zwykle duża część drzewa jest niewykorzystywana (nie wszystkie możliwe znaki pojawiają się w tekście), w miejsce estymatora KT stosuje się inne estymatory.

2.3.5 Dynamiczna kompresja Markowa (DMC)

Dynamiczna kompresja Markowa (*Dynamic Markov Compression*, DMC) [COHO87] wykorzystuje do estymacji prawdopodobieństwa modele Markowa pierwszego rzędu. Przy przewidywaniu następnego stanu uwzględnia się w nich tylko jeden stan – obecny.

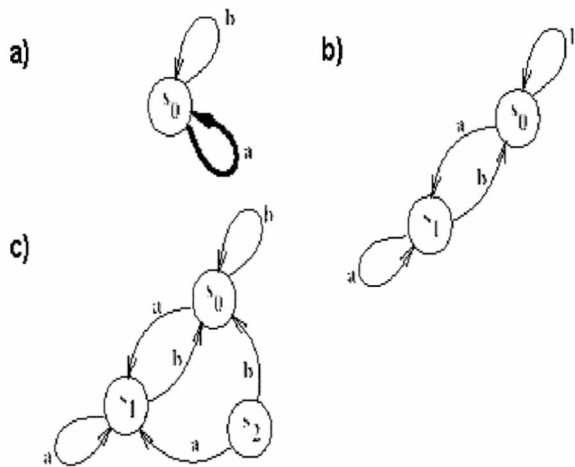
Model Markowa pierwszego rzędu można przedstawić jako skierowany graf, w którym krawędzie opisane są prawdopodobieństwami przejścia (rysunek 2.3). W DMC prawdopodobieństwa te wyliczane są w oparciu o następujące formuły:

$$P(0 | \text{obecny stan}) = (q_0 + q_c) / (q_0 + q_1 + 2q_c) \quad (2.7)$$

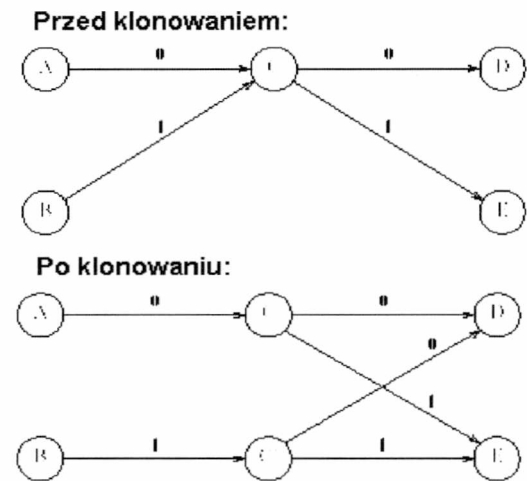
$$P(1 | \text{obecny stan}) = (q_1 + q_c) / (q_0 + q_1 + 2q_c), \quad (2.8)$$

gdzie: q_0 – liczba dotychczasowych wystąpień symbolu ‘0’ w obecnym stanie, q_1 – liczba dotychczasowych wystąpień symbolu ‘1’ w obecnym stanie, q_c – parametr modelu ($q_c > 0$). Konieczność użycia parametru q_c wynika stąd, iż nie można dopuścić by przejścia o zerowej liczbie dotychczasowych wystąpień miały zerowe prawdopodobieństwa. Im wyższa jego wartość, tym słabiej model dostosowuje się do przetwarzanego źródła.

W DMC model budowany jest wraz z przetwarzanymi danymi. Na początku zawiera on tylko jeden stan z dwoma pętlami (rysunek 2.3a) (tak jak CTW, DMC zaprojektowany został dla źródeł binarnych).



Rys. 2.3. Modele Markowa pierwszego rzędu
Źródło: [BUNT96], opracowanie własne.



Rys. 2.4. Proces klonowania stanów modelu Markowa
Źródło: [COHO87], opracowanie własne.

W momencie kodowania danej następuje przejście do następnego stanu. Jeżeli do stanu tego prowadzą co najmniej dwie gałęzie, może on zostać sklonowany. Operacja ta polega na skopiowaniu stanu następnego (C) do nowego stanu (C'), i przekierowania przejścia ze stanu dotychczasowego do stanu C na stan C' (rysunek 2.4).

W trakcie przetwarzania danych klonowanie stanów powoduje wzrost wykorzystywanej przez algorytm pamięci. W momencie, kiedy dojdzie do zapełnienia całej przydzielonej pamięci następuje wyczyszczenie modelu.

Zmniejszenie liczby stanów tworzonych w modelu DMC możliwe jest po zmianie alfabetu źródła na większy. Jako że dla każdego stanu istnieje tyle możliwości przejścia, ile jest elementów alfabetu, pociąga to za sobą drastyczny wzrost zapotrzebowania na pamięć. Sposobem na zaradzenie temu problemowi jest technika leniwego klonowania

(ang. *lazy cloning*), w której do sklonowanego stanu kopiuje się przejścia wychodzące dopiero wtedy, gdy okazują się być tam potrzebne. Implementacje tej techniki autorstwa J. Teuholi i T. Raity (GDMC) [TERA93] i – niezależnie – S. Bunton (LazyDMC) [BUNT94], pozwoliły przy ograniczonym wykorzystaniu pamięci na zachowanie wysokiej efektywności kompresji.

2.3.6 Algorytm przełączający Volf i Willemsa (VW98)

Metoda VW98 [VOWI98a] posługuje się dwoma niezależnymi modelami źródła. Są one traktowane jako „czarne skrzynki”: dostarczany jest im kolejny symbol źródła, a odbierane prawdopodobieństwo jego wystąpienia. Kodowanie jest efektywne, gdy z dwóch otrzymanych prawdopodobieństw wybierze się wyższe. Dekoder nie jest jednak wtedy w stanie ustalić, którego modelu powinien użyć do odtworzenia oryginalnego symbolu. Przed każdym symbolem należy zatem zakodować numer użytego modelu, lub – jak to się ma w VW98 – flagę przejścia ('0', gdy model nie zmienił się od poprzedniego symbolu, '1', gdy zmienił się).

Koszt zakodowania flagi przejścia powinien być uwzględniony przy dokonywaniu wyboru modelu. Jeżeli model 'A' jest dla danego symbolu wydajniejszy od modelu 'B', ale koszt zakodowania przejścia do modelu 'A' przekracza zysk z użycia go, należy pozostać przy modelu 'B'.

Volf i Willems zaproponowali by zamiast próbować znaleźć algorytm przydzielający sekwencje symboli źródła poszczególnym modelom w optymalny sposób, posłużyć się wagą uwzględniającą wszystkie możliwe podziały. W algorytmie przełączania VW98 posłużono się w tym celu estymatorem KT [KRTR81]. Algorytm przełączania ma złożoność obliczeniową $O(n^2)$. Volf i Willems przedstawili także algorytm o zredukowanej złożoności obliczeniowej, o liczbie zapamiętywanych stanów danej parametrem, dzięki któremu możliwa jest pewna regulacja pomiędzy szybkością działania, a efektywnością kompresji.

Kolejną ważną sprawą jest dobór modeli, między którymi ma następować przełączanie. W swojej oryginalnej publikacji Volf i Willems sprawdzili kombinacje następujących modeli: CTW z PPMD rzędu 5 oraz CTW z LZ77 (każdą metodę substytucyjną można przekształcić do postaci modelu predykcyjnego, co udowodniono już w pracy [LANG84]).

Uzyskane wyniki przemawiały na korzyść kombinacji CTW z LZ77, co logicznie wyjaśniono tym, że lepiej do łączenia nadają się odmienne modele źródła (CTW / LZ77), niż opierające się na podobnych zasadach (CTW / PPMD). W kolejnej publikacji [VOWI98b] przedstawiono rezultaty kombinacji CTW z PPM* i jego dwoma

autorskimi modyfikacjami (mającymi za zadanie upodobnić jego model do LZ77). Najlepsze wyniki uzyskano jednak dla połączenia CTW z oryginalnym PPM*. Kompresja taką metodą jest efektywna, ale dalece zbyt powolna dla praktycznych zastosowań.

2.4 Metody transformacyjne

W odróżnieniu od poprzednio przedstawionych metod, w metodach transformacyjnych, modelowanie i kodowanie musi być poprzedzone wykonaniem na zbiorze danych wejściowych dwóch następujących po sobie transformacji.

Pierwszą z nich jest blokowa transformacja sortująca Burrowsa-Wheelera (BWT) [BUWH94]. Jej celem jest przekształcenie występującej w kompresowanym zbiorze danych korelacji globalnej wyższego rzędu (dla całego zbioru, w określonym kontekście dominuje pewien symbol) na łatwiejszą do wymodelowania korelację lokalną rzędu 0 (dla wycinka zbioru, niezależnie od kontekstu dominuje pewien symbol).

Transformację BWT rozpoczyna utworzenie macierzy rotacji (rysunek 2.5). Jej pierwszy wiersz stanowi oryginalny zbiór wejściowy. Kolejne wiersze zawierają te same dane, przesunięte jednak o rosnącą liczbę pozycji w lewo. Drugi wiersz zaczyna się zatem od drugiego symbolu oryginalnego zbioru, a kończy na pierwszym symbolu.

Wiersze tak otrzymanej macierzy sortuje się w porządku leksykograficznym. Rezultatem transformacji jest ostatnia kolumna posortowanej macierzy. Do wykonania transformacji odwrotnej potrzebny jest jeszcze numer wiersza, w którym po sortowaniu znalazł się oryginalny zbiór danych.

1. Sekwencja wejściowa: ABAC	
2. Macierz rotacji	3. Macierz posortowana
ABAC	ABAC
BACA	ACAB
ACAB	BACA
CABA	CABA
4. Sekwencja wyjściowa (ostatnia kolumna macierzy posortowanej): CBAA	

Rys. 2.5. Transformacja sortująca Burrowsa-Wheelera.

Druga z transformacji ma na celu zdekorelowanie tak otrzymanych danych. Oryginalnie zaproponowanym przez Burrowsa i Wheelera (i dotąd używanym, na przykład w programie Bzip2 [SEWA00]) było kodowanie MTF (*move-to-front*). Polega ono na zastąpieniu kodowanego symbolu jego indeksem w dynamicznym stosie typu LIFO, i przesunięciu tego symbolu na szczyt stosu. Później zaproponowano rozmaite

modyfikacje i zamienniki tego kodowania. Ich dyskusję wraz z opisem najlepszego obecnie algorytmu WFC znaleźć można w pracy [DEOR03].

W efekcie przekształcenia dekorrelującego, zbiór wynikowy zawiera nieproporcjonalnie dużą liczbę zer i kolejnych małych liczb naturalnych. Aby uzyskać wysoki stopień jego kompresji, wystarcza prosty model statystyczny rzędu 0, w połączeniu z koderem statystycznym.

Szybszym odpowiednikiem BWT jest transformacja sortująca ST (*Sort Transform*) [SCHI97]. W odróżnieniu od BWT, w ST rotacje sortuje się uwzględniając tylko ograniczoną liczbę ich początkowych znaków. Jeżeli BWT jest transformacyjnym odpowiednikiem metod predykcyjnych o nieskończonym rzędzie modelu (np. PPM*), o tyle ST odpowiada metodom predykcyjnym o skończonym rzędzie modelu (np. PPM). Porównanie ST i BWT zawiera tabela 2.4.

Tabela 2.4. Porównanie metod transformacyjnych

Transformacja	BWT	ST
Długość kontekstu	Nieograniczona	Ograniczona
Szybkość kompresji	Wolna	Szybka
Szybkość dekompresji	Szybka	Wolna
Lokalność referencji	Nieuwzględniona	Częściowo uwzględniona

Źródło: opracowanie własne.

2.5 Metody hybrydowe

2.5.1 Metody substytucyjne ze słownikiem kontekstowym

Zarówno metody substytucyjne jak i predykcyjne działają dzięki wykorzystaniu korelacji występującej pomiędzy symbolami źródła. Różnią się jednak przyjętym podejściem, wykorzystaniem specyficznych właściwości źródła, jak również szybkością działania i osiąganą efektywnością kompresji.

Istniejące różnice motywowały badaczy do prób łączenia obu rodzajów metod. Pierwszym zaproponowanym pomysłem było podzielenie słownika na wiele osobnych części, z których każdą przyporządkowano określone kontekstowi.

Prekursorem był R. Williams i jego algorytm LZRW4 [WILL91]. Dzielił on słownik typu LZ77 na słowniki kontekstowe. Każdy z nich zawierał wskaźniki do 32 ostatnich fraz, dla których wartość funkcji mieszającej wyliczona dla dwóch symboli poprzedzających daną frazę jest taka sama.

Kolejne dwa algorytmy przedstawiono w roku 1994. Propozycja Gutmanna i Bella [GUBE94] koncentrowała się na szybkości działania (mniejszy słownik łatwiej jest przeszukać), natomiast algorytm CSD [NAKA94], oparty na LZW pozwalał także na osiągnięcie pewnej poprawy efektywności kompresji (wynikającej z wykorzystania

korelacji na granicy fraz). W obu przypadkach, kontekst dla słownika stanowił pojedynczy symbol poprzedzający frazę.

Najbardziej zaawansowaną metodą tego typu jest DPM (*Dictionary by Partial Matching*, Słownik (otrzymany) poprzez częściowe dopasowanie) [HOAN95], którą można określić jako substytucyjny odpowiednik metody PPM.

Hoang zaproponował oryginalnie trzy implementacje swojego algorytmu: wykorzystującą jako algorytm bazowy LZ77 (LZ77-PM), opartą na LZ78 (LZ78-PM), oraz opartą na LZFG (LZFG-PM). W porównaniu do algorytmu bazowego zaobserwowano następującą poprawę efektywności kompresji: LZ77-PM – 2%, LZ78-PM – 11%, LZFG-PM – 4,3%.

Algorytm LZ77-PM dla każdego kodowanego symbolu przedstawić można następująco:

```
// i - pozycja w źródle danych
// K - rząd modelu (najdłuższy kontekst dla którego istnieje słownik)
k:=K
REPEAT
  Kontekst:=Fragment_źródła(i-k,i-1)
  IF Słownik[Kontekst].LiczbaElementów > 0 THEN BREAK
  Zaktualizuj(Słownik[Kontekst])
  DEC(k)
UNTIL k = 0
Długość:=0
REPEAT
  Kontekst:=Fragment_źródła(i-k,i-1)
  FOR f:=1 TO Słownik[Kontekst].LiczbaElementów DO BEGIN
    Tmp:=Długość_frazy_zgodnej(i, Słownik[Kontekst].Element[f])
    IF Tmp > Długość THEN BEGIN
      Długość:=Tmp
      Indeks:=f;      END
  END
  IF Długość > 0 THEN GOTO Koduj_frazę
  Zapisz(Symbol_ucieczki_z_kontekstu)
  Zaktualizuj(Słownik[Kontekst])
  DEC(k)
UNTIL k < 0
// Koduj symbol
Zapisz_symbol(Źródło[i])
INC(i)

Koduj_frazę:
  Zapisz_indeks(f)
  Zapisz(Długość)
  WHILE k > 0 DO BEGIN
    Kontekst:=Fragment_źródła(i-k,i-1)
    Zaktualizuj(Słownik[Kontekst])
    DEC(k);      END
  INC(i, Długość) // opcja - zaktualizuj słowniki wewnątrz frazy
```

2.5.2 Metoda LZP

Istotnym krokiem do przodu była dopiero przedstawiona w roku 1996 przez C. Blooma metoda LZP (LZ+Predykcja) [BLOO96]. Jej algorytm dla każdego kodowanego symbolu jest następujący:

```
// i - pozycja w źródle danych
c:=Źródło[i-1]*$10000+Źródło[i-2]*$100+Źródło[i-3]
h:=((c SHR 11) XOR c) AND $FFF // Formuła dotyczy wersji LZP1 i 2
l:=Długość_frazy_zgodnej(i,Słownik[h])
Słownik[h]:=i
IF l > 0 THEN BEGIN
    Zapisz_flagę(PODMIANA)
    Zapisz_długość(l)
    INC(i, l)
END ELSE BEGIN
    Zapisz_flagę(LITERAŁ)
    Zapisz_symbol(Źródło[i])
    INC(i)
END
```

Oryginalnie Bloom przedstawił cztery warianty opisanej metody, oznaczone odpowiednio LZP1 do LZP4. W metodach LZP1 i 2 wykorzystuje się tryb bajtowy kontekst, który redukuje się do 12 bitów (patrz algorytm), co daje łącznie 4096 jednoelementowych słowników.

Jedyna różnica pomiędzy metodami LZP1 i 2 tkwi w sposobie zapisu danych wyjściowych. LZP1 zapisuje nie podmienione symbole (literały) na 8 bitach, flagi (braku) podmiany w postaci 1-2 bitowych sekwencji, długości podmiany zapisywane są zaś przy pomocy kodu przyrostowego (schemat: 2-3-5-(8) bitów). Natomiast metoda LZP2 wykorzystuje do kodowania klasyczny algorytm Huffmana.

Wariant LZP3 posługuje się czterobajtowym kontekstem reprezentowanym 16-bitową wartością funkcji mieszającej:

```
h:=((c SHR 15) XOR c) AND $FFFF // Formuła dotyczy wersji LZP3
```

LZP3 wykorzystuje technikę potwierdzania kontekstu: jeżeli dana wartość funkcji mieszającej wystąpiła już wcześniej, ale zachowany kontekst różni się od aktualnego, podmiany nie wykonuje się. Technika ta podwaja wielkość tablicy indeksów, ale jest ważna dla dobrego działania kontekstu rzędu 4. Ponadto, LZP3 wprowadza (przywołującą na myśl PPM) zasadę skracania kontekstu: jeżeli kontekst o długości k dotąd nie wystąpił, następuje przejście do kontekstu o długości $k-1$. LZP3 stosuje również bardziej wyrafinowane metody kodowania: flagi i długości kodowane są metodą Huffmana rzędu 1 (kontekstem jest rząd modelu, w którym nastąpiła podmiana), natomiast literały metodą Huffmana rzędu 1 opisaną w [BLOO96].

LZP4 wykorzystuje pięciobajtowy kontekst, bez redukcji funkcją mieszającą. Konteksty pamiętane są w strukturze przypominającej używane w PPM (przy czym

LZP4 nie wymaga przechowywania jakichkolwiek danych poza adresem kontekstu). Do zapisu wykorzystuje się kodowanie arytmetyczne.

Odmiany LZP1 i LZP2 przeznaczone są do bardzo szybkiej kompresji, dorównując efektywnością metodzie LZSS, LZP3 zbliża się efektywnością do metod wykorzystujących połączenie LZ77 i kodowania Huffmana, podczas gdy LZP4 zbliża się do efektywności PPMD.

2.5.3 Inne algorytmy hybrydowe

Przez kilka ostatnich lat, jednym z najbardziej efektywnych programów kompresujących pozostawał RK M. Taylora [GILC02]. W podstawowym trybie kompresji wykorzystuje on metodę ROLZ (*Reduced Offset Lempel-Ziv*, kompresja Lempela-Ziva o Ograniczonej Odległości). Łączy ona elementy LZP z rozwiązaniami znanymi z metod przedstawionych w punkcie 2.5.1. Z jednej strony literały kodowane są w niej przy pomocy modelu PPM, z drugiej zaś kontekstem dla słowników jest pojedynczy symbol, a zawierają one więcej niż jeden element.

Obecnie, wśród najbardziej efektywnych programów kompresujących [JUKI03], plasuje się program UHARC [HERK02]. W trybie najlepszej kompresji, wykorzystuje on algorytm będący rozwinięciem LZP4:

```
i:=1
WHILE i <= Rozmiar_źródła DO BEGIN
  Długość:=0
  Kontekst:=Funkcja_mieszająca(Fragment_źródła(i-12,i-1))
  IF Słownik[Kontekst].LiczbaElementów > 0 THEN BEGIN
    Indeks:=1
    IF Słownik[Kontekst].LiczbaElementów > 1 THEN
      IF Wsteczna_zgodność(i,Słownik[Kontekst].Element[1])
        < Wsteczna_zgodność(i,Słownik[Kontekst].Element[2])
      THEN Indeks:=2 // wsteczna zgodność - liczba takich samych
                      // symboli przed początkiem frazy
    Długość:=Długość_frazy_zgodnej(i,Słownik[Kontekst].Element[Indeks])
  END ELSE BEGIN
    Kontekst:=Funkcja_mieszająca(Fragment_źródła(i-4,i-1))
    IF Słownik[Kontekst].LiczbaElementów <> 0 THEN
      Długość:=Długość_frazy_zgodnej(i,Słownik[Kontekst].Element[1])
    END
  IF (Długość < Długość_minimalna) THEN BEGIN
    PPM.Zakoduj_symbol_i_zaktualizuj_model(i)
    INC(i, 1)
    Zaktualizuj_długość_minimalną()
  END
  ELSE BEGIN
    LZP.Zakoduj_podmianę(Długość)
    INC(i,Długość)
  END
  LZP.Zaktualizuj_słowniki()
END
```

Model predykcyjny oparty jest na wariancie PPM, rzędu 4 (z pominięciem modeli rzędu 3) i modelowaniem kontekstów zawierających wtrącenia.

Algorytmy Taylora i Herklotza osiągają bardzo wysoką efektywność kompresji, ale są powolne. Odmiennego rodzaju jest algorytm użyty w programie GRZip [GREB03]. Jest to algorytm dwuprzebiegowy. Pierwszy, substytucyjny przebieg, kompresuje dane metodą LZP o długości kontekstów 4 i 5. W drugim przebiegu na tak przygotowanych danych wykonuje się transformację BWT [BUWH94].

Celem użycia LZP jest przyspieszenie kompresji. Po pierwsze, skraca zbiór wejściowy, po drugie znacznie redukuje średnią długość frazy zgodnej, co przekłada się na duże przyspieszenie najbardziej czasochłonnego w BWT etapu sortowania danych.

GRZip oferuje także szybki tryb kompresji, w którym w miejsce BWT stosowana jest transformacja ST [SCHI97]. Przy sortowaniu uwzględnia ona tylko 4 pierwsze znaki, dlatego w tym przypadku wcześniejsze użycie LZP, prócz prędkości działania, przekłada się na wzrost efektywności kompresji. Tę wersję algorytmu Grebnowa porównać można do LZP4, w którym miejsce PPM zajęła ST.

Szczegółowe zestawienia porównawcze metod hybrydowych umieszczono w punkcie 3.5.4, przy okazji omawiania różnic pomiędzy nimi a nową metodą.

2.6 Zestawienie metod kompresji bezstratnej

Znajdujące się poniżej tabele pomocne będą w porównaniu opisanych w tym rozdziale metod. Tabela 2.5 zawiera najniższą średnią bitową (*BR*) uzyskaną przez zastosowanie każdej z wymienionych metod na zestawie plików testowych *Calgary Corpus* [WITT87]. Zamieszczone dane pochodzą albo z oryginalnych publikacji przedstawiających wymienione metody, albo z innych publikacji, zawierających zestawienia porównawcze dla różnych metod.

Ze względu na to, że zbudowanie w analogiczny sposób zestawienia czasów działania jest niemożliwe, gdyż algorytmy były testowane przez swoich autorów na sprzęcie różnej mocy obliczeniowej, a niedostępność programów, w szczególności dla starszych metod, uniemożliwia wykonanie testów dla wszystkich metod, tabela 2.6 zawiera tylko kilka wybranych pozycji. Testy przeprowadzono na komputerze PentiumII-233 (292 MHz) z 64 MB RAM. Testowano *Calgary Corpus*.

Jak można zauważyć, poszczególne metody kompresji różnią się nie tylko uzyskiwaną efektywnością kompresji, ale także – i to w dużo większym stopniu – zapotrzebowaniem na zasoby systemowe, w szczególności czas, potrzebny do wykonania kompresji. W następnym rozdziale przedstawiona zostanie nowa metoda kompresji, pomyślana tak, by osiągnąć możliwie jak najlepsze połączenie efektywności kompresji i czasu jej wykonywania.

Tabela 2.5. Zestawienie efektywności wybranych metod kompresji bezstratnej

Algorytm	Rok	BR	Algorytm	Rok	BR
Huffman	1952	4,33	CTW	1995	2,19
LZ77	1977	3,94	DPM (LZFG-PM)	1995	2,80
LZW	1984	3,58	Deflate (LZ77+Huffman)	1995	2,69
DMC	1986	2,74	LZP-4	1996	2,29
LZFG	1989	3,02	PPM* (Bunton)	1997	2,18
PPMC (rząd 3)	1990	2,48	ST (Schindler, rząd 4)	1997	2,54
PPMD (rząd 5)	1993	2,29	VW98 (CTW+PPM*)	1998	2,12
PPM*	1993	2,34	PPMZ	1998	2,14
GDMC	1993	2,52	BWT (Deorowicz)	2000	2,25
CSD	1994	3,53	PPMII (rząd 8)	2002	2,12
BWT (+MTF)	1994	2,43	cPPMII (rząd 64)	2002	2,04

Źródło: [BLOO96], [CLEA95], [DEOR02], [HOAN97], [SCHI97], [SZKA02a], [TERA93],
opracowanie własne.

Tabela 2.6. Szybkość działania i użycie pamięci wybranych metod kompresji

Metoda	Implementacja	Czas kompresji (s)	Użyta pamięć (MB)
BWT	Bzip2	5,87	6
cPPMII (rząd 10)	PPMonstr	17,57	13,8
Deflate (-9)	Info-Zip	5,93	0,5
PPMD (rząd 5)	Szkarin	4,67	3,5
PPMII (rząd 5)	PPMd	5,21	3,5
PPMII (rząd 10)	PPMd	7,25	7,25
PPMZ	PPMZ2	>60	>100

Źródło: [SZKA02a].

3 Metoda predykcijno-substytucyjna

3.1 Wprowadzenie do problematyki predykcijno-substytucyjnej

3.1.1 Podstawowa terminologia

Tematyka predykcijno-substytucyjnej kompresji danych obejmuje kilka kluczowych pojęć, które wielokrotnie występować będą w opisie podejmowanych w tym rozdziale zagadnień. Nie wszystkie z tych pojęć zostały wcześniej zdefiniowane; niektóre zaś nabierają w kontekście predykcijno-substytucyjnej kompresji danych specyficznego znaczenia. Zaczniemy zatem od definicji pojęć, które będą potrzebne do opisu metody predykcijno-substytucyjnej.

W dalszej części pracy rozważane będzie kodowanie źródła danych na dwa sposoby: predykcijny i substytucyjny. Istnieją różne algorytmy realizujące kompresję, które można zaliczyć do każdej z tych grup (patrz rozdział 2). Aby nadać rozważaniom ukierunkowanym na opracowanie nowej metody konkretność, konieczne było ograniczenie się do pojedynczych reprezentantów tych dwóch grup. Wybrano w tym celu algorytmy rokujące największe nadzieje: nieskomplikowane, posiadające dobre parametry użytkowe.

Za punkt wyjścia dla metody predykcyjnej przyjęto odmianę C algorytmu PPM [MOFF90]. Odniesienia do **metody predykcyjnej** (dla uproszczenia oznaczana również skrótem **PPM**) będą zatem odwołaniami do tego właśnie algorytmu, dopóki nie zostaną doń w jawny sposób wprowadzone pewne modyfikacje, co nastąpi w dalszej części tego rozdziału.

Metoda substytucyjna (dla uproszczenia oznaczana również skrótem **LZ**) na początek oznaczać będzie algorytm typu LZ77 połączony z kodowaniem statystycznym rzędu 0 dla symboli nie podlegających podmianie. Można to traktować jako pewne przybliżenie powszechnie stosowanego algorytmu Deflate [DEUT96a]. Pełna charakterystyka wyjściowej postaci tej metody nie jest w niczym potrzebna do ostatecznego określenia jej formy docelowej, stąd ograniczono się do sprecyzowania tylko tych jej szczegółów, które będą istotne dla prowadzonych rozważań, i dopiero wtedy, gdy okazało się to konieczne.

Należy zatem pamiętać, że w tym rozdziale wszystkie sformułowania dotyczące metody substytucyjnej (lub krótko LZ) dotyczą powyższego algorytmu, a nie metod

substytucyjnych w ogóle. Podobnie, wszystkie sformułowania dotyczące metody predykcyjnej (lub krótko PPM) dotyczą opisanego wcześniej algorytmu, a nie metod predykcyjnych w ogóle. Jakiegokolwiek odstępstwa od tej reguły będą sygnalizowane.

Niech X oznacza źródło danych, a x_i i -tą daną elementarną wygenerowaną przez to źródło. Ciąg danych elementarnych zaczynający się od x_i , a kończący na x_{i+l-1} , gdzie $l \geq 1$ nazywać będziemy **sekwencją** lub **frazą** o **długości** l i oznaczać x_i^{i+l-1} .

Jeżeli fraza nie zawiera jakichkolwiek symboli, to nazywamy ją **frazą pustą**, a jej długość $l = 0$.

Sekwencję poprzedzającą daną elementarną x_i nazywać będziemy jej **kontekstem** x_{i-k}^{i-1} o **długości** k , $k \geq 1$. Dla kontekstu x_{i-k}^{i-1} obserwujemy:

- **kontekst poprzedzający** x_{i-k}^{i-2} ,
- **kontekst następujący** x_{i-k+1}^i ,
- **kontekst nadrzędny** x_{i-k+1}^{i-1} ,
- **kontekst podrzędny** x_{i-k}^i .

Frazą zgodną dla pozycji źródła i nazwiemy taką frazę x_j^{j+l-1} , gdzie $j < i$, że $x_i^{i+l-1} = x_j^{j+l-1}$. Symbol x_{j+l} nazywamy **symbolem kończącym** frazę.

Różnicę $d = i - j$ nazywamy **odległością** od frazy zgodnej.

Jeżeli dla danej frazy x_i^{i+l-1} istnieje fraza zgodna x_j^{j+l-1} do której odległość d , $d \geq 1$, jest mniejsza od długości l , to frazę x_i^{i+l-1} nazywamy **frazą okresową** o okresie d .

Najdłuższą frazą zgodną dla pozycji źródła i nazwiemy taką frazę zgodną x_j^{j+l-1} , gdzie $j < i$, że nie da się znaleźć w słowniku innej frazy zgodnej x_m^{m+n-1} , gdzie $m < i$, takiej że $n > l$. Dla każdego $i > 0$ istnieje co najmniej jedna najdłuższa fraza zgodna o długości $l \geq 0$.

Jeżeli x_j^{j+l-1} jest najdłuższą frazą zgodną dla pozycji źródła i , to o kończącym tę frazę symbolu x_{j+l} wiemy że $x_{i+l} \neq x_{j+l}$.

Słownikiem D_i dla pozycji źródła i nazywamy zbiór wszystkich pozycji źródła $j \geq 0$, $j < i$, rozpatrywanych przy wyszukiwaniu najdłuższej frazy zgodnej dla pozycji źródła i . Dla każdego $i > 0$, $|D_i| \leq i$. Jeżeli słownik D_i^k , gdzie $k > 0$, zawiera tylko takie pozycje źródła j , dla których $x_{i-k}^{i-1} = x_{j-k}^{j-1}$, to nazywamy go **słownikiem kontekstowym rzędu** k .

Modelem statystycznym M_i^k rzędu k dla kontekstu x_{i-k}^{i-1} nazywamy zbiór liczników $q(a|M_i^k)$, $a \in A$, odpowiadających liczbie wystąpień na pozycjach źródła $j < i$ takich fraz x_{j-k}^j , że $x_{j-k}^j = x_{i-k}^{i-1}a$, wraz z licznikiem $q(Esc|M_i^k)$ odpowiadającym liczbie niezerowych liczników $q(a|M_i^k)$.

Przez $c(M_i^k)$ oznaczamy **liczbę niezerowych liczników** w modelu M_i^k .

Przez $T(M_i^k)$ oznaczmy **sumę liczników** modelu M_i^k :

$$T(M_i^k) = q(Esc | M_i^k) + \sum_{m=1}^{|A|} q(a_m | M_i^k). \quad (3.1)$$

Modelem PPM rzędu K źródła danych X nazywamy zbiór wszystkich modeli statystycznych tego źródła, których rząd $k \leq K$. W modelu PPM licznik $q(Esc)$ modelu statystycznego M_i^k określa prawdopodobieństwo przejścia do modelu statystycznego M_i^{k-1} , w sytuacji gdy $q(x_i | M_i^k) = 0$. Model PPM obejmuje także model statystyczny M^{-1} , w którym, dla każdego $a \in A$, $q(a) = 1$ i $q(Esc) = 0$.

W modelu PPM stosuje się **zasadę wyłączania symboli**, która mówi że przy redukcji rzędu modelu z $k+1$ do k , do sumy liczników modelu M_i^k nie włączamy liczników tych symboli a , dla których $q(a|M_i^{k+1}) > 0$. Sumę liczników uwzględniającą tę zasadę obliczymy następująco:

$$T(M_i^k | M_i^{k+1}) = q(Esc | M_i^k) + \sum_{m=1}^{|A|} \begin{cases} q(a_m | M_i^k), & q(a_m | M_i^{k+1}) = 0 \\ 0, & q(a_m | M_i^{k+1}) > 0 \end{cases} \quad (3.2)$$

Przyjmujemy, że dla każdego $a \in A$, $q(a | M_i^{K+1}) = 0$.

Prawdopodobieństwo symbolu x_i estymowane przy pomocy modelu PPM rzędu K dane jest wzorem

$$P_{est}(x_i) = \frac{q(a | M_i^{k_i})}{T(M_i^{k_i} | M_i^{k_i+1})} \prod_{k=k_i+1}^K \frac{q(Esc | M_i^k)}{T(M_i^k | M_i^{k+1})}, \quad (3.3)$$

gdzie k_i oznacza największe $k \leq K$ dla którego istnieje takie M_i^k , że $q(x_i | M_i^k) > 0$.

Model PPM nazywamy **spójnym**, jeżeli dla każdego należącego do niego modelu M_i^k , gdzie $k \geq 0$, nie istnieje takie x_i dla którego $q(x_i | M_i^k) = 0$ i $q(x_i | M_i^{k+1}) \neq 0$.

Długość zakodowanego w oparciu o model PPM zbioru danych X o długości N wyznaczamy ze wzoru

$$L_{PPM}(X) = -\gamma \sum_{i=1}^N \lg P_{est}(x_i), \quad (3.4)$$

gdzie przez γ oznaczamy **współczynnik nieefektywności kodera statystycznego**:

$$\gamma = \frac{L}{L_{est}}, \quad (3.5)$$

gdzie L_{est} oznacza długość po zakodowaniu wyliczoną w oparciu o prawdopodobieństwo ustalone przy pomocy modelu, natomiast L długość rzeczywiście uzyskaną po zastosowaniu kodera statystycznego. W przypadku kodowania arytmetycznego wartość współczynnika γ tylko nieznacznie przekracza 1, więc nie odgrywa on w praktyce istotnej roli.

Analogicznie do $L_{PPM}(X)$, przez $L_{LZ}(X)$ oznaczać będziemy długość uzyskaną przy użyciu metody substytucyjnej. W tej chwili nie ma jeszcze potrzeby precyzować sposobu wyliczenia $L_{LZ}(X)$.

3.1.2 Predykcyjno-substytucyjny model źródła danych

Bezstratna kompresja danych ma na celu zmianę postaci zbioru danych w taki sposób, aby jej długość uległa skróceniu, a jednocześnie dało się przywrócić postać oryginalną. Operacja taka jest możliwa za sprawą powtarzalności, którą można często dostrzec w zbiorach danych.

Myśląc o powtarzalności intuicyjnie rozpoznajemy jej dwa podstawowe rodzaje: korelację z kontekstem oraz występowanie powtarzających się wzorów. Choć powtarzalność tych dwóch rodzajów nie jest jedyną możliwą (patrz punkt 1.1.3), to dominuje ona w typowych zbiorach danych poddawanych kompresji, takich jak tekst czy kod programów.

Przykładem zjawiska korelacji z kontekstem niech będzie następująca sytuacja: w zbiorze danych po sekwencji 'ABC' wielokrotnie występował dotąd symbol 'D', rzadko 'E', a nigdy 'F'. Po napotkaniu kolejnej sekwencji 'ABC' można zatem spodziewać się, że symbolem następującym po niej jest raczej 'D' a nie 'F'.

Przykładem zjawiska występowania powtarzających się wzorów niech będzie następująca sytuacja: do pewnego miejsca w zbiorze danych wystąpiła wielokrotnie sekwencja 'ABCD', a ani razu sekwencja 'BACD'. W dalszej części zbioru należy więc spodziewać się, że kolejne wystąpienie sekwencji 'ABCD' jest prawdopodobne, a wystąpienie sekwencji 'BACD' nieprawdopodobne.

Zauważmy, że oba te zjawiska są różnymi interpretacjami tego samego stanu rzeczy. A mianowicie, jeżeli po symbolu 'A' bardzo prawdopodobny jest symbol 'B', po 'B' – 'C', a po 'C' – 'D', to i prawdopodobieństwo sekwencji 'ABCD' musi być wysokie. Z drugiej strony, jeżeli prawdopodobieństwo sekwencji 'ABCD' jest wysokie, to i warunkowe prawdopodobieństwo 'D' po 'ABC' będzie wysokie.

Zauważmy także, że w zależności od struktury źródła skłaniamy się do jednej bądź drugiej interpretacji powtarzalności. W szczególności, jeżeli powtarzających się fraz jest wiele i są one krótkie, budowę źródła wyjaśnić można prościej posługując się interpretacją korelacyjną. Natomiast gdy powtarzających się fraz jest niewiele i są one długie, budowę źródła wyjaśnić można prościej posługując się opisem występujących wzorów. Długość frazy pełni w tym rozróżnieniu szczególną rolę; dlatego będzie ona stanowić jeden z podstawowych parametrów algorytmu łączącego oba podejścia.

Na wykorzystywaniu zjawiska korelacji z kontekstem opierają się metody predykcyjne, natomiast na wykorzystywaniu zjawiska występowania powtarzających się wzorów metody substytucyjne. Wady i zalety takiego podejścia (czysto substytucyjnego bądź czysto predykcyjnego) zostaną omówione w podrozdziale 3.2. Teraz ograniczmy się do przypuszczenia, że źródło danych da się lepiej opisać wykorzystując obydwa rodzaje powtarzalności. Potrzebny jest zatem model źródła danych, w którym ujęto zarówno korelację, jak i wielokrotne występowanie się tych samych sekwencji.

Definicja 3.1. Niech X oznacza źródło danych. Jeżeli źródło to dla pozycji i generuje:

- z prawdopodobieństwem α sekwencję x_i^{i+l-1} , taką że $x_i^{i+l-1} = x_j^{j+l-1}$, $j < i$, $l \geq l_{min}$, a prawdopodobieństwo $P(x_i^{i+l-1} = x_j^{j+l-1})$ zależy od odległości $d = i - j$,
- z prawdopodobieństwem $1-\alpha$ symbol x_i , którego prawdopodobieństwo $P(x_i)$ zależy od kontekstu x_{i-k}^{i-1} o długości $k \leq K$,

to nazwiemy je **źródłem predykcjno-substytucyjnym** rzędu K , a jego minimalna długość substytucji wynosi l_{min} .

Współczynnik α nazywać będziemy **współczynnikiem substytucyjności** źródła danych. Jego estymacja przy nieznanym rzędzie źródła K jest trudna, gdyż źródło predykcjno-substytucyjne może generować identyczne sekwencje symboli niezależnie od wartości α .

Przyjmując pewną niewielką wartość K , i założywszy że $l_{min} > K+1$, to postępując w poniższy sposób:

- oznaczymy przez s_s liczbę tych pozycji źródła dla których rozpoczynają się sekwencje posiadające frazę zgodną o długości $l \geq l_{min}$;
- oznaczymy przez s_p liczbę tych pozycji źródła dla których rozpoczynają się sekwencje posiadające frazę zgodną o długości $l < l_{min}$ i które nie znajdują się wewnątrz fraz zaliczonych do s_s ;

– obliczmy:

$$\alpha^* = \frac{s_s}{s_s + s_p}, \quad (3.6)$$

otrzymamy wartość empirycznego współczynnika substytucyjności α^* będącego przybliżeniem α obarczonym jednak istotnym błędem wynikającym z kwalifikacji jako wygenerowanych substytucyjnie fraz będących złożeniem kilku fraz wygenerowanych predykcyjnie ($\alpha^* \geq \alpha$).

Porównywać wartości współczynników α różnych zbiorów danych można także w inny sposób. Źródłom o α zbliżonym do 1 lepiej odpowiada model substytucyjny, natomiast źródłom o α zbliżonym do 0 lepiej odpowiada model predykcyjny. Można zatem domniemywać, że zbiory danych dla których efektywność kompresji metod substytucyjnych przewyższa efektywność kompresji metod predykcyjnych (o pewnym rzędzie modelu K) mają wysoką wartość α ; odwrotnie, należy spodziewać się, że zbiory danych, dla których efektywność kompresji metod substytucyjnych jest niższa niż efektywność kompresji metod predykcyjnych, mają niską wartość α . Przy czym dopiero spełnienie układu nierówności $L_{LZ}(X) > L_{PPM}(X)$ i $L_{LZ}(Y) < L_{PPM}(Y)$ stanowi warunek dostateczny do stwierdzenia, że $\alpha_X < \alpha_Y$.

3.1.3 Minimalna i zadowalająca długość substytucji

Predykcjno-substytucyjne źródło danych może zostać zakodowane przy użyciu zarówno metod predykcyjnych, jak i substytucyjnych. Istotną kwestią jest zatem określenie własności frazy, która pozwoli wybrać do jej zakodowania jedną z dwóch dostępnych metod, tak by spodziewana efektywność kompresji była jak najwyższa.

W tym punkcie pracy, na przykładzie wybranej metody predykcyjnej i wybranej metody substytucyjnej, tak jak zostały one przedstawione w punkcie 3.1.1, zostanie wskazana długość frazy jako ta własność, która pozwala wybrać bardziej efektywną metodę kompresji dla rozpatrywanej frazy.

Uwaga. W rozważaniach prowadzonych w tym punkcie nie będzie uwzględniany współczynnik nieefektywności kodera statystycznego. Tak jak zostało to już napisane, jego wpływ na całkowitą długość po zakodowaniu jest śladowy, a konieczność uwzględniania go, znacznie skomplikowała by przedstawione niżej wywody.

Najpierw rozpatrzmy sytuację gdy długość frazy $l = 1$. W przypadku PPM jest jasne, że w takiej sytuacji długość wygenerowanego słowa kodowego będzie równa długości słowa kodowego symbolu znajdującego się na wskazanej pozycji źródła i :

$$L_{PPM}(x_i^i) = L_{PPM}(x_i). \quad (3.7)$$

Pozostaje zatem ustalić ile wynosi $L_{LZ}(x_i^i)$. Dla metody substytucyjnej przyjęto, że w przypadku, gdy kodowany symbol nie jest elementem frazy, kodowany jest statystycznie przy pomocy modelu rzędu 0, a zatem:

$$L_{LZ}(x_i^i) = L_{LZ}(\text{Len}(1)) + L_{LZ}(x_i), \quad (3.8)$$

gdzie $L_{LZ}(\text{Len}(1))$ oznacza długość słowa kodowego odpowiadającego długości frazy równej 1. Dokładna wartość $L_{LZ}(\text{Len}(1))$ nie jest na razie określona. Jednak z faktu, że frazy o długości $l > 1$ muszą mieć odpowiadające im słowa kodowe, wynika że

$$L_{LZ}(\text{Len}(1)) > 0. \quad (3.9)$$

Z powyższego wynika że jeżeli $L_{PPM}(x_i) \leq L_{LZ}(x_i)$, to $L_{LZ}(x_i^i) > L_{PPM}(x_i^i)$.

Ograniczenie $L_{PPM}(x_i) \leq L_{LZ}(x_i)$ nie jest przy tym istotne. $L_{LZ}(x_i)$ pochodzi z modelu statystycznego rzędu 0, podczas gdy $L_{PPM}(x_i)$ opiera się na modelu PPM nieznanego rzędu $K \geq 0$. Model wyższego rzędu, jeżeli tylko poparty jest odpowiednio dużą liczbą statystyk, nie może ustępować modelowi niższego rzędu:

$$\lim_{i \rightarrow \infty} P(L(x_i | M_i^{k-1}) < L(x_i | M_i^k)) = 0. \quad (3.10)$$

Dla $i \rightarrow \infty$, $L_{PPM}(x_i^i)$ będzie prawie zawsze mniejsze od $L_{LZ}(x_i^i)$. Spostrzeżenie to jest podstawą do zdefiniowania pojęcia, które znajdzie praktycznie zastosowanie w algorytmie realizującym metodę predykcjno-substytucyjną.

Definicja 3.2. Dla danego zbioru danych X o długości N istnieje **minimalna długość substytucji** $l_{min} \geq 1$, taka że dla każdego $l \leq l_{min}$, $l > 0$ i $i \leq N - l$, prawdopodobieństwo $P(L_{LZ}(x_i^{i+l-1}) < L_{PPM}(x_i^{i+l-1}))$ jest mniejsze od pewnej liczby p_{min} , takiej że $0 < p_{min} < 1$.

Im mniejszą wartość nada się p_{min} , tym mniejszą wartość osiągnie l_{min} . Duża wartość p_{min} odpowiada dużej wartości l_{min} .

Rozpatrzywszy sytuację dla bardzo małego l , należy sprawdzić jak kształtuje się efektywność kompresji obu metod dla bardzo dużego l .

Założmy, że dla pozycji źródła i znaleziono frazę zgodną o długości $l > 1$. Jeżeli przez dL oznaczmy o ile długość frazy zakodowanej predykcyjnie jest krótsza od długości frazy zakodowanej substytucyjnie (w sytuacji odwrotnej dL przyjmuje wartości ujemne), można zapisać:

$$L_{LZ}(x_i^{i+l-1}) = L_{PPM}(x_i^{i+l-1}) + dL. \quad (3.11)$$

Jak zmienia się $L_{LZ}(x_i^{i+l-1})$ i $L_{PPM}(x_i^{i+l-1})$, gdy l zostanie zwiększone o Δl ? Dla PPM będzie to:

$$\Delta L_{PPM} = L_{PPM}(x_i^{i+l-1}) - L_{PPM}(x_i^{i+l+\Delta l-1}) = L_{PPM}(x_{i+l}^{i+l+\Delta l-1}). \quad (3.12)$$

Tymczasowo oznaczmy przez c wartość licznika $q(x_i)$ dla pozycji i , i właściwego jej modelu, zaś przez e różnicę sumy wszystkich liczników i c : $e = T - q(x_i)$, dla tej samej pozycji (zauważmy, że $e \geq q(Esc)$). W oparciu o wzór 3.4 wiadomo, że dla dowolnego $j < \Delta l$ L_{PPM} będzie najmniejsze, gdy $\frac{q(x_j)}{T}$ jest największe. Spośród różnych rodzajów fraz, będzie tak, gdy x_i^{i+l-1} będzie frazą okresową o okresie $d = 1$. Wtedy:

$$\begin{aligned} \Delta L_{PPM} &= L_{PPM}(x_{i+l}^{i+l+\Delta l-1}) - \sum_{j=0}^{\Delta l-1} \lg \frac{c+l+j+e}{c+l+j} = \\ &= \sum_{j=0}^{\Delta l-1} \lg(c+l+j+e) - \sum_{j=0}^{\Delta l-1} \lg(c+l+j) = \sum_{j=\Delta l-e}^{\Delta l-1} \lg(c+l+j+e) - \sum_{j=0}^{e-1} \lg(c+l+j) = \\ &= \sum_{j=0}^{e-1} \lg(c+l+\Delta l+j) - \sum_{j=0}^{e-1} \lg(c+l+j) = \sum_{j=0}^{e-1} \lg \frac{c+l+\Delta l+j}{c+l+j} \\ \Delta L_{PPM} &= \sum_{j=0}^{e-1} \lg \left(1 + \frac{\Delta l}{c+l+j} \right). \end{aligned} \quad (3.13)$$

Dla LZ zaś:

$$\Delta L_{LZ} = L_{LZ}(x_i^{i+l-1}) - L_{LZ}(x_i^{i+l+\Delta l-1}). \quad (3.14)$$

W tym momencie potrzebne jest określenie, w jaki sposób w metodzie substytucyjnej kodowane są frazy o długości $l > 1$. Wiadomo, że do poprawnego ich zdekodowania potrzebna jest para wiadomości: wskaźnik do frazy zgodnej oraz jej długość. Oznaczmy długość słowa kodowego wskaźnika przez $L_{LZ}(\text{Ptr})$, a długość słowa kodowego długości frazy przez $L_{LZ}(\text{Len}(l))$. Otrzymamy wtedy co następuje:

$$\begin{aligned} \Delta L_{LZ} &= L_{LZ}(\text{Ptr}) + L_{LZ}(\text{Len}(l+\Delta l)) - L_{LZ}(\text{Ptr}) - L_{LZ}(\text{Len}(l)) \\ \Delta L_{LZ} &= L_{LZ}(\text{Len}(l+\Delta l)) - L_{LZ}(\text{Len}(l)). \end{aligned} \quad (3.15)$$

Do dalszych rozważań potrzebne jest zdefiniowanie długości zapisu $\text{Len}(l)$. Wiadomo, że każdą liczbę dodatnią n da się zapisać na $L(n) = 2\lg(n) + 1$ bitach (używając na przykład kodu Eliasa [ELIA75]). Prawdopodobnie długość l da się zapisać na mniejszej liczbie bitów. W tej chwili nie jest to jednak istotne. Chodzi bowiem o znalezienie przypadku, w którym $\Delta L_{LZ} < \Delta L_{PPM}$. Skoro taki przypadek dla podanego sposobu kodowania długości l istnieje, tym bardziej musi istnieć przy oszczędniejszym sposobie zapisu długości. Przyjmijmy zatem, że

$$L_{LZ}(\text{Len}(l)) = 2\lg(l) + 1 \quad (3.16)$$

$$\Delta L_{LZ} = 2\lg(l+\Delta l) + 1 - 2\lg(l) - 1 = 2\lg \frac{l+\Delta l}{l}$$

$$\Delta L_{LZ} = 2\lg\left(1 + \frac{\Delta l}{l}\right). \quad (3.17)$$

Zauważmy, że:

$$\lim_{\Delta l \rightarrow \infty} \Delta L_{LZ} = \lim_{\Delta l \rightarrow \infty} \Delta L_{PPM} = +\infty. \quad (3.18)$$

O tym która wielkość będzie rosła szybciej decyduje wykładnik przy Δl . W przypadku ΔL_{LZ} wynosi on 2. W przypadku ΔL_{PPM} wynosi on e . Oznacza to, że jeżeli $e > 2$, przy odpowiednio dużym Δl różnica $L_{PPM}(x_i^{i+l+\Delta l-1}) - L_{LZ}(x_i^{i+l+\Delta l-1})$ musi przekroczyć dL . A zatem, dla każdej pozycji źródła i , dla którego $T - q(x_i) > 2$, istnieje długość frazy l , dla której

$$L_{LZ}(x_i^{i+l-1}) < L_{PPM}(x_i^{i+l-1}). \quad (3.19)$$

Definicja 3.3. Dla danego zbioru danych X o długości N istnieje **zadowalająca długość substytucji** $l_{suf} > 1$, taka że dla każdego $l \geq l_{suf}$ i $i \leq N - l$, prawdopodobieństwo $P(L_{LZ}(x_i^{i+l-1}) > L_{PPM}(x_i^{i+l-1}))$ jest mniejsze od pewnej liczby p_{suf} , takiej że $0 < p_{suf} < 1$.

Analogicznie do p_{min} , im mniejszą wartość nada się p_{suf} , tym mniejszą wartość osiągnie l_{suf} . Duża wartość p_{suf} odpowiada dużej wartości l_{suf} .

3.1.4 Przełączanie. Metody składowe

Załóżmy, że potrzeba skompresować pewien zbiór danych X o długości N mając do dyspozycji dwie metody kodowania: 'B' i 'C'. Jeżeli długość po zakodowaniu $L(X)$ ma być jak najmniejsza, należałoby rozważyć sprawdzenie obu, a następnie wybranie tej, która okaże się bardziej efektywna. Aby można było prawidłowo odkodować tak skompresowany zbiór, zakodowane dane muszą być poprzedzone specjalnym znacznikiem informującym o tym, którą z dwóch możliwych metod użyto do zakodowania zbioru. Znacznik ten nazywać będziemy **przełącznikiem**⁹ i oznaczać Sw .

Z faktu że $L_C(X) < L_B(X)$ nie wynika, że dla każdego $x \in X$, $L_C(x) < L_B(x)$. Na przykład w poprzednim punkcie (3.1.3) pokazano, że dla źródła predykcyjno-substytucyjnego mogą istnieć zarówno frazy, dla których długość zapisu PPM jest krótsza od długości zapisu LZ, jak i takie dla których długość zapisu LZ jest krótsza od długości zapisu PPM.

⁹ W dalszej części pracy nazwa ta służyć będzie również na określenie części algorytmu zajmującej się określaniem stanu przełącznika w podanym tu znaczeniu.

Oznacza to, że chcąc zminimalizować długość zapisu przy użyciu dwóch metod powinno się rozważyć przełączanie przed każdym elementem zbioru danych. W takiej sytuacji istotnego znaczenia nabiera długość zapisu samego przełącznika $L(Sw)$. Metodę ‘C’ wybrać należy bowiem do zakodowania symbolu x wtedy i tylko wtedy, gdy

$$L_C(x) + L(Sw=C) \leq L_B(x) + L(Sw=B). \quad (3.20)$$

Metodę ‘B’ zaś w przypadku odwrotnym:

$$L_C(x) + L(Sw=C) \geq L_B(x) + L(Sw=B). \quad (3.21)$$

Każdą metodę uwzględnianą przy dokonywaniu wyboru sposobu kodowania nazywać będziemy **metodą składową**. Sam zaś sposób dokonywania tego wyboru nazywać będziemy **metodą przełączania**.

Zastanawiając się nad efektywnością takiego sposobu kodowania zbioru danych należy rozważyć dwie kwestie.

1. Ryzyko: Jakie warunki muszą być spełnione, aby całkowita długość zbioru danych zakodowanego z przełączaniem nie przekroczyła długości uzyskanej przez gorszą z dwóch rozpatrywanych metod?
2. Szansa: Jakie warunki muszą być spełnione, aby całkowita długość zbioru danych zakodowanego z przełączaniem była krótsza od długości uzyskanej przez lepszą z dwóch rozpatrywanych metod?

Ad. 1. Przypadek w którym wprowadzenie przełączania spowoduje wydłużenie długości zakodowanego zbioru danych odpowiada sytuacji w której, przy założeniu że ‘C’ oznacza lepszą, a ‘B’ gorszą z dwóch metod w ujęciu globalnym:

$$\sum_{i \in C} (L_C(x_i) + L(Sw = C)) + \sum_{i \in B} (L_B(x_i) + L(Sw = B)) > \sum_{i \in C \cup B} L_B(x_i), \quad (3.22)$$

gdzie $i \in M$ oznacza pozycje źródła dla których metoda ‘M’ okazała się lepsza.

Dla dowolnej pozycji źródła wydłużenie nastąpi wtedy, gdy przy założeniu że ‘C’ oznacza lepszą, a ‘B’ gorszą z dwóch metod dla wybranej pozycji źródła:

$$L_C(x_i) + L(Sw = C) > L_B(x_i),$$

czyli wtedy, gdy:

$$L_B(x_i) - L_C(x_i) < L(Sw = C). \quad (3.23)$$

Dla efektywności kodowania z użyciem przełączania ważne jest zatem, by różnica w długości słów kodowych generowanych przez obie metody składowe była jak największa. Będzie tak raczej, gdy metody składowe oparte będą na modelach źródła różnego typu; jeżeli bowiem obie metody opierać się będą na zbliżonych modelach, mało prawdopodobne jest by generowały one znacząco różne długości słów kodowych. Warunek ten nazywać będziemy warunkiem **różnorodności metod składowych**, a metody spełniające go **metodami komplementarnymi**.

Ryzyko wydłużenia długości zbioru danych zakodowanego z przełączaniem można prawie całkowicie wyeliminować stosując przełączanie z zasadą wyłączania z pasmem pośrednim (patrz 3.1.5). Jednak implementacja takiej metody byłaby zapewne zbyt powolna dla praktycznych zastosowań, gdyż wymagałaby dla każdej pozycji źródła oceny efektywności przełączania dla wszystkich symboli alfabetu.

Ad. 2. Przypadek w którym wprowadzenie przełączania spowoduje skrócenie długości zakodowanego zbioru danych odpowiada sytuacji w której (przy założeniu że 'C' oznacza lepszą, a 'B' gorszą z dwóch metod w ujęciu globalnym):

$$\sum_{i \in C} (L_C(x_i) + L(Sw = C)) + \sum_{i \in B} (L_B(x_i) + L(Sw = B)) < \sum_{i \in C \cup B} L_C(x_i), \quad (3.24)$$

gdzie $i \in M$ oznacza pozycje źródła dla których metoda 'M' okazała się lepsza.

Dla dowolnej pozycji źródła skrócenie nastąpi wtedy, gdy (przy założeniu że 'C' oznacza lepszą z dwóch metod w ujęciu globalnym, a 'B' lepszą z dwóch metod dla wybranej pozycji źródła):

$$L_B(x_i) + L(Sw = B) < L_C(x_i), \quad (3.25)$$

czyli wtedy, gdy:

$$L_C(x_i) - L_B(x_i) > L(Sw = B). \quad (3.26)$$

Można zatem powiedzieć, że drugim po różnorodności metod składowych, czynnikiem warunkującym efektywność kodowania z użyciem przełączania jest, by dla możliwie jak największej liczby pozycji źródła, gorsza z dwóch metod w ujęciu globalnym generowała krótsze słowa kodowe. Będzie tak, gdy efektywność obu metod składowych będzie porównywalna; jeżeli bowiem jedna z uwzględnianych metod znacząco odstawać będzie efektywnością kompresji od drugiej, mało prawdopodobne jest by przełączanie pomiędzy nimi mogło doprowadzić do skrócenia długości kodowanego zbioru danych. Warunek ten nazywać będziemy warunkiem **równoprawności metod składowych**, a metody spełniające go **metodami konkurencyjnymi**.

3.1.5 Kod niepełny. Zasady wyłączania

Mając do dyspozycji system kodowania z przełączaniem wykorzystujący dwie metody składowe 'C' i 'B', nie ma wymogu by każda z metod składowych określała słowa kodowe dla wszystkich symboli alfabetu. W istocie, aby można było zakodować dowolne źródło danych opisane na pewnym alfabecie A , wymagane jest jedynie to, by każdy symbol tego alfabetu posiadał przydzielone mu słowo kodowe przynajmniej przez jedną z metod składowych. A zatem, jeżeli symbol $a \in A$ nie posiada słowa kodowego przydzielonego mu przez pewną metodę 'B', to:

$$L_B(a) = \infty, \quad (3.27)$$

$$L_C(a) \leq L_{\max}, \quad (3.28)$$

gdzie $L_{\max}$ oznacza najdłuższą dozwoloną długość słowa kodowego.

Kod przydzielający słowa kodowe tylko niektórym symbolom alfabetu źródła nazywamy **kodem niepełnym**. Odpowiednio przez **kod pełny** rozumiemy kod przydzielający słowa kodowe wszystkim symbolom alfabetu źródła.

Jeżeli w metodzie kodowania z przełączaniem, obie metody składowe dysponują kodem pełnym bądź niepełnym, to taką sytuację nazywamy **przełączaniem symetrycznym**. Jeżeli natomiast jedna z metod składowych dysponuje kodem pełnym, druga zaś niepełnym, to taką sytuację nazywamy **przełączaniem asymetrycznym**.

Klasycznym przypadkiem metody kodowania z przełączaniem jest sam algorytm PPM. Metodami składowymi odpowiadają tu modele różnego rzędu, w liczbie $K+2$, gdzie K jest równe rzędowi całego modelu PPM. Jest to przełączanie asymetryczne, gdyż jedynie model rzędu -1 dysponuje kodem pełnym.

Do stosowania w PPM zdefiniowano dwie zasady wyłączenia, które poniżej przedstawione zostaną w formie bardziej ogólnej.

Zasada wyłączenia symboli. Reguła ta mówi, że jeżeli do zakodowania symbolu a wybieramy metodę składową 'B', to dla każdej innej metody składowej 'C'

$$\begin{aligned} P_C(a) &= 0, \\ L_C(a) &= \infty. \end{aligned} \quad (3.29)$$

W przypadku, gdy przełącznikiem jest długość zakodowanego symbolu, zasadę tę zapiszemy następująco:

$$P'_B(a) = \begin{cases} P_B(a), & L_B(a) + L(Sw = B) \leq L_C(a) + L(Sw = C) \\ 0, & L_B(a) + L(Sw = B) > L_C(a) + L(Sw = C) \end{cases} \quad (3.30)$$

gdzie $P_B(a)$ oznacza prawdopodobieństwo przypisane symbolowi a przez metodę składową 'B', a $P'_B(a)$ prawdopodobieństwo symbolu a uwzględniające zasadę wyłączenia symboli.

Jak zostało to omówione w punkcie 3.1.4, przełączanie gdy różnice w długości słów kodowych pomiędzy obiema metodami są niewielkie nie jest uzasadnione. Rozwiązaniem jest tu stosowanie **zasady wyłączenia z pasmem pośrednim**, która definiuje pasmo pośrednie różnicy długości słów kodowych o szerokości $2\varepsilon+1$ dla którego przełączania nie stosuje się:

$$L(a) = \begin{cases} L_B(a) + L(Sw = B), & L_C(a) + L(Sw = C) - L_B(a) - L(Sw = B) \geq \varepsilon \\ L_{BC}(a), & |L_B(a) + L(Sw = B) - L_C(a) - L(Sw = C)| < \varepsilon, \\ L_C(a) + L(Sw = C), & L_B(a) + L(Sw = B) - L_C(a) - L(Sw = C) \geq \varepsilon \end{cases} \quad (3.31)$$

gdzie L_{BC} oznacza długość słowa kodowego ustaloną bez stosowania przełączania, w oparciu o jedną z metod składowych lub obie (ważone).

Metoda kodowania z przełączaniem, która nie uwzględnia zasady wyłączania symboli, posługuje się kodem nieoptymalnym, gdyż pojedynczemu symbolowi odpowiadać w niej może tyle różnych słów kodowych ile jest jej metod składowych.

Z drugiej strony, stosowanie tej zasady negatywnie rzutuje na szybkość działania algorytmu. Ponadto, jeżeli metody składowe posługują się różnymi alfabetami, to użycie jej dodatkowo się komplikuje. Przykładowo, alfabetem metody PPM jest alfabet źródła, podczas gdy alfabetem metody LZ jest słownik fraz. Wykluczenie frazy z modelu PPM, w odróżnieniu od wyłączenia pojedynczego symbolu, jest czynnością skomplikowaną, ze względu na trudność wyznaczenia wartości o którą należałoby zmniejszyć prawdopodobieństwa symboli tworzących wykluczaną frazę.

Zasada wyłączania aktualizacji. Jeżeli modele obu metod składowych są aktualizowane na bieżąco z kodowaniem symboli źródła, to w przypadku wybrania metody 'B' do zakodowania symbolu x_i pojawia się problem: w którym modelu należy odnotować fakt wystąpienia x_i . Znane są trzy rozwiązania tego problemu:

1. W klasycznej aktualizacji bez wyłączeń należy aktualizować wszystkie modele.
2. Zasada wyłączania aktualizacji, stosowana w PPMC [HOWA93], mówi, że należy aktualizować tylko model uznany za właściwy do zakodowania x_i .
3. Stosując podejście pośrednie, stosowane w PPMII [SZKA02a], należy aktualizować na pewno jedynie model uznany za właściwy do zakodowania x_i , pozostałe zaś modele tylko pod pewnymi warunkami.

Wadą aktualizacji bez wyłączeń jest to, że zawyża prawdopodobieństwo wystąpienia symbolu w modelu, który nie został wybrany do jego zakodowania, i być może nigdy nie zostanie wybrany. Niepotrzebnie wydłuża to długość słów kodowych wszystkich pozostałych, zapewne bardziej dla tego modelu typowych, symboli.

Wadą stosowania zasady wyłączania aktualizacji jest to, że w nieuzasadniony sposób wiąże pewne symbole z pewnymi modelami. Jeżeli dla pewnej liczby symboli 'a' metoda 'B' okazała się lepsza od 'C', to prawdopodobieństwo wybrania w przyszłości do zakodowania symbolu 'a' metody 'C' maleje. Wynika to stąd, że z każdym kodowanym symbolem 'a' jego prawdopodobieństwo w modelu 'B' rośnie, podczas gdy w modelu 'C' nie. Symbol 'a' zostanie zatem dość trwale przypisany do metody 'B', mimo że globalnie lepszym estymatorem dla późniejszych jego wystąpień

mogła okazać się metoda ‘C’. Negatywne skutki tego zjawiska redukuje skalowanie, ale tylko w pewnym stopniu.

Podejście pośrednie mieści się pomiędzy tymi dwoma skrajnościami, a zatem konsekwencją jego użycia będą oba wymienione zjawiska, ale o znacznie mniejszej sile.

Oprócz efektywności kompresji ważna jest także jej szybkość. Tutaj bezsprzecznie wygrywa aktualizacja z wyłączaniem, gdyż dla każdego symbolu źródła aktualizuje wyłącznie jeden model. Najwolniejsza jest aktualizacja bez wyłączeń, która dla każdego symbolu źródła aktualizuje modele wszystkich metod składowych. Ponownie podejście pośrednie mieści się pomiędzy dwoma wcześniejszymi, gdyż pozostałe modele aktualizowane są w nim tylko dla niektórych symboli źródła. Konsekwencją takiego stanu rzeczy jest, że dążąc do zachowania dużej szybkości działania należy faworyzować przełączanie z wyłączaniem aktualizacji, unikać zaś pełnej aktualizacji bez wyłączeń.

3.2 Przesłanki do poszukiwania nowej metody

3.2.1 Krytyczna dyskusja istniejących metod

Większość istniejących implementacji bezstratnych metod kompresji, a jeszcze bardziej tych które weszły do powszechnego użytku, opiera się na metodach substytucyjnych, w szczególności modyfikacjach algorytmu LZ77 [SWAC99].

Z kolei najlepsze znane rezultaty efektywności kompresji osiągnięto stosując metody predykcyjne, a w szczególności odmiany algorytmu PPM (patrz 2.6).

Dlaczego zatem użytkownicy wybierają „gorsze” programy stosujące LZ nad „lepsze” wykorzystujące PPM? Z pewnością duże znaczenie mają tu czynniki pozamerytoryczne, takie jak przyzwyczajenie do istniejących standardów, czy też nieznamość nowszych programów. Nie należy ich jednak przeceniać. Czy programy kompresujące substytucyjnie są z punktu widzenia użytkownika rzeczywiście gorsze od tych opartych na PPM?

W punkcie 1.1.8 wskazano na czas jako istotny czynnik, który obok efektywności kompresji należy brać pod uwagę przy ocenianiu metod kompresji. Należy sprawdzić więc, czy przewaga PPM nad LZ utrzyma się, gdy jako miarę porównawczą długość po zakodowaniu zastąpi się całkowitym czasem transmisji.

Tabela 3.1 zawiera rezultaty dla kilku plików testowych (opisanych w 4.1.1). Do testów użyto w roli reprezentanta metody substytucyjnej – *Zip* 2.3 [INFO99], w roli reprezentanta metody predykcyjnej – *PPMd* v.I [SZKA02b]. Środowisko testowe zostało opisane w punkcie 4.1.2.

Tabela 3.1. Porównanie rozmiaru po kompresji (L) i całkowitego czasu transmisji (CCT , przy prędkości 512 kb/s) dla Zipa (tryb domyślny) i PPMd (rzęd 8)

Metoda:	Zip (LZ)		PPMd (PPM)	
Plik	L (kB)	CCT (s)	L (kB)	CCT (s)
<i>Exe</i>	340,9	5,97	297,0	7,60
<i>Txt</i>	217,5	4,04	155,3	4,21
<i>Xml</i>	63,8	1,53	63,0	1,91

Źródło: badania własne.

Mimo że każdy z testowych plików został skompresowany z użyciem algorytmu typu PPM efektywniej niż z użyciem algorytmu typu LZ, to jednocześnie w każdym przypadku całkowity czas transmisji był krótszy w przypadku programu stosującego algorytm typu LZ.

Pierwszą kwestią która ma znaczenie dla uzyskanych rezultatów jest założona prędkość transmisji. Jeżeli prędkość tę zwiększy się, na znaczeniu straci efektywność kompresji, zyska zaś szybkość (przewaga algorytmów substytucyjnych). Jeżeli prędkość tę zmniejszy się, na znaczeniu straci szybkość kompresji, zyska zaś efektywność (przewaga algorytmów predykcyjnych). Przyjęta w powyższym teście prędkość 512 kb/s dobrana została dla środowiska testowego tak, by nie akcentować przesadnie tylko jednej z dwóch rozpatrywanych wielkości.

Tabela 3.2. Porównanie rozmiaru po kompresji (L) i całkowitego czasu transmisji (CCT , przy prędkości 512 kb/s) dla PPMd, rząd modelu 2, 4 i 8

Rząd PPM:	2		4		8	
Plik	L (kB)	CCT (s)	L (kB)	CCT (s)	L (kB)	CCT (s)
<i>Exe</i>	334,0	7,13	305,0	7,39	297,0	7,60
<i>Txt</i>	201,2	3,69	162,6	3,74	155,3	4,21
<i>Xls</i>	125,0	2,79	79,2	2,08	63,0	1,91

Źródło: badania własne.

Druga z kwestii to parametry pracy programów testowych. W przypadku Zipa, różnice pomiędzy uzyskiwaną efektywnością nie są wielkie, a domyślny tryb '6' osiąga zwykle najlepsze CCT . W przypadku PPMd, sytuacja jest nieco bardziej złożona. Ilustruje ją tabela 3.2, która zawiera rezultaty dla różnych wielkości rzędu modelu PPM.

Przy zmniejszeniu rzędu PPM, czas kompresji i dekompresji znacząco maleje (w przypadku pliku *Txt* aż ponad trzykrotnie). Jeżeli towarzyszy temu znacząca degradacja efektywności kompresji (plik *Xml*), CCT ulega wydłużeniu. Gdy tak nie jest, przejście na model niższego rzędu oznacza skrócenie CCT . Porównując te rezultaty z wynikami w tabeli 3.1 widać, że po redukcji rzędu, PPMd jest lepszy od Zipa pod względem CCT tylko dla pliku tekstowego *Txt*.

Wnioski z powyższych rozważań ująć można następująco:

1. Metody substytucyjne wyraźnie ustępują predykcyjnym efektywnością dla większości spotykanych typów danych.
2. Metody predykcyjne wyższych rzędów są powolne na tyle, by zakwestionować ich wynikającą z efektywności przewagę.
3. Metody predykcyjne niższych rzędów działają szybciej, ale często jeszcze szybciej spada ich efektywność.

3.2.2 Zasadność połączenia predykcji i substytucji

Przed rozpoczęciem prac nad stworzeniem rozwiązania hybrydowego opartego o dwie istniejące metody, należy wpierw zastanowić się, czy spełnione są trzy przesłanki:

1. Połączenie tych dwóch metod jest technicznie wykonalne.
2. Jedna z metod nie okazuje się lepsza od drugiej w każdej sytuacji (warunek równoprawności metod składowych).
3. Metody mają na tyle odmienne zasady działania, by ich połączenie mogło dać nową jakość, lepszą niż krokowe doskonalenie wybranej z tych metod (warunek różnorodności metod składowych).

Zaczynając od pierwszego z postawionych zagadnień, zarówno LZ, jak i PPM przetwarzają dane źródłowe w ich oryginalnej kolejności, symbol po symbolu. Przełączanie pomiędzy nimi może odbywać się niezależnie dla każdej pozycji źródła. Połączenie metod typu LZ i PPM można uznać zatem za technicznie wykonalne.

W odpowiedzi na pozostałe dwie kwestie pomocna będzie tabela 3.3, w której zestawiono ze sobą najważniejsze wady i zalety metod typu LZ i PPM.

Metoda predykcyjna świetnie modeluje korelację niskiego rzędu, natomiast substytucyjna korelację wysokiego rzędu i lokalność referencji. Po pierwsze, istnieją typy danych, w których dominuje jeden lub drugi typ korelacji. Należy więc uznać te dwie metody za konkurencyjne, a warunek równoprawności metod składowych za spełniony. Tym bardziej będzie on spełniony, gdy weźmie się pod uwagę miarę kompresji uwzględniającą czas działania. Nawet dla typów danych, dla których efektywność kompresji metody substytucyjnej ustępuje metodzie predykcyjnej, często na korzyść tej pierwszej przemawia szybkość kompresji.

Po drugie, w większości popularnych typów danych występują oba typy korelacji. Jednoczesne użycie predykcji i substytucji pozwoli na właściwe wymodelowanie korelacji niezależnie od jej rzędu. Należy więc uznać te dwie metody za komplementarne, a warunek różnorodności metod składowych za spełniony.

Tabela 3.3. Porównanie wybranych cech metody predykcijnej i substytucyjnej

	LZ	PPM
Szybkość działania	Wysoka	Niska
Modelowanie korelacji niskiego rzędu	Brak	Bardzo dobre
Modelowanie korelacji wysokiego rzędu	Dobre	Słabe
Wykorzystanie lokalności referencji	Bardzo dobre	Słabe
Efektywność dla typowych źródeł	Średnia	Dobra

3.3 Zarys metody predykcijno-substytucyjnej

3.3.1 Podstawy metody predykcijno-substytucyjnej

Omówione dotychczas zagadnienia wydają się stanowić wystarczającą podstawę do przedstawienia metody predykcijno-substytucyjnej. W kolejnych krokach wiodących od ogółu do szczegółu zostanie sprecyzowane co właściwie należy rozumieć pod pojęciem metody predykcijno-substytucyjnej.

Nowa metoda musi zawierać się w pewnych z góry ustalonych ramach. Opierając się na koncepcji źródła predykcijno-substytucyjnego (definicja 3.1), ramy te ujęto w postaci poniższych definicji.

Definicja 3.4. Jeżeli metoda kompresji przy kodowaniu źródła danych X dla każdej należącej do tego źródła frazy x_i^{i+L-1} dokonuje wyboru pomiędzy zakodowaniem tej frazy z uwzględnieniem jej kontekstu $x_{i-k_f}^{i-1}$ o maksymalnej długości k_f , $k_f \geq 1$, i przejściem do pozycji źródła $i + L$, a zakodowaniem jej pierwszego symbolu x_i z uwzględnieniem jego kontekstu $x_{i-k_s}^{i-1}$ o maksymalnej długości k_s , $k_s \geq 1$, i przejściem do pozycji źródła $i + 1$, to nazywamy ją **metodą predykcijno-substytucyjną**.

W tak pojmowanej metodzie predykcijno-substytucyjnej wyróżnić można cztery podstawowe elementy składowe:

- Przełącznik, którego zadaniem jest wybór jednego z dwóch dostępnych modeli,
- Komponent substytucyjny, który służy do modelowania fraz,
- Komponent predykcyjny, który służy do modelowania pojedynczych symboli,
- Koder statystyczny, który służy do kodowania danych pochodzących z pozostałych komponentów.

Definicja 3.4 określa wąskie znaczenie metody predykcijno-substytucyjnej, gdyż wymaga jednocześnie by obie metody składowe uwzględniały kontekst ($k_f \geq 1$ i $k_s \geq 1$). W szerokim znaczeniu, którym nie będziemy się tu zajmować, do uznania metody za predykcijno-substytucyjną wystarcza, by choć jedna z jej metod składowych uwzględniała kontekst ($k_f \geq 1$ lub $k_s \geq 1$).

W zależności od złożoności stosowanego kryterium przełączania, rozróżnić można dwie podstawowe odmiany metody predykcyno-substytucyjnej:

Definicja 3.5. Jeżeli dla danej metody predykcyno-substytucyjnej wybór pomiędzy zakodowaniem frazy x_i^{i+l-1} a zakodowaniem jej pierwszego symbolu x_i zależy wyłącznie od długości frazy l , to nazywamy ją **metodą predykcyno-substytucyjną bez estymacji długości zapisu**.

Definicja 3.6. Jeżeli dla danej metody predykcyno-substytucyjnej wybór pomiędzy zakodowaniem frazy x_i^{i+l-1} a zakodowaniem jej pierwszego symbolu x_i zależy od długości frazy l i długości zakodowanej frazy $L(x_i^{i+l-1})$ dla każdej z metod składowych, to nazywamy ją **metodą predykcyno-substytucyjną z estymacją długości zapisu**.

Ze względu na kryterium ogólności stosowania, metodę predykcyno-substytucyjną klasyfikujemy jako metodę uniwersalną. Nie jest ona bowiem przygotowana pod kątem szczególnego typu danych, ale każdego typu, który określić można jako wygenerowany przez źródło predykcyno-substytucyjne. Użyty w niej model oparty jest wyłącznie na zaobserwowanych własnościach strukturalnych przetwarzanych danych, nie bierze pod uwagę ich semantyki.

Podobnie jak inne metody uniwersalne, metoda predykcyno-substytucyjna mogłaby posłużyć za podstawę do zbudowania metody dedykowanej; dostosowanie do wybranego typu danych mogłoby mieć na przykład formę przekształcenia wstępnego, poprzedzającego właściwą kompresję. W niniejszej pracy pomysł ten nie będzie jednak rozwijany, skupiono się tu na algorytmie predykcyno-substytucyjnym jako takim.

Ze względu na kryterium adaptacyjności modelu, metodę predykcyno-substytucyjną klasyfikujemy jako metodę o modelu adaptacyjnym. Metoda predykcyno-substytucyjna rozpoczyna kompresję z modelem pustym, w którym nie ma gotowych modeli wyższych rzędów, a jedyny dostępny model statystyczny rzędu -1 przypisuje wszystkim możliwym symbolom źródła takie samo prawdopodobieństwo. Model staje się w pełni gotowy dopiero w momencie przetworzenia wszystkich danych.

Metoda predykcyno-substytucyjna jest metodą dynamiczną. Używany model rozbudowywany i doskonalony jest w trakcie procesu kompresji, a nie przed jego rozpoczęciem. Rozbudowa modelu i kodowanie danych wykonywane są w niej naprzemiennie. W przypadku schematu dwuprzebiegowego (patrz następny punkt pracy), dynamiczny charakter mają oba wykonywane przebiegi.

Charakterystyczną własnością metody predykcyno-substytucyjnej jest asymetryczność jej modeli. Wyraża się ona tym, że dowolna fraza źródła x_i^{i+l-1} może

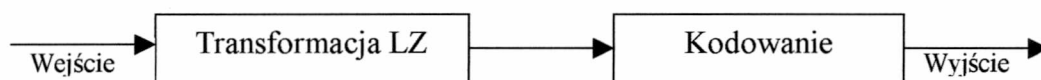
być zakodowana przy pomocy modelu predykcyjnego, natomiast tylko niektóre frazy źródła x_i^{i+l-1} mogą być zakodowane przy pomocy modelu substytucyjnego.

Metoda predykcijno-substytucyjna może być określona jako asymetryczna również ze względu na różną złożoność obliczeniową algorytmów kompresji i dekompresji danych. Algorytm kompresujący musi bowiem obejmować wyszukanie frazy zgodnej, podczas gdy algorytm dekompresujący jest od tej czynności zwolniony.

3.3.2 Schematy działania metody predykcijno-substytucyjnej

Istnieją zasadniczo trzy schematy działania metody predykcijno-substytucyjnej: dwa dla wariantu bez estymacji długości zapisu i jeden dla wariantu z estymacją długości zapisu [SWAC04].

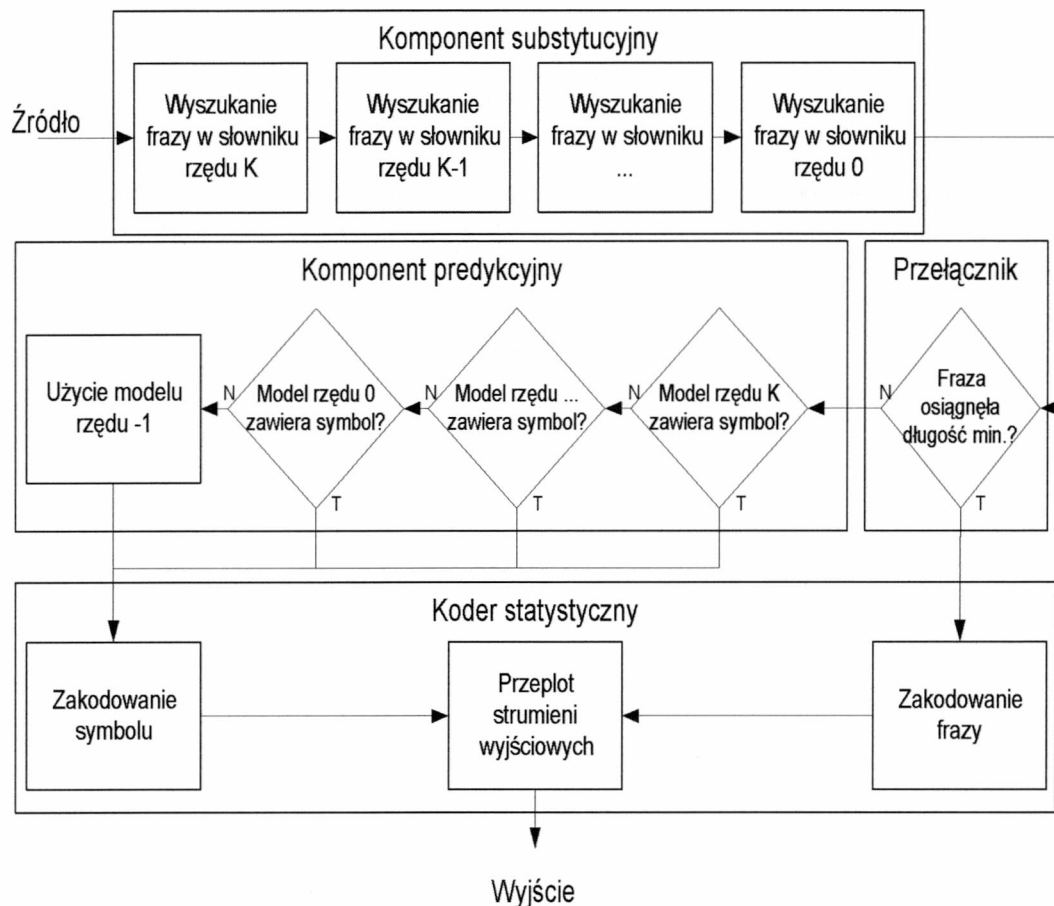
Schemat dwuprzebiegowy. Sposób działania jest tu następujący: w pierwszym przebiegu dokonuje się transformacji danych źródłowych (transformacji LZ), polegającej na podmianie wszystkich wytypowanych przez przełącznik sekwencji symboli, na ich identyfikator umożliwiający późniejsze dokonanie transformacji odwrotnej, zwykle w postaci: (*znacznik podmiany*, *indeks słownika*, *długość podmienionej frazy*). W drugim przebiegu tak przekształcone dane są modelowane i kodowane statystycznie. Algorytm ten ilustruje rysunek 3.1.



Rys. 3.1. Ogólny schemat dwuprzebiegowej realizacji metody predykcijno-substytucyjnej dla całego zbioru danych

Niewątpliwie największą zaletą algorytmu dwuprzebiegowego jest to, że poprzez wyodrębnienie osobnego etapu substytucji, pozwala na użycie dowolnej metody kompresji w charakterze komponentu predykcyjnego. Podstawową wadą jest niszczenie kontekstu dla symboli następujących po podmienionej frazie – zamiast ostatnich kilku symboli podmienionej frazy, kontekstem staje się identyfikator podmiany. Nie ma w nim także możliwości weryfikacji opłacalności podmiany.

Schemat jednoprzebiegowy bez estymacji długości zapisu. Sposób jego działania dokładnie odpowiada definicjom 3.4 i 3.5: dla każdej pozycji źródła określa się długość jej najdłuższej frazy zgodnej. Na jej podstawie przełącznik wybiera czy zakodować substytucyjnie całą zaczynającą się tu frazę, czy predykcyjnie jej pierwszy symbol. Ten schemat działania ilustruje rysunek 3.2.

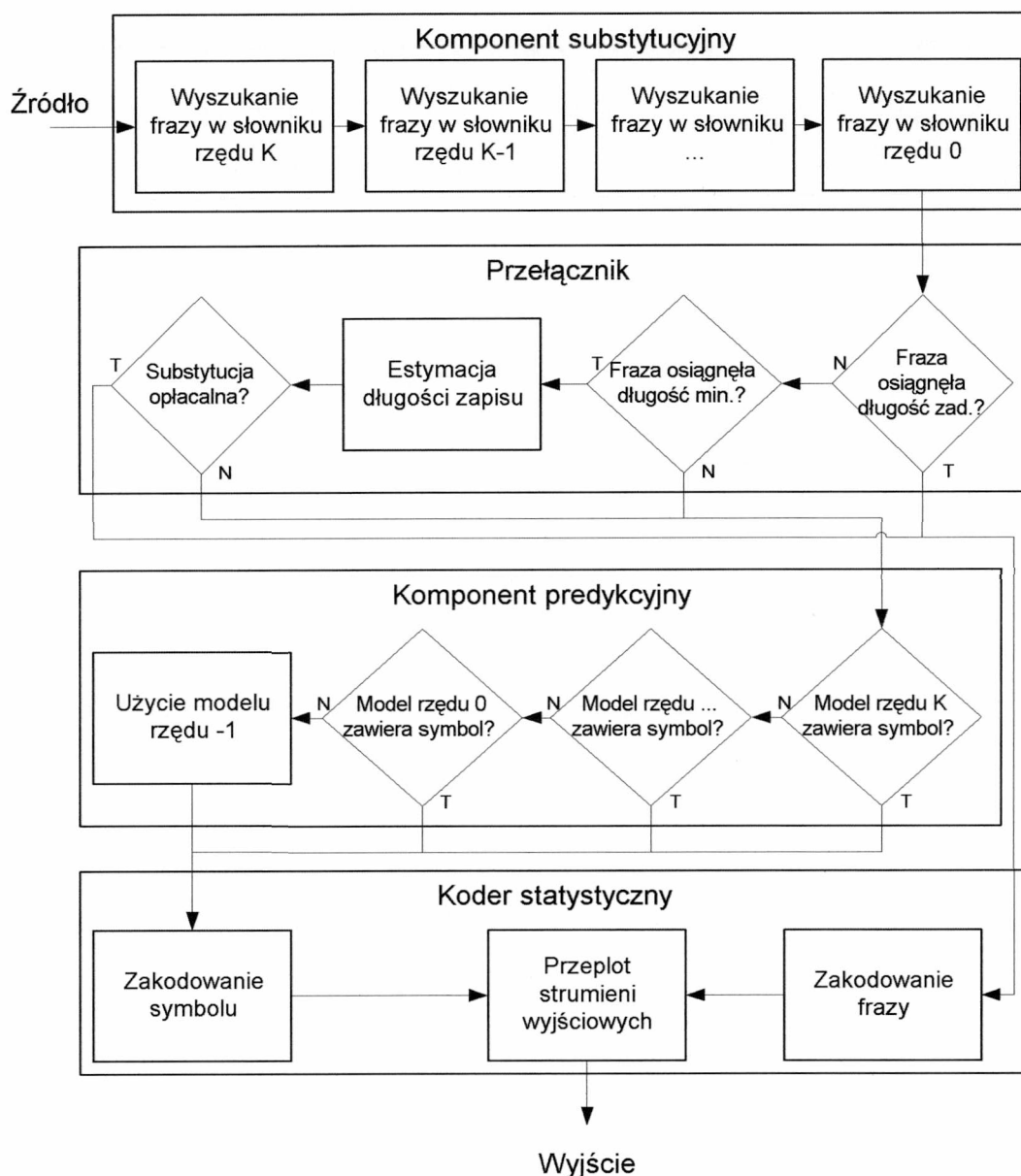


Rys. 3.2. Schemat jednorzbiegowej realizacji metody predykcyjno-substytucyjnej bez estymacji długości zapisu

Algorytm jednorzbiegowy integruje obie metody składowe, dlatego może w ich charakterze użyć tylko algorytmy spełniające pewne warunki (przede wszystkim sekwencyjnego przetwarzania danych). Umożliwia właściwe modelowanie kontekstu dla symboli następujących po podmienionej frazie. Nie ma możliwości weryfikacji opłacalności podmiiany, choć może stosować jej heurystyczne namiastki, na przykład dopasowując minimalną długość substytucji w zależności od średniej długości kodu uzyskanej dla dotychczasowych danych (czego użyteczność jest ograniczona). Ponadto, algorytm jednorzbiegowy jest zwykle szybszy od odpowiadającego mu algorytmu dwurzbiegowego – tym bardziej, jeżeli korzysta z ujednoliconych struktur danych dla obu modeli.

Schemat jednorzbiegowy z estymacją długości zapisu. Sposób działania jest tu następujący: dla każdej pozycji źródła określa się długość jej najdłuższej frazy zgodnej. Dla niej wylicza się długość zapisu dla obu dostępnych metod składowych. Dopiero na jej podstawie przełącznik wybiera czy zakodować substytucyjnie całą zaczynającą się tu frazę, czy predykcyjnie jej pierwszy symbol. Ten schemat działania ilustruje rysunek 3.3.

Wprowadzenie do algorytmu jednorazowej estymacji długości zapisu zazwyczaj poprawia efektywność kompresji. Z drugiej strony powoduje spadek szybkości działania.



Rys. 3.3. Schemat jednorazowej realizacji metody predykcijno-substytucyjnej z estymacją długości zapisu

3.3.3 Problemy badawcze w obszarze metody predykcijno-substytucyjnej

W trakcie projektowania algorytmów, które zostaną przedstawione w podrozdziałach 3.4 i 3.5, w naturalny sposób pojawiły się pewne specyficzne dla metody predykcijno-substytucyjnej problemy badawcze.

Problemy te podzielić można ze względu na elementy składowe metody, które je generują.

Problemy dotyczące przełącznika (w nawiasach podano punkt pracy, w którym w pierwszej kolejności należy szukać odpowiedzi na dany problem):

1. Ile powinna wynosić minimalna długość frazy, dla której nastąpi przełączenie na tryb substytucyjny (4.2.2).
2. W jaki sposób zapisać przełącznik, by zminimalizować jego zakodowaną długość (3.4.3).
3. Jak możliwie najszybciej wykonać estymację długości zapisu (3.5.1).

Problemy dotyczące komponentu substytucyjnego:

1. Jakie rozmiary i rzędy powinny mieć słowniki, by umożliwiały podmianę jak największej liczby fraz nie przekraczając założonej prędkości działania (4.2.2).
2. Według jakich zasad zorganizować słowniki, by dla danego ich rozmiaru umożliwiały podmianę jak największej liczby fraz (4.2.2).
3. Jak zapisać parę danych (*identyfikator pozycji słownika, długość podmiany*), by zminimalizować jej zakodowaną długość (3.4.3).
4. Jak możliwie najszybciej znaleźć frazę zgodną (3.4.3).

Problemy dotyczące komponentu predykcyjnego:

1. Jaki rząd powinien mieć model PPM, by zapewnić dobrą efektywność nie przekraczając założonej prędkości działania (4.2.1).
2. Co można zrobić, aby poprawić efektywność predykcji (3.4.4).
3. Jak możliwie najszybciej dokonać predykcji (3.4.4).

Problemy dotyczące kodera statystycznego:

1. W jaki sposób zorganizować strumień wyjściowy, aby podczas dekompresji każdy z komponentów miał dostęp do właściwych sobie danych (3.4.3).
2. Jak możliwie najszybciej wykonać kodowanie (2.1.3).

3.4 Algorytm predykcjno-substytucyjny bez estymacji długości zapisu

3.4.1 Struktury danych

Algorytm realizujący wariant metody predykcjno-substytucyjnej bez estymacji długości zapisu zawiera wszystkie zaproponowane przez Autora rozwiązania za wyjątkiem tych dotyczących samej estymacji długości zapisu.

Nadrzędnym celem, który przyświecał skonstruowaniu tego algorytmu była maksymalizacja efektywności kompresji w czasie (wyrażona wzrostem współczynnika przyspieszenia transmisji). W związku z tym, niektóre zastosowane w nim rozwiązania

wpływają negatywnie na efektywność kompresji, poprawiają jednak szybkość jego działania.

Wszelkie odniesienia do wielkości struktur opierają się na wartościach dla 32-bitowej architektury Intela (IA-32). W szczególności, przez BYTE rozumiemy typ 8-bitowy, WORD – 16-bitowy, CARDINAL/INTEGER/POINTER – 32-bitowy.

Struktury wspólne. Współużytkowanie przez oba komponenty tych samych struktur danych poprawia efektywność pamięci podręcznej. Do takich struktur zaliczamy przede wszystkim: tablicę kontekstów oraz nagłówki kontekstu.

Tablica_kontekstów :

```
ARRAY [1..Rozmiar_alfabetu^Długość_kontekstu] OF
    ^Nagłówek_kontekstu
```

Tablica kontekstów dla danego kontekstu zwraca adres jego nagłówka (patrz niżej), który zawiera podstawowe informacje dotyczące tego kontekstu. Przy 256-elementowym alfabecie, już przy długości kontekstu równej 3 wielkość tej struktury osiągnęłaby 64 MB. Dlatego utrzymywana jest ona tylko dla kontekstów o długości 1 i 2. Z tego powodu jedynym sposobem odnalezienia nagłówków kontekstów o większej długości (jeżeli takowe w modelu występują) jest wykorzystanie powiązań istniejących w modelu PPM (patrz struktury komponentu predykcyjnego).

Nagłówek_kontekstu:

```
Model:      PPM.ModelKontekstu
Słownik:    LZ.Słownik
```

Nagłówek kontekstu składa się z dwóch części; omówimy je w kontekście komponentów, których dotyczą.

Komponent substytucyjny. Kluczową dla komponentu substytucyjnego strukturą danych jest słownik. Jego budowa jest następująca:

LZ.Słownik :

```
Aktualny_indeks:      BYTE
Element_słownika :    ARRAY [1..Rozmiar_słownika] OF POINTER
```

Aktualny_indeks oznacza numer elementu słownika, który zostanie zapisany bieżącą pozycją źródła w momencie jego aktualizacji. Jeżeli aktualny indeks przekroczy rozmiar słownika, redukowany jest do zera.

Element_słownika dla słowników rzędu wyższego niż 0 zawiera wskaźnik do pozycji źródła których kontekst jest identyczny z kontekstem słownika. Zastosowanie tej samej zasady do słownika rzędu 0 uczyniłoby go wspólnym narzędziem kompresji RLE – kolejne indeksy słownika odpowiadały by ciągom RLE o różnym okresie powtórzenia. Choć pewne typy danych obfitują w takie sekwencje, to jednak nie dotyczy to typów najbardziej popularnych – na przykład w plikach wykonywalnych występują w ograniczonej liczbie (tylko poza obszarem samego kodu programu),

w plikach tekstowych – sporadycznie (separatory liniowe, znakowe formatowanie tabeli, lub tak zwany *ASCII-art*).

Dlatego w proponowanym algorytmie wprowadzono dwa alternatywne sposoby wykorzystania słownika rzędu 0. Pierwszy z nich to użycie go jako pamięci podręcznej (*cache*) pozostałych słowników. Taki słownik oznaczać będziemy jako słownik rzędu 0c (*c* – *cache*; rząd takiego słownika wynosi nadal 0, określenie *słownik rzędu 0c* jest wygodnym w użyciu skrótem pojęciowym oznaczającym słownik typu *cache* rzędu 0). Aktualizacja takiego słownika następuje tylko po dokonaniu substytucji w oparciu o którykolwiek z pozostałych słowników. Przykładowo, podmiana frazy ‘CDEF’ ze słownika dla kontekstu ‘AB’ spowoduje umieszczenie w słowniku rzędu 0c frazy ‘ABCDEF’.

Drugi sposób wykorzystuje koncepcję „wtrąceń”. Załóżmy, że po frazie ‘ABCD-EF-GHIJ’ występuje fraza ‘ABCD-FE-GHIJ’. Frazy ‘EF’ i ‘FE’ traktujemy właśnie jako wtrącenia, która zaburzają zgodność frazy i powodują, że tylko fraza ‘ABCD’ będzie podlegać podmianie (zakładając, że ‘GHIJ’ po ‘FE’ pojawia się po raz pierwszy). Zauważmy jednak, że odległość pomiędzy dwoma wystąpieniami ‘ABCD’ jest identyczna z odległością pomiędzy dwoma wystąpieniami ‘GHIJ’. Jeżeli utrzymywać będziemy słownik różnicowy, zawierający podobnie jak w wersji 0c ostatnio wykonane podmiany, ale nie w postaci adresów bezwzględnych, a względnych (różnic z pozycją źródła), to podmiana ‘GHIJ’ stanie się możliwa. Taki słownik oznaczać będziemy jako słownik rzędu 0d (*d* – *dystans*).

Słownik rzędu 0d nie pozwala na wykrycie wtrąceń o różnej (w oryginale i kopii) długości. W tym celu należałoby, oprócz pozycji *j*, wyliczonej z pomniejszenia pozycji źródła *i* o pochodzącą ze słownika różnicę *d*, sprawdzić jej najbliższe otoczenie (*j-r*, *j+r*). Odrzucono ten pomysł, obawiając się poważnego ($2r$ -krotnego) spowolnienia algorytmu wyszukującego frazy zgodne, w stosunku do wyjściowej wersji 0d.

Komponent predykcyjny. Kluczową dla komponentu predykcyjnego strukturą danych jest model kontekstu. Jego budowa jest następująca:

```
PPM.ModelKontekstu:
    Suma:      WORD
    Liczba:    BYTE
    Rozmiar:   BYTE
    Ucieczki:  BYTE
    Liczniki:  ^PPM.Liczniki
    Rodzic:    ^Nagłówek_kontekstu
```

Odpowiednio: *Suma* zawiera sumę wszystkich liczników, *Liczba* – ile jest liczników niezerowych, *Rozmiar* – ile niezerowych liczników pomieści struktura *PPM.Liczniki* bez potrzeby jej realokacji, *Ucieczki* – licznik ucieczek z kontekstu,

Liczniki – wskaźnik do listy liczników symboli (patrz niżej), Rodzic – wskaźnik do nagłówka kontekstu krótszego o 1 (występuje tylko dla kontekstów o długości powyżej 3; krótsze konteksty ustalają kontekst nadrzędny poprzez tablicę kontekstów).

Struktura PPM.Liczniki występować może w dwóch postaciach. Jeżeli liczba niezerowych liczników jest mniejsza od przyjętej wartości krytycznej, występuje w postaci tablicy o dynamicznym rozmiarze:

```
PPM.Liczniki:
    ARRAY[1..Rozmiar] OF PPM.Licznik
    // gdzie
    PPM.Licznik:
        Symbol:      BYTE
        Licznik:     BYTE
        Dziecko:     ^Nagłówek_kontekstu
```

Licznik zawiera liczbę wystąpień Symbolu w bieżącym kontekście. Dziecko występuje tylko dla kontekstów o długości powyżej 1 i zawiera:

- dla kontekstów o długości poniżej rzędu modelu PPM, wskaźnik do nagłówka kontekstu podrzędnego (na przykład dla symbolu ‘E’ z kontekstu ‘ABCD’ będzie to ‘ABCDE’);
- dla kontekstów o długości równej rzędowi modelu PPM, wskaźnik do nagłówka kontekstu następującego (na przykład dla symbolu ‘F’ z kontekstu ‘ABCDE’ będzie to ‘BCDEF’).

Jeżeli liczba niezerowych liczników jest nie mniejsza od przyjętej wartości krytycznej, PPM.Liczniki występuje w postaci struktury, którą można określić jako drzewo binarne pozbawione dwóch przedostatnich poziomów:

```
PPM.Liczniki:
    Suma128:  ARRAY[0..0] OF WORD
    Suma64:   ARRAY[0..1] OF WORD
    Suma32:   ARRAY[0..3] OF WORD
    Suma16:   ARRAY[0..7] OF WORD
    Suma8:    ARRAY[0..15] OF WORD
    Licznik:  ARRAY[0..255] OF BYTE
    Dziecko:  ^Nagłówek_kontekstu
```

Licznik o indeksie s zawiera liczbę wystąpień w bieżącym kontekście symbolu s ; i -ty element SumyN zawiera liczbę wystąpień wszystkich symboli nie mniejszych niż $i \cdot N$ i mniejszych niż $(i + 0,5) \cdot N$ (ułatwia to wyliczanie skumulowanego prawdopodobieństwa).

Powodem użycia dwojaki struktury liczników jest powolność ustalania prawdopodobieństwa symbolu w oparciu o tablicę dla kontekstów o dużej liczbie niezerowych symboli i nieefektywność (głównie pod względem wymaganej pamięci, ale także szybkości) drzewa dla kontekstów o małej liczbie niezerowych symboli. Ilustruje to tabela 3.4. Interesującym rozwiązaniem wydaje się być też tablica

dwupoziomowa (16 grup po 16 symboli); testy praktyczne wykazały iż jest ona tylko nieznacznie wolniejsza od skróconego drzewa.

Tabela 3.4. Struktury danych dla kontekstu o $c = 256$ elementach

	Operacje				Wymagana pamięć (realnie, B)
	Odczyt			Zapis	
Struktura	Min	Średnio	Max		
Tablica niesortowana	1	128	256	1	512
Tablica posortowana ¹⁰	1	64	128	1	256
Tablica dwupoziomowa	2	16	32	2	320
Drzewo binarne	8			8	512
Skrócone (do 6) drzewo binarne	7	8	10	7	384

Źródło: opracowanie własne.

3.4.2 Algorytm przełączający

Prezentację algorytmu predykcyjno-substytucyjnego bez estymacji długości zapisu rozpoczyna jego najważniejszy element – główny algorytm przełączający. Stanowi on główną pętlę programu, a jego podstawowym zadaniem jest wybór odpowiedniej dla danej pozycji źródła metody kodowania. Dalej zostaną omówione zasady działania poszczególnych składników, do których algorytm przełączający odwołuje się.

Algorytm przełączający dla kompresji ma następującą postać:

```

PLZ.Kompresja:
  LZ.Inicjalizuj()
  PPM.Inicjalizuj()
  i:=1
  WHILE i <= rozmiar_źródła DO BEGIN
    Najdłuższa_fraza_zgodna:=LZ.Znajdź_najdłuższą_frazę_zgodną(i)
    IF Najdłuższa_fraza_zgodna >= Minimalna_długość_substytucji
    THEN BEGIN
      LZ.Zakoduj_pozycję_podmiany(i)
      LZ.Zakoduj_substytucję()
      INC (i, Najdłuższa_fraza_zgodna); END
    ELSE BEGIN
      PPM.Zakoduj_symbol(i)
      INC (i, 1); END
  END
  LZ.Zakoduj_pozycję_podmiany(i)
  LZ.Zakończ()
  PPM.Zakończ()

```

Jak należało się tego spodziewać, jedynym kryterium branym tu pod uwagę przy dokonywaniu wyboru metody jest minimalna długość substytucji. Wszystkie frazy,

¹⁰ Leksykograficznie, to jest w kolejności ustalonej przez wartości symboli.

które nie są od niej nie krótsze, podlegają podmianie. W przeciwnym wypadku pierwszy symbol frazy kodowany jest predykcyjnie.

Funkcje z przedrostkiem LZ (Inicjalizuj, Znajdź_najdłuższą_frazę_zgodną, Zakoduj_pozycję_podmiany, Zakoduj_substytucję, Zakończ) należą do komponentu substytucyjnego i zostaną omówione w punkcie 3.4.3. Funkcje z przedrostkiem PPM (Inicjalizuj, Zakoduj_symbol, Zakończ) należą do komponentu predykcyjnego i zostaną omówione w punkcie 3.4.4.

Algorytm przelączający dla dekompresji ma następującą postać:

```
PLZ.Dekompresja:
  LZ.Inicjalizuj()
  PPM.Inicjalizuj()
  i:=1
  Pozycja_podmiany:=LZ.Odkoduj_pozycję_podmiany()
  WHILE i <= rozmiar_źródła DO BEGIN
    IF i = Pozycja_podmiany THEN BEGIN
      Długość_frazy:=LZ.Odkoduj_substytucję(i)
      INC (i, Długość_frazy);
      Pozycja_podmiany:=LZ.Odkoduj_pozycję_podmiany(i)
    END ELSE BEGIN
      LZ.Uaktualnij_słownik(i)
      PPM.Odkoduj_symbol(i)
      INC (i, 1); END
  END
```

Dekompresja wykonywana jest szybciej, gdyż w jej przypadku nie ma potrzeby wyszukiwania najdłuższej frazy zgodnej. Warto zwrócić uwagi jest wykonywane z wyprzedzeniem ustalenie pozycji podmiany: musi być ona bowiem znana dekoderni, zanim zostanie osiągnięta.

Ponownie, funkcje z przedrostkiem LZ (Inicjalizuj, Odkoduj_pozycję_podmiany, Odkoduj_substytucję, Uaktualnij_słownik) należą do komponentu substytucyjnego i omówione zostaną w punkcie 3.4.3.

Funkcje z przedrostkiem PPM (Inicjalizuj, Odkoduj_symbol) przynależą do komponentu predykcyjnego i omówione zostaną w punkcie 3.4.4.

Koder statystyczny. Wszystkie operacje kodowania w proponowanym algorytmie na najniższym poziomie realizowane są przy pomocy implementacji koda przedziałów [SUBB99], tak jak został on opisany w punkcie 2.1.3.

Podsystem zarządzania pamięcią. Choć prezentowany algorytm może współdziałać z zewnętrznym (na przykład systemowym) menadżerem pamięci, to lepszym rozwiązaniem jest użycie własnego. Po pierwsze, przed rozpoczęciem działania algorytmu alokowana jest poprzez system sterujący, z której wewnętrzny podsystem zarządzania pamięcią przydziela bloki na potrzeby algorytmu. W ten sposób przyspieszone zostaje uwolnienie pamięci po zakończeniu działania, jako że sprowadza się jedynie do dealokacji całej sterty.

Po drugie, ze względu na to, że dealokowane przez komponent PPM bloki pamięci mają tylko kilka możliwych rozmiarów, ponowny przydział zwolnionej pamięci może być w prosty sposób zrealizowany poprzez prowadzenie tablic zawierających zwolnione bloki o określonym rozmiarze. Przyspiesza to przydział pamięci.

Po trzecie, własny moduł przydziału pamięci pozwala na lepszą jego integrację z resztą algorytmu. W istocie, jeżeli podsystem ten nie jest w stanie z braku wolnej pamięci zrealizować żądanej alokacji, wykonuje restart modelu PPM. Polega on na porzuceniu statystyk wszystkich rzędów większych od 0.

3.4.3 Komponent substytucyjny

Inicjalizacja komponentu substytucyjnego (LZ.Inicjalizuj). Przed rozpoczęciem działania, inicjalizacji wymagają następujące elementy komponentu substytucyjnego:

- słowniki rzędu 0 (słowniki wyższych rzędów inicjalizowane są w momencie tworzenia struktur odpowiadających im kontekstów, patrz 3.4.4);
- modele statystyczne pozycji i długości podmiany;
- instancja kodera przedziałów przydzielona komponentowi substytucyjnemu.

W przypadku słownika rzędu 0d, inicjalizacja polega na wypełnieniu go kolejnymi liczbami naturalnymi (1, 2, 3, ...) reprezentującymi odwołania do fraz odległych odpowiednio o 1, 2, 3, ... symbole. Liczba elementów słownika rzędu 0c jest zerowana.

Inicjalizacja modeli statystycznych pozycji i długości podmiany polega na przypisaniu wszystkim możliwym wartościom niezerowych prawdopodobieństw. Licznikom dla większych pozycji i długości nadaje się mniejsze wartości początkowe, co uzasadnione jest lokalnością referencji; ze względu na dużą szybkość aktualizacji tych modeli ma to znaczenie dla efektów kompresji w zasadzie tylko w przypadku zbiorów danych o niewielkiej długości.

Wyszukiwanie frazy w słowniku (LZ.Znajdź_najdłuższą_frazę_zgodną). Jego celem jest zwrócenie: indeksu najdłuższej frazy zgodnej dla aktualnej pozycji źródła, długości frazy zgodnej oraz identyfikatora słownika, w którym frazę tę znaleziono.

```
LZ.Znajdź_najdłuższą_frazę_zgodną(i: pozycja_źródła):
    Najdłuższa_fraza_zgodna:= Minimalna_długość_substytucji-1
    Rząd_słownika:= Maksymalny_dostępny_rząd_słownika
    WHILE Rząd_słownika >= 0 DO BEGIN
        Słownik:= Znajdź_słownik (Rząd_słownika, i)
        Przeszukaj_słownik (Słownik,i)
        Aktualizuj_słownik (Słownik,i)
        DEC(Rząd_słownika);      END
```

Funkcja `Znajdź_słownik` określa słownik właściwy dla rzędu słownika i kontekstu dla pozycji źródła *i*. Funkcja `Aktualizuj_słownik` uzupełnia słownik o wpis dotyczący pozycji źródła *i*. Wyszukanie frazy zgodnej realizowane jest w poniższy sposób:

```
Przeszukaj_słownik (Słownik, Kopia):
    Indeks:= 0
    WHILE Indeks < Rozmiar_słownika DO BEGIN
        INC(Indeks)
        Oryginał:= Słownik.Element_słownika[Indeks]
        IF Oryginał = NIL THEN BREAK          // nowy element słownika
        IF Oryginał >= Kopia THEN CONTINUE    // zbyt stary element
        IF (Oryginał+Najdłuższa_fraza_zgodna)^
           = (Kopia+Najdłuższa_fraza_zgodna)^
        THEN BEGIN
            Fraza_zgodna:= Takie_same(Oryginał, Kopia)
            IF Fraza_zgodna < Najdłuższa_fraza_zgodna THEN
                CONTINUE
            Najdłuższa_fraza_zgodna:= Fraza_zgodna
            NFZ_Słownik:= Słownik
            NFZ_Indeks:= Indeks;      END
    END
```

Funkcja `Takie_same` zwraca dla pozycji *Kopia* długość frazy zgodnej na pozycji *Oryginał*. Zauważmy, że wywołana jest ona tylko wtedy, gdy kolejne cztery symbole na pozycji po określonej przez dotychczasową *Najdłuższą_frazę_zgodną* są identyczne, co ma na celu skrócenie czasu wyszukiwania w przypadku gdy wiele fraz ma identyczne fragmenty początkowe.

Kodowanie przełącznika (`LZ.Zakoduj_pozycję_podmiany`). Istnieją trzy zasadnicze sposoby zapisu przełącznika:

- przy użyciu istniejących modeli, jako dodatkowy symbol alfabetu,
- niezależnie w osobnym modelu,
- poprzez zapis odległości pomiędzy przełączeniami.

Sposób pierwszy jest najszybszy w działaniu, jednak nieefektywny z powodu problemu z zebraniem dostatecznej liczby statystyk. Dla wielu kontekstów liczba podmian będzie znikoma; przydzielenie przełącznikowi nawet jednostkowego licznika w modelu kontekstu, którego suma liczników nie przekracza kilku oznaczała by istotne wydłużenie słów kodowych dla pozostałych symboli. Ponadto, sposób ten nie pozwalałby na wykorzystanie redundancji pozycji przełączania – rozkład odległości pomiędzy przełączeniami jest bowiem nierównomierny.

Drugi ze sposobów pozwala na użycie do zakodowania przełącznika dowolnego modelu statystycznego, a zatem jest efektywny. Jego słabą stroną jest szybkość – wymaga bowiem nie tylko zakodowania faktu wystąpienia przełącznika, ale również faktu jego niewystąpienia – przed każdym symbolem kodowanym PPM.

Trzeci ze sposobów jest prawie tak samo szybki jak sposób pierwszy, i prawie tak samo efektywny jak sposób drugi. Jego wadą jest to, że wymaga użycia dwóch strumieni wyjściowych. Bierze się to stąd, że odległość pomiędzy przełączeniami znana jest koderowi dopiero po dotarciu do drugiego z przełączeń, dekodery natomiast wymaga jej już po dotarciu do pierwszego z przełączeń.

W proponowanym algorytmie zastosowano sposób trzeci. Problem połączenia dwóch strumieni wyjściowych (dla LZ i PPM) rozwiązano poprzez przeplot w jednym strumieniu wyjściowym bloków słów kodowych. Blok taki generowany jest na wyjściu w momencie, gdy bufor strumienia wyjściowego odpowiedniego typu został przepełniony. Wymaga to od dekodera odnalezienia w strumieniu odpowiedniego bloku przed jego załadowaniem, co nieznacznie wydłuża czas dekompresji w przypadku, gdy bloki obu typów nie są ułożone naprzemiennie. Alternatywne rozwiązanie polegające na przeplocie pojedynczych bajtów pochodzących z obu strumieni nie miało by tej wady, jednak wymagałoby (w sytuacji ograniczonego rozmiaru bufora strumienia wyjściowego) dodatkowego przejścia po danych skompresowanych, co znacznie wydłużyłoby czas kompresji.

```
LZ.Zakoduj_pozycję_podmiany(i: pozycja podmiany):
    Odległość:= i-Poprzednia_pozycja_podmiany
    Nowa_pozycja_podmiany:=i+Najdłuższa_fraza_zgodna
    Wyklucz_symbol(Poprzednia_pozycja_podmiany,Nowa_pozycja_podmiany)
    Poprzednia_pozycja_podmiany:=Nowa_pozycja_podmiany
    WHILE Odległość >= Limit_Odległości DO BEGIN
        DEC(Odległość, Limit_Odległości)
        Zakoduj_odległość(Limit_Odległości + (Odległość AND 1))
        Odległość:= Odległość SHR 1;      END
    Zakoduj_odległość(Odległość)
```

Funkcja `Zakoduj_odległość` koduje przekazaną jej wartość używając szybko adaptującego się modelu statystycznego rzędu 1. Optymalną wartość `Limitu_Odległości` oszacowano eksperymentalnie na 8.

Funkcja `Wyklucz_symbol` ustawia w tablicy wykluczeń wartość dla symbolu źródła numer `Poprzednia_pozycja_podmiany` na `Nowa_pozycja_podmiany` (patrz 3.4.4). Wiadomo bowiem, że symbol ten nie może wystąpić (gdyby wystąpił, `Najdłuższa_fraza_zgodna` musiałaby go obejmować).

Kodowanie substytucji (`LZ.Zakoduj_substytucję`). Ta część algorytmu ma na celu zakodowanie identyfikatora słownika, indeksu pozycji słownika oraz długości frazy. Obserwacje przeprowadzone na bazie rezultatów uzyskanych dla różnego rodzaju plików testowych wskazują na to, że:

- długość frazy skorelowana jest z rzędem słownika,

- dla większości typowych plików wartości indeksu pozycji słownika (i ich przyrosty) mają rozkład jednostajny.

Opierając się na powyższych obserwacjach, w proponowanym algorytmie długość frazy i identyfikator słownika łączone są ze sobą i kodowane jako pojedyncza wartość, natomiast indeks pozycji słownika kodowany jest przy pomocy symetrycznego modelu Bernoulliego.

```
LZ.Zakoduj_substytucję():
    Długość:=Najdłuższa_fraza_zgodna * Liczba_rodzajów_słowników
    INC (Długość, Identyfikator(NFZ_Słownik))
    WHILE Długość >= Limit_Długości DO BEGIN
        DEC(Długość, Limit_Długości)
        Zakoduj_Długość (Limit_Długości + (Długość AND 1))
        Długość:= Długość SHR 1;    END
    Zakoduj_Długość(Długość)
    RC.Zakoduj (NFZ_Indeks, 1, Rozmiar (NFZ_Słownik))
```

Funkcje Identyfikator i Rozmiar zwracają odpowiednio: niepowtarzalną wartość identyfikującą słownik (mniejszą niż Liczba_rodzajów_słowników) i liczbę ważnych (bez niewykorzystanych i niedostępnych) pozycji słownika. Funkcja Zakoduj_długość koduje przekazaną jej wartość używając szybko adaptującego się modelu statystycznego rzędu 1. Optymalną wartość limitu_długości oszacowano eksperymentalnie na 16.

Deinicjalizacja komponentu substytucyjnego (LZ.Zakończ). Jedynym zadaniem tej funkcji jest przepisanie wartości identyfikatora z kodera przedziałów LZ do strumienia wyjściowego.

Jako że wszystkie najważniejsze elementy algorytmu kompresującego zostały już omówione, przejdziemy teraz do funkcji występujących jedynie w algorytmie dekompresującym.

Dekodowanie przełącznika (LZ.Odkoduj_pozycję_podmiany) jest czynnością odwrotną do opisanej w przypadku kompresji:

```
LZ.Odkoduj_pozycję_podmiany(i: pozycja_źródła):
    Odległość:= 0
    Skala:= 0
    REPEAT
        Kod:= Odkoduj_odległość()
        IF Kod < Limit_Odległości THEN BREAK
        INC(Odległość, Kod SHL Skala)
        INC(Skala, 1)
        IF Skala > MaxSkala THEN HALT // Przekłamanie w danych
    UNTIL FALSE
    Kod:= Kod SHL Skala
    INC(Odległość, Kod)
    Pozycja_podmiany:= i+Odległość
```

Funkcja `Odkoduj_odległość` zwraca wartość uzyskaną z modelu statystycznego identycznego z użytym przy kodowaniu odległości. Wartość `MaxSkala` wynosi $\lg(\text{maksymalny dopuszczalny rozmiar pliku})$.

Dekodowanie substytucji (`LZ.Odkoduj_substytucję`). Ta część algorytmu ma na celu odekodowanie identyfikatora słownika, indeksu pozycji słownika oraz długości frazy.

```
LZ.Odkoduj_substytucję(i: pozycja_źródła):
    Długość:=0
    Skala:=0
    REPEAT
        Kod:= Odkoduj_długość ()
        IF Kod < Limit_Długości THEN BREAK
        INC(Długość, Kod SHL Skala)
        INC(Skala,1)
        IF Skala > MaxSkala THEN HALT // Przekłamanie w danych
    UNTIL FALSE
    Kod:= Kod SHL Skala
    INC(Długość,Kod)
    ID_Słownika:=Długość MOD Liczba_rodzajów_słowników
    Długość:= Długość DIV Liczba_rodzajów_słowników
    Element_Słownika:=RC.PobierzQ(Rozmiar(ID_Słownika))
    RC.Odkoduj(Element_Słownika,1)
    Oryginał:=Ustal_frazę(ID_Słownika, Element_Słownika)
    WHILE Długość > 0 DO BEGIN
        (i + Długość)^:=(Oryginał + Długość)^
        DEC (Długość)
    END
```

Funkcja `Odkoduj_długość` zwraca wartość uzyskaną z modelu statystycznego identycznego z użytym przy kodowaniu odległości. Wartość `MaxSkala` wynosi $\lg(\text{maksymalny dopuszczalny rozmiar pliku})$. Funkcja `Rozmiar` zwraca faktyczny rozmiar słownika o podanym `ID_Słownika`. Funkcja `Ustal_frazę` zwraca wskaźnik do początku frazy znajdującej się na pozycji `Element_Słownika` w słowniku określonym przez `ID_Słownika`.

Dodanie frazy do słownika (`LZ.Uaktualnij_słownik`). Zadaniem tej funkcji jest dodanie frazy znajdującej się na bieżącej pozycji źródła do słowników rzędu powyżej 0.

3.4.4 Komponent predykcyjny

Inicjalizacja komponentu predykcyjnego (`PPM.Inicjalizuj`). Przed rozpoczęciem działania, inicjalizacji wymagają następujące elementy komponentu predykcyjnego:

- modele statystyczne rzędu -1 i 0 (modele wyższych rzędów inicjalizowane są w momencie tworzenia struktur odpowiadających im kontekstów, patrz dalej) i drugorzędnej estymacji ucieczek;

- instancja koder przedziałów przydzielona komponentowi predykcyjnemu;
- zmienna `Aktualny_model` wskazuje na model rzędu -1 .

Inicjalizacja modeli statystycznych rzędu -1 i 0 polega na przypisaniu wszystkim symbolom alfabetu prawdopodobieństwa $1/256$ w modelu rzędu -1 oraz prawdopodobieństwa 0 w modelu rzędu 0 .

W przypadku inicjalizacji modelu drugorzędnej estymacji ucieczek, początkowe prawdopodobieństwa symbolu wyjścia ustala się na wartości wyliczone według zasad algorytmu PPMC.

Kodowanie symbolu (`PPM.Zakoduj_symbol`). Kluczowy element komponentu predykcyjnego oparty jest na algorytmie PPMC, ulepszony autorskimi implementacjami technik lokalnej estymacji rzędu i drugorzędnej estymacji ucieczek.

```
LZ.Zakoduj_symbol(i: pozycja symbolu):
    Symbol:=i^
    WHILE TRUE DO BEGIN
        Indeks:=Znajdź_symbol(Aktualny_model, Symbol)
        IF Indeks = Aktualny_model.Liczba THEN BEGIN
            IF Indeks = Aktualny_model.Rozmiar THEN
                Powiększ_model(Aktualny_model)
            IF Model_wiarygodny(Aktualny_model) THEN BEGIN
                Wyklucz_symbole(Aktualny_model, i)
                IF Aktualny_model.Suma < WyłączSEE THEN
                    SEE.Zakoduj_ucieczkę(Aktualny_model)
                ELSE Zakoduj_ucieczkę(Aktualny_model)
            END
            Dodaj_symbol(Aktualny_model, Symbol)
            Aktualny_model:=Rodzic(Aktualny_model)
            CONTINUE;    END
        Aktualizuj(Aktualny_model, Symbol)
        IF NOT Model_wiarygodny(Aktualny_model) THEN BEGIN
            Aktualny_model:=Rodzic(Aktualny_model)
            CONTINUE;    END
        IF Aktualny_model.Suma < WyłączSEE THEN
            SEE.Zakoduj_brak_ucieczki(Aktualny_model)
        Aktualny_model:=Dziecko(Aktualny_model, Symbol)
        BREAK;    END
```

Funkcja `Znajdź_symbol` określa pozycję Symbolu w `Aktualnym_modelu`. Jednocześnie wylicza wymagane jako parametry koder przedziałów wartości Q_{sk} , Q_s i Q_t . Wartości te nie obejmują liczników tych symboli dla których w tablicy wykluczeń figuruje wartość odpowiadająca pozycji symbolu i .

`Powiększ_model` relokuje tablicę liczników `Aktualnego_modelu` do obszaru pamięci o większym rozmiarze. Przyrost rozmiaru jest parametrem decydującym między zapotrzebowaniem na pamięć a szybkością działania. Szybki przyrost oznacza większe zapotrzebowanie na pamięć (tylko część nowo przydzielonego obszaru jest wykorzystana). Wolny przyrost oznacza wolniejsze działanie (wymaga częstszych relokacji). W proponowanej implementacji, zwiększenie rozmiaru jest dwukrotne.

Funkcja `Model_wiarygodny` implementuje lokalną estymację rzędu. Jeżeli dla danego rzędu modelu liczba dotychczasowych trafionych predykcji nie przekracza założonej stałej (w oparciu o wyniki eksperymentów nadano jej wartość 0), rząd ten jest automatycznie redukowany (funkcja zwraca wartość logiczną 'fałsz').

`Wyklucz_symbole` ustala wartości w tablicy wykluczeń na i dla wszystkich symboli o niezerowych licznikach w `Aktualnym_modelu`.

Drugorzędna estymacja ucieczek (SEE) uwzględniana jest tylko dla modeli uznanych za podatne na błędy w szacowaniu prawdopodobieństwa ucieczki. Model uznany jest za taki, gdy suma jego liczników nie przekracza założonej wartości progowej `WyłączSEE`. Im wyższa wartość `WyłączSEE`, tym dokładniejsza estymacja prawdopodobieństwa ucieczki i wolniejsze działanie; im niższa, tym mniej dokładna estymacja prawdopodobieństwa ucieczki i szybsze działanie (w oparciu o wyniki eksperymentów przyjęto wartość 24).

`SEE.Zakoduj_ucieczkę` koduje symbol ucieczki przy użyciu modelu SEE i zwiększa licznik ucieczek. Kontekst dla ucieczki w proponowanym modelu SEE obejmuje: aktualną długość kontekstu, wartość funkcji $\text{sgn}(\text{aktualna długość kontekstu} - \text{poprzednia długość kontekstu})$, liczbę niezerowych liczników aktualnego modelu i ich sumę. `SEE.Zakoduj_brak_ucieczki` jest analogiczną funkcją używaną w przypadku, gdy ucieczka nie nastąpiła.

`Zakoduj_ucieczkę` koduje symbol ucieczki przy użyciu PPMC i zwiększa licznik ucieczek.

`Dodaj_symbol` dodaje do modelu i inicjalizuje licznik symbolu dotąd w nim nie występującego.

Funkcja `Rodzic` zwraca model dla kontekstu podrzędnego w oparciu o pole modelu kontekstu `Rodzic` lub tablicę kontekstów, gdy ono nie występuje. Analogicznie, funkcja `Dziecko` zwraca model dla kontekstu następującego w oparciu o pole `Dziecko` lub tablicę kontekstów, gdy ono nie występuje. Jeżeli znaleziony model dotąd nie istniał, jest inicjalizowany.

Funkcja `Aktualizuj` realizuje trzy zadania:

- zwiększa licznik `Symbolu` w `Aktualnym_modelu` o stałą inkrementacji,
- jeżeli licznik ten przekroczył okres skalowania, wykonuje skalowanie wszystkich liczników modelu i uaktualnia sumę liczników,
- wykonuje sortowanie liczników modelu, przesuwając `Symbol` o jedną pozycję tablicy do przodu tak długo jak jego licznik ma wartość wyższą od licznika symbolu poprzedzającego.

Sortowanie liczników wykonywane jest oczywiście tylko dla liczników w postaci tablicy. Ma ono na celu przyspieszenie wykonywania operacji wyszukiwania symbolu. W przypadku, gdy redundancja danych jest duża, liczba niezerowych liczników jest mała, a ich rozkład jest często eksponentyjalny. Sortowanie trwa zatem krótko, a uzyskane w jego wyniku przyspieszenie wyszukiwania jest duże. W przypadku, gdy redundancja danych jest mała, liczba niezerowych liczników jest duża, a ich rozkład jest często jednostajny. Sortowanie trwa zatem długo, a uzyskane w jego wyniku przyspieszenie wyszukiwania jest małe. Aby uniknąć sortowania w sytuacji, gdy jest ono nieefektywne, wprowadzono ograniczenie sortowania: przesuwanie symbolu do przodu tablicy rozpocznie się tylko wtedy, gdy różnica wartości jego licznika z wartością licznika symbolu poprzedzającego przekroczy założoną stałą `WłączSortowanie` (w oparciu o wyniki eksperymentów nadano jej wartość 1).

Deinicjalizacja komponentu predykcyjnego (`PPM.Zakończ`). Jedynym zadaniem tej funkcji jest przepisanie wartości identyfikatora z kodera przedziałów PPM do strumienia wyjściowego.

Jako że wszystkie najważniejsze elementy algorytmu kompresującego zostały już omówione, przejdziemy teraz do funkcji występujących jedynie w algorytmie dekompresującym.

Dekodowanie symbolu (`PPM.Odkoduj_symbol`) jest czynnością odwrotną do opisanej w przypadku kompresji:

```
LZ.Odkoduj_symbol(i: pozycja symbolu):
  Poprzedni_model:=NIL
  WHILE TRUE DO BEGIN
    IF NOT Model_wiarygodny(Aktualny_model) THEN BEGIN
      Aktualny_model:=Rodzic(Aktualny_model)
      CONTINUE;    END
    IF Aktualny_model.Suma < WyłączSEE THEN
      Symbol:=SEE.Odkoduj_symbol(Aktualny_model)
    ELSE
      Symbol:=Odkoduj_symbol(Aktualny_model)
    IF Symbol = Ucieczka THEN BEGIN
      IF Aktualny_model.Liczba = Aktualny_model.Rozmiar THEN
        Powiększ_model(Aktualny_model)
      Wyklucz_symbole(Aktualny_model,i)
      Poprzedni_model:=Aktualny_model
      Aktualny_model:=Rodzic(Aktualny_model)
      CONTINUE;    END
    Dodaj_symbol(Poprzedni_model,Symbol)
    Aktualizuj(Aktualny_model,Symbol)
    Aktualny_model:=Dziecko(Aktualny_model,Symbol)
  BREAK;END
```

Funkcje `Model_wiarygodny`, `Rodzic`, `Dziecko`, `Powiększ_model`, `Wyklucz_symbole`, `Dodaj_symbol` i `Aktualizuj` zostały już opisane przy okazji opisu algorytmu kompresującego. `Model_wiarygodny` zwraca wartość logiczną

‘fałsz’, gdy liczba dotychczasowych udanych predykcji z użyciem aktualnego modelu nie przekracza założonej stałej. Rodzic zwraca model dla kontekstu podrzędnego. Dziecko zwraca model dla kontekstu następującego; jeżeli taki model dotąd nie istniał, jest inicjalizowany. Powiększ_model relokuje tablicę liczników Aktualnego_modelu do obszaru pamięci o większym rozmiarze. Wyklucz_symbole ustala wartości w tablicy wykluczeń na i dla wszystkich symboli o niezerowych licznikach w Aktualnym_modelu. Dodaj_symbol dodaje do modelu licznik nowego (w tym modelu) symbolu i go inicjalizuje. Aktualizuj zwiększa licznik Symbolu w Aktualnym_modelu o stałą inkrementacji (a jeżeli licznik ten przekroczył okres skalowania, wykonuje skalowanie wszystkich liczników modelu i uaktualnia sumę liczników) i wykonuje sortowanie liczników modelu, przesuując Symbol o jedną pozycję tablicy do przodu tak długo jak jego licznik ma wartość wyższą od licznika symbolu poprzedzającego.

Przez Ucieczkę należy rozumieć specjalny symbol wyjścia (nie będący elementem alfabetu źródła).

SEE.Odkoduj_symbol dekoduje symbol w dwuetapowym procesie, w którym najpierw dekodowany jest znacznik SEE określający czy nastąpiła ucieczka. W przypadku gdy nie miało to miejsca, dekodowany jest sam symbol.

Odkoduj_symbol dekoduje symbol w jednoetapowym procesie; w tym przypadku, klasycznym dla PPM sposobem symbol wyjścia reprezentowany jest przedziałem wartości wydzielonym z prawdopodobieństw pozostałych symboli uwzględnionych w modelu.

Obie powyższe funkcje odpowiadają za przeprowadzenie aktualizacji licznika ucieczek modelu. W przypadku wersji SEE, także liczników modelu SEE.

3.4.5 Parametry algorytmu

Do podstawowych parametrów przedstawionego algorytmu zaliczamy: (LZ:) maksymalną wielkość bufora słownikowego, minimalną długość substytucji, wielkości słowników poszczególnych rzędów oraz (PPM:) maksymalną wielkość i rząd modelu PPM. Parametry te pozostawiono do wyboru użytkownikowi, umożliwiając mu w ten sposób kontrolę nad wymaganiami pamięciowymi i szybkością działania.

Maksymalna wielkość bufora słownikowego (bufora wejścia) decyduje o tym jak długo znajdujące się w słownikach frazy mogą zostać podmienione. Po przetworzeniu zawartości bufora słownikowego, zostaje on zapisany kolejną porcją danych. Odwołania do fraz, które zostały nadpisane nowymi danymi nie będą mogły zostać zrealizowane. Parametr ten ma oczywisty wpływ na wielkość zapotrzebowania

na pamięć, a ponadto, w mniej jednoznaczny sposób, wpływa na szybkość i efektywność kompresji. Mniejsza liczba dostępnych elementów zmniejsza czas przeszukiwania słownika i długość kodu indeksu pozycji słownika. Z drugiej strony zmniejsza liczbę możliwych podmian i ich długość.

Minimalna długość substytucji. Określa minimalną długość frazy, dla której dokonana zostanie substytucja. Im niższa, tym szybciej odbywa się kompresja i dekompresja. Z kolei efektywność kompresji najwyższa jest dla pewnej charakterystycznej dla każdego zbioru danych wartości, która zależy od przeciętnej efektywności komponentu predykcyjnego. Wraz z oddalaniem się minimalnej długości substytucji od tej optymalnej wartości, spada efektywność kompresji.

Wielkości słowników poszczególnych rzędów. Po pierwsze, parametr ten, a w szczególności jego wartość dla słowników wyższych rzędów, wpływa na wielkość zapotrzebowania na pamięć. Zwiększanie rozmiarów słowników powoduje spadek szybkości ich przeszukiwania, z drugiej strony zbyt małe słowniki zmniejszają liczbę podmian, a co za tym idzie całkowitą szybkość działania. Wielkość słownika określonego rzędu ma wpływ na efektywność kompresji zależny od charakterystyki kompresowanego zbioru danych. Zwykle zwiększanie jej powoduje wzrost efektywności.

Maksymalna wielkość modelu PPM. Określa częstotliwość restartu modelu. Wyznacza górną granicę zapotrzebowania na pamięć. Dla plików homogenicznych, większy model daje zawsze lepszą efektywność kompresji. Dla plików heterogenicznych, restarty modelu mogą poprawić efektywność.

Rząd modelu PPM. Im wyższy, tym większe zapotrzebowanie na pamięć i niższa szybkość komponentu predykcyjnego. Efektywność kompresji zwykle rośnie ze wzrostem rzędu modelu, choć zależy to od charakterystyki danych.

Drugorzędne parametry algorytmu. Stanowią je przede wszystkim częstotliwości skalowania. Podobnie jak parametry pierwszorzędne, wpływają one na efektywność i szybkość kompresji, jednakowoż ich wpływ jest dalece mniejszy – pod warunkiem, że nie nada się im wartości skrajnie patologicznych. W odróżnieniu od wymienionych wyżej parametrów pierwszorzędnych, parametry drugorzędne nie są jawne dla użytkownika końcowego – inaczej niepotrzebnie komplikowałoby to obsługę narzędzi kompresji. W związku z tym nadane zostały im stałe wartości oparte na wynikach uzyskanych na testowanych plikach różnego typu.

3.4.6 Podstawowe właściwości algorytmu

Przedstawiony powyżej algorytm stosuje zasadę wyłączania symboli w ograniczonym zakresie. Mianowicie, wyłączaniu z modelu PPM podlega jedynie pojedynczy symbol wskazywany przez koniec najdłuższej frazy zgodnej po wykonaniu substytucji. Pełna implementacja tej zasady byłaby zbyt czasochłonna w stosunku do osiąganego wzrostu efektywności (problem ten opisano w punkcie 3.1.5).

Algorytm bez estymacji długości zapisu stosuje jednostronnie zasadę wyłączania aktualizacji. W modelu PPM odnotowywane są tylko te symbole, które nie zostały podmienione. Natomiast model LZ rejestruje również frazy, których pierwszy symbol kodowany jest predykcyjnie. Przy czym model LZ nie rejestruje fraz znajdujących się wewnątrz frazy podmienianej.

Model PPM utrzymywany jest w stanie spójnym. Upraszcza to procedurę wykluczeń: dla każdego kontekstu, poza równym rzędowi modelu, pełną listę wykluczeń stanowi lista niezerowych symboli kontekstu poprzedzającego.

Złożoność obliczeniowa algorytmu. Złożoność obliczeniowa przełącznika jest liniowa w czasie: $O(n)$. Złożoność obliczeniowa komponentu substytucyjnego:

- W przypadku nie odnalezienia frazy zgodnej o długości osiągającej minimalną długość substytucji (l_{min}): $O(|D| \cdot l_{min} \cdot n)$, gdzie $|D|$ oznacza sumę wszystkich rozmiarów słowników. Jako że parametry użytkownika $|D|$ i l_{min} pozostają w czasie niezmiennie, otrzymujemy złożoność $O(n)$.
- W przypadku odnalezienia frazy zgodnej o długości l osiągającej minimalną długość substytucji: $O(|D| \cdot n)$. Jako że parametr użytkownika $|D|$ pozostaje w czasie niezmienny, otrzymujemy złożoność $O(n)$.

Złożoność obliczeniowa komponentu predykcyjnego dla rzędu modelu PPM K i wielkości alfabetu $|A|$ wynosi $O(K \cdot |A| \cdot n)$. Jako że wielkość alfabetu jest trwałą własnością modelu, a parametr użytkownika K pozostaje w czasie niezmienny, otrzymujemy złożoność $O(n)$.

Złożoność obliczeniowa całego algorytmu wyniesie zatem $O(n) + O(|D| \cdot l_{min} \cdot n) + O(K \cdot |A| \cdot n)$, po uproszczeniu $O((|D| \cdot l_{min} + K \cdot |A|) \cdot n)$, co przy niezmiennych w czasie $|D|$, l_{min} , $|A|$ i K daje $O(n)$.

Złożoność pamięciowa algorytmu. Struktury danych możemy podzielić na dwie grupy:

- te, których wielkość nie zależy od długości zbioru danych: bufory wejścia i wyjścia, tablice nagłówek, modele i słowniki rzędu 0 i -1 ; ich wielkość jest niezmienna w czasie, więc złożoność pamięciową określimy jako $O(1)$;

- te, których wielkość zależy od długości zbioru danych: modele i słowniki wyższych rzędów; zauważmy że maksymalna liczba możliwych w zbiorze danych kontekstów określonego rzędu nie może przekroczyć jego długości; ich złożoność pamięciową określimy zatem jako $O(n)$.

Złożoność pamięciowa całego algorytmu wyniesie zatem: $O(n)$.

Długość po zakodowaniu. Oznaczmy przez L_{PPM1} maksymalną teoretyczną długość po zakodowaniu metodą predykcijną pojedynczego symbolu źródła:

$$L_{PPM1} = -(K+1)\lg P_{\min}, \quad (3.32)$$

gdzie: $P_{\min}$ – minimalne dopuszczalne prawdopodobieństwo (dane wzorem 2.4), K – rząd modelu PPM.

Przez L_{LZ1} oznaczmy maksymalną teoretyczną długość po zakodowaniu metodą substytucyjną pojedynczej frazy źródła:

$$L_{LZ1} = \left(5 - \max \left(4, \lg \frac{n}{|G|} \right) - \max \left(3, \lg \frac{n}{|G|} \right) \right) \lg P_{\min} + \lg |D_{\max}|, \quad (3.33)$$

gdzie: n – liczba danych w zbiorze wejściowym, G – zbiór pozycji źródła dla których szukano i znaleziono frazę zgodną o długości osiagającej minimalną długość substytucji, $D_{\max}$ – rozmiar największego słownika.

Dla proponowanego algorytmu maksymalna długość zbioru danych po zakodowaniu $L_{\max}$ nie przekroczy:

$$L_{\max} = \gamma \sum_{i=1}^n \begin{cases} L_{PPM1}, & i \in F, \\ L_{LZ1}, & i \in G, \\ 0, & i \notin F \cup G, \end{cases} \quad (3.34)$$

gdzie F oznacza zbiór pozycji źródła dla których szukano, ale nie znaleziono frazy zgodnej o długości osiagającej minimalną długość substytucji, a G – zbiór pozycji źródła dla których szukano i znaleziono frazę zgodną o długości osiagającej minimalną długość substytucji.

3.5 Algorytm predykcijno-substytucyjny z estymacją długości zapisu

3.5.1 Algorytm przełączający

Algorytm wykonujący estymację długości zapisu z algorytmem bez estymacji długości zapisu różni przede wszystkim algorytm przełączający. Ponadto, pewne dodatkowe funkcje zostały nałożone na koder statystyczny. Poniżej omówione zostaną

tylko te elementy algorytmu, które uległy zmianie w stosunku do wersji przedstawionej w poprzednim podrozdziale.

Algorytm przełączający w oparciu o estymację długości zapisu ma dla kompresji następującą postać:

```
PLZ.Kompresja:
  LZ.Inicjalizuj()
  PPM.Inicjalizuj()
  i:=1
  Koniec_substytucji:=0
  WHILE i <= rozmiar_źródła DO
    IF i >= Koniec_substytucji THEN BEGIN
      IF i = Koniec_substytucji THEN BEGIN
        IF Długość_LZ() <= Długość_PPM() THEN BEGIN
          RC.Odtwórz_stan_kodera()
          LZ.Zakoduj_pozycję_podmiany(i)
          LZ.Zakoduj_substytucję(); END
        END
        Najdłuższa_fraza_zgodna:=LZ.Znajdź_najdłuższą_frazę_zgodną(i)
        IF Najdłuższa_fraza_zgodna >= Minimalna_długość_substytucji
        THEN
          IF Najdłuższa_fraza_zgodna >=
            Zadowalająca_długość_substytucji
          THEN BEGIN
            LZ.Zakoduj_pozycję_podmiany(i-Najdłuższa_fraza_zgodna)
            LZ.Zakoduj_substytucję()
            INC (i, Najdłuższa_fraza_zgodna)
            CONTINUE
          END ELSE BEGIN
            RC.Zachowaj_stan_kodera()
            Koniec_substytucji:= i + Najdłuższa_fraza_zgodna
          END
        END
        END
        PPM.Zakoduj_symbol(i)
        INC (i, 1)
      END
      LZ.Zakoduj_pozycję_podmiany(i)
      LZ.Zakończ()
      PPM.Zakończ()
```

W przypadku podanego wyżej algorytmu, warunkiem koniecznym do wykonania podmiany jest by długość najdłuższej znalezionej frazy zgodnej nie była mniejsza od minimalnej długości substytucji; warunkiem dostatecznym do wykonania podmiany jest by długość najdłuższej znalezionej frazy zgodnej nie była mniejsza od zadowalającej długości substytucji. Estymacji długości po zakodowaniu podlega każda fraza, która spełnia warunek konieczny, lecz nie spełnia warunku dostatecznego.

Opis funkcji, które wystąpiły już w przypadku algorytmu bez estymacji długości zapisu (LZ: Inicjalizuj, Znajdź_najdłuższą_frazę_zgodną, Zakoduj_pozycję_podmiany, Zakoduj_substytucję, Zakończ; PPM: Inicjalizuj, Zakoduj_symbol, Zakończ) znajduje się w punktach 3.4.3 i 3.4.4.

Długość_LZ zwraca długość zakodowanej frazy metodą substytucyjną. Prosty sposób na dokładne wyliczenie długości frazy zakodowanej jedną z metod składowych jest zasymulowanie kodowania frazy tą metodą. Symulowanie tym różni się od właściwego kodowania, że jedynie zwraca osiągnięty przyrost długości kodu, nie zapisując nic do strumienia wyjściowego, ani nie aktualizując modeli statystycznych, które wykorzystuje. Tak właśnie działa funkcja Długość_LZ.

W przypadku metody predykcyjnej, kodowanie (a właściwie poprzedzające je modelowanie) trwałoby zbyt długo, by cała estymacja (poprzez symulację) była opłacalna w sensie efektywności w czasie. Problem ten rozwiązano w ten sposób, że na początku frazy, której podmiana jest rozważana, zapisywany jest stan kodera (RC.Zachowaj_stan_kodera), a następnie wykonywane kodowanie metodą PPM. Po dotarciu do końca tej frazy, jej zakodowaną długość uzyskuje się odejmując bieżącą pozycję kodera od jej zachowanej kopii (to właśnie zadanie funkcji Długość_PPM). Jeżeli podmiana jest nieopłacalna, koder nie musi robić nic więcej, gdyż fraza została już zakodowana predykcyjnie. W przeciwnym wypadku, stan kodera z początku frazy jest odtwarzany (RC.Odtwórz_stan_kodera) i kodowany substytucyjnie, co nie jest operacją czasochłonną.

Podstawowy problem jaki rodzi taki sposób działania to utrudnienie realizacji zasady wyłączania aktualizacji dla fraz, których opłacalność podmiany estymuje się. Wymagałaby ona w tej sytuacji przechowywania nie tylko kopii stanu kodera statystycznego, ale również kopii stanu modelu PPM. Choć w istocie nie trzeba tworzyć kopii całego modelu, a jedynie tych jego elementów, które uległy zmianie, to ze względu na wykonywanie skalowań, które zmieniają duże obszary modelu PPM, nadal byłaby to operacja zbyt powolna.

Problem ten spróbowano rozwiązać poprzez zmniejszenie (w obszarze frazy, której opłacalność podmiany jest badana) stałych inkrementacji. Na przykład, zredukowanie ich do 0 oznaczałoby pełną realizację wykluczeń aktualizacji. Problem w tym, że dotyczy ono zarówno frazy, które zostały podmienione, jak i te, z których podmiany w wyniku estymacji ostatecznie zrezygnowano. Praktyczne testy pokazały, że nawet nadanie stałym inkrementacji kompromisowych wartości takich jak 0,5 nie daje istotnego polepszenia efektywności kompresji. Znacznie gorszą jego konsekwencją jest potrzeba wykonywania przez dekodera wyszukiwania fraz, by wiedział on, w których momentach zmniejszać stałe inkrementacji (tam, gdzie nie doszło do podmiany). Znacznie spowolniłoby to proces dekompresji.

Lepszy z praktycznego punktu widzenia sposób polega na posłużeniu się dodatkowym parametrem, to jest zadowalającą długością substytucji. Jeżeli znaleziona

fraza zgodna osiąga zadowalającą długością substytucji, podmiana wykonywana jest automatycznie, bez estymacji. W ten sposób, oprócz efektywności poprawie ulega czas kompresji (a także dekompresji). Im mniejsza różnica między zadowalającą a minimalną długością substytucji, tym szybsze działanie. Z tych powodów właśnie ten sposób zaimplementowany został w proponowanym algorytmie.

Dla dekompresji algorytm przełączający z estymacją długości zapisu ma następującą postać:

```
PLZ.Dekompresja:
LZ.Inicjalizuj()
PPM.Inicjalizuj()
i:=1;
Pozycja_podmiany:=LZ.Odkoduj_pozycję_podmiany()
WHILE i <= Rozmiar_źródła DO BEGIN
  IF i = Pozycja_podmiany THEN BEGIN
    Długość_frazy:=LZ.Odkoduj_substytucję(i)
    IF Długość_frazy < Zadowalająca_długość_substytucji THEN
      PPM.Uaktualnij_model(i, Długość_frazy)
    INC (i, Długość_frazy);
    Pozycja_podmiany:=LZ.Odkoduj_pozycję_podmiany(i)
  END ELSE BEGIN
    LZ.Uaktualnij_słownik(i)
    PPM.Odkoduj_symbol(i)
    INC (i, 1); END
END
```

Podobnie jak w przypadku kompresji, opis funkcji, które wystąpiły już w przypadku algorytmu bez estymacji długości zapisu znaleźć można w punktach 3.4.3 i 3.4.4. Jedyną nową funkcją, w stosunku do wersji bez estymacji zapisu, jest tutaj PPM.Uaktualnij_model. Jej zadaniem jest zapewnienie zgodności modelu PPM dekodera z modelem koder, poprzez uaktualnienie modelu PPM o symbole podmienionej frazy o długości poniżej zadowalającej długości substytucji.

3.5.2 Podstawowe właściwości algorytmu

Dodanie estymacji długości zapisu nie zmienia nic (w porównaniu do właściwości algorytmu bez estymacji długości, opisanych w punkcie 3.4.6) w zakresie stosowania zasady wyłączania symboli. Zmiany, które wywołuje w wyłączaniu aktualizacji omówione zaś zostały w punkcie 3.5.1.

Złożoność obliczeniowa algorytmu. Dodatkowe operacje wprowadzone wraz z estymacją długości (zapisywanie i odtwarzanie stanu koder statystycznego, symulacja kodowania substytucji) mają czas działania liniowo zależny od liczby symboli w zbiorze. Złożoność obliczeniowa całego algorytmu (przy stałych parametrach) pozostaje zatem liniowa: $O(n)$.

Złożoność pamięciowa algorytmu. Jedyną nową strukturą danych to bufor przechowania stanu kodera o złożoności $O(1)$. Złożoność pamięciowa całego algorytmu (przy stałych parametrach) pozostaje zatem liniowa: $O(n)$.

Długość po zakodowaniu. Przyjmijmy za L_{PPM1} maksymalną teoretyczną długość po zakodowaniu metodą predykcyjną pojedynczego symbolu źródła (tak jak opisano ją we wzorze 3.32). Przyjmijmy za L_{LZ1} maksymalną teoretyczną długość po zakodowaniu metodą substytucyjną pojedynczej frazy źródła (tak jak opisano ją we wzorze 3.33). Wtedy, maksymalna długość zbioru danych po zakodowaniu L_{max} dla proponowanego algorytmu nie przekroczy:

$$L_{max} = \gamma \sum_{i=1}^n \begin{cases} L_{PPM1}, & i \in F, \\ L_{LZ1}, & i \in E, \\ \min(L_{LZ1}, L_{PPM1}), & i \in G \setminus E, \\ 0, & i \notin F, \end{cases} \quad (3.35)$$

gdzie: n – liczba danych w zbiorze wejściowym, F – zbiór pozycji źródła dla których szukano, ale nie znaleziono frazy zgodnej o długości osiągającej minimalną długość substytucji, G – zbiór pozycji źródła dla których szukano i znaleziono frazę zgodną o długości osiągającej minimalną długość substytucji, E – zbiór pozycji źródła dla których znaleziona fraza zgodna osiąga zadowalającą długość substytucji.

3.5.3 Przykład kompresji algorytmem predykcyjno-substytucyjnym

Poniżej zademonstrowana zostanie kompresja algorytmem predykcyjno-substytucyjnym z estymacją długości zapisu na przykładzie następującego tekstu:

„ALA_MA_KOTA,ALA_DA_KOTA,ALA_DA_KOTA”.

Dla uproszczenia, w przykładzie pominięto drugorzędną estymację ucieczek i lokalną estymację rzędu, przyjęto także stałą długość zapisu substytucji – 10 bitów. Przyjęto następujące parametry użytkownika: rząd modelu predykcyjnego 1; wykorzystane rzędy słowników: 1 i 0d; wielkość każdego słownika: 1; minimalna długość substytucji: 2; zadowalająca długość substytucji: 8.

Rząd aktywnego modelu predykcyjnego	Zawartość słownika rzędu 1	Zawartość słownika rzędu 0d	Licznik symbolu	Suma liczników	Długość po zakodowaniu (bity)
<i>Początek danych</i>					
Sekwencja do zakodowania: ALA_MA_KOTA,ALA_DA_KOTA,ALA_DA_KOTA					
-1	Pusty	Pusty	1	256	8
Sekwencja do zakodowania: LA_MA_KOTA,ALA_DA_KOTA,ALA_DA_KOTA					
0	Pusty	Pusty	0/ESC 1	2	1
-1			1	256	8
Sekwencja do zakodowania: A_MA_KOTA,ALA_DA_KOTA,ALA_DA_KOTA					
0	Pusty	Pusty	1	4	2

Sekwencja do zakodowania: MA_KOTA,ALA_DA_KOTA,ALA_DA_KOTA					
1 (kontekst: 'A')	LA_M...	Pusty	0/ESC 1	2	1
0			0/ESC 2	5	1,3
-1			1	256	8
Sekwencja do zakodowania: MA_KOTA,ALA_DA_KOTA,ALA_DA_KOTA					
0	Pusty	Pusty	0/ESC 3	7	1,2
-1			1	256	8
Sekwencja do zakodowania: A_KOTA,ALA_DA_KOTA,ALA_DA_KOTA					
0	Pusty	Pusty	2	9	2,2
Sekwencja do zakodowania: KOTA,ALA_DA_KOTA,ALA_DA_KOTA					
1 (kontekst: 'A')	MA_...	Pusty	1	4	2
Sekwencja do zakodowania: KOTA,ALA_DA_KOTA,ALA_DA_KOTA					
1 (kontekst: ' ')	MA_K...	Pusty	0/ESC 1	2	1
0			0/ESC 4	10	1,3
-1			1	256	8
Sekwencja do zakodowania: OTA,ALA_DA_KOTA,ALA_DA_KOTA					
0	Pusty	Pusty	0/ESC 5	12	1,3
-1			1	256	8
Sekwencja do zakodowania: TA,ALA_DA_KOTA,ALA_DA_KOTA					
0	Pusty	Pusty	0/ESC 6	14	1,2
-1			1	256	8
Sekwencja do zakodowania: A,ALA_DA_KOTA,ALA_DA_KOTA					
0	Pusty	Pusty	3	16	2,4
Sekwencja do zakodowania: ,ALA_DA_KOTA,ALA_DA_KOTA					
1 (kontekst: 'A')	KOT...	Pusty	0/ESC 3	5	0,7
0			0/ESC 7	17	1,3
-1			1	256	8
Sekwencja do zakodowania: ALA_DA_KOTA,ALA_DA_KOTA					
0	Pusty	Pusty	4	19	2,2
Sekwencja do zakodowania: LA_DA_KOTA,ALA_DA_KOTA					
1 (kontekst: 'A')	,ALA...	Pusty	1	7	2,8
Sekwencja do zakodowania: A_DA_KOTA,ALA_DA_KOTA					
1 (kontekst: 'L')	A_MA...	Pusty	Znaleziono frazę zgodną: A_		
Fraza nie osiąga długości zadowalającej (2) -> Estymacja długości zapisu					
1 (kontekst: 'L')	Symbol: 'A'		1	2	1
1 (kontekst: 'A')	Symbol: ' '		1	8	3
Estymacja długości zapisu: 10>4, substytucja nieopłacalna					
Sekwencja do zakodowania: DA_KOTA,ALA_DA_KOTA					
1 (kontekst: ' ')	KOTA...	MA_K...	0/ESC 2	4	1
0			0/ESC 8	20	1,3
-1			1	256	8
Sekwencja do zakodowania: A_KOTA,ALA_DA_KOTA					
0	Pusty	A_KOT...	Znaleziono frazę zgodną: A_KOT...		
Fraza osiąga długość zadowalającą (18) -> Wykonano substytucję					10
Koniec danych					

Długość przed zakodowaniem: 280 bitów. Długość po zakodowaniu: 114 bitów.

3.5.4 Porównanie z istniejącymi algorytmami hybrydowymi

Zamieszczone niżej tabele zawierają zestawienie porównawcze podstawowych własności nowych algorytmów z istniejącymi algorytmami hybrydowymi.

Tabela 3.5. Hybrydowe algorytmy kompresji – porównanie, część pierwsza

Algorytm	LZP2	LZP4	LZFG-PM	Proponowany
Parametry użytkowe				
Efektywność kompresji	Niska	Wysoka	Niska	Średnia-Wysoka
Szybkość działania	Bardzo Wysoka	Niska	Średnia	Wysoka-Średnia
Wymogi pamięciowe	Bardzo Niskie	Wysokie	Średnie	Średnie-Wysokie
Stosowane rozwiązania				
Kodowanie symboli	Huffman	PPM	Kopiowanie	PPM
Estymacja długości zapisu	Nie	Nie	Nie	Tak
Specjalne słowniki rzędu 0	Nie	Nie	Nie	Tak
Funkcja mieszająca ustala kontekst frazy	Tak	Tak	Nie	Nie
Właściwości modeli				
Długość kontekstu dla symboli	0	1	0	≥ 1
Długość kontekstu dla fraz	3	5	1	≥ 1
Liczba elementów słownika	1	1	$\gg 1$	≥ 1

Tabela 3.6. Hybrydowe algorytmy kompresji – porównanie, część druga

Algorytm	ROLZ	UHARC	GRZip Fast	Proponowany
Parametry użytkowe				
Efektywność kompresji	Bardzo Wysoka	Bardzo Wysoka	Wysoka	Średnia-Wysoka
Szybkość działania	Bardzo Niska	Bardzo Niska	Średnia	Wysoka-Średnia
Wymogi pamięciowe	Wysokie	Wysokie	Średnie	Średnie-Wysokie
Stosowane rozwiązania				
Kodowanie symboli	PPMZ	PPM	ST	PPM
Estymacja długości zapisu	Nie	Nie	Nie	Tak
Specjalne słowniki rzędu 0	Nie	Nie	Nie	Tak
Funkcja mieszająca ustala kontekst frazy	Nie	Tak	Tak	Nie
Właściwości modeli				
Długość kontekstu dla symboli	≈ 16	4	4	≥ 1
Długość kontekstu dla fraz	1	4, 12	4, 5	≥ 1
Liczba elementów słownika	$\gg 1$	1	1	≥ 1

Żaden z wymienionych w tabelach 3.5 i 3.6 istniejących algorytmów nie posiada zaimplementowanej estymacji długości zapisu. Problem długości zapisu przełącznika podejmuje natomiast algorytm VW98. Choć nie jest on algorytmem hybrydowym (gdyż

przełączanie odbywa się w nim pomiędzy różnymi modelami predykcyjnymi, nie modelującymi fraz jako takich), w tabeli 3.7 przedstawiono porównanie z nim algorytmu predykcyjno-substytucyjnego.

Tabela 3.7. Algorytm predykcyjno-substytucyjny a VW98

Algorytm	VW98	Proponowany
Parametry użytkowe		
Efektywność kompresji	Bardzo Wysoka	Średnia-Wysoka
Szybkość działania	Bardzo Niska	Wysoka-Średnia
Wymogi pamięciowe	Bardzo Wysokie	Średnie-Wysokie
Stosowane rozwiązania		
Użyte modele	Symetryczne	Asymetryczne
Modele (reprezentacja w pamięci)	Dwa osobne	Jeden wspólny
Typ modeli	Predykcyjny	Pred.-Substytucyjny
Złożoność algorytmu przełączania	Duża	Mała
Złożoność komponentu predykcyjnego	Duża	Mała
Złożoność komponentu substytucyjnego	Duża	Mała
Wybór metody dla	Symbolu	Frazy
Metoda składowa	Ważona	Wybierana

Źródło tabel 3.5-7: opracowanie własne.

4 Weryfikacja metody predykccyjno-substytucyjnej

4.1 Metodologia badawcza

4.1.1 Testowy zestaw plików

Celem każdej innowacji powinno być poprawienie istniejącego stanu rzeczy. Od nowej metody kompresji wymagać należy zatem, że będzie lepsza niż dotąd dostępne. W dziedzinie kompresji danych zdefiniowanie pojęcia „lepsze” nie jest jednak rzeczą oczywistą.

Ocenę metod kompresji można przeprowadzać jedynie przy zachowaniu pewnych ograniczeń. Najpoważniejsze zastrzeżenie dotyczy przedmiotu kompresji.

Kompresja danych zamienia zbiór plików wejściowych F_1 w zbiór plików wyjściowych F_2 . W przypadku kompresji bezstratnej, każdemu plikowi wejściowemu odpowiada dokładnie jeden plik wyjściowy; i odwrotnie – każdemu plikowi wyjściowemu odpowiada dokładnie jeden plik wejściowy.

Jeżeli przyjmiemy że zbiór plików wejściowych zawiera wszystkie możliwe pliki, to okaże się, że nie można uzyskać skrócenia choćby jednego pliku wyjściowego nie powodując identycznego wydłużenia innego pliku wyjściowego. Nie istnieje zatem metoda kompresji bezstratnej, dla której średnia długość wszystkich możliwych plików wyjściowych byłaby krótsza od średniej długości wszystkich możliwych plików wejściowych.

Dopiero gdy ograniczymy się do podzbioru plików wejściowych $F_{1k} = F_1 \setminus F_{1n}$, skrócenie średniej długości odpowiadających im plików wyjściowych F_{2k} stanie się możliwe. Będzie ono okupione wydłużeniem plików z podzbioru F_{2n} .

W rzeczywistości, pliki o pewnej charakterystyce spotyka się o wiele częściej niż inne. Chcąc ocenić praktyczną wartość metody należy zatem ograniczyć badanie jej rezultatów do takich właśnie prawdopodobnych plików.

O tym, które typy danych są popularne decyduje badacz – czysto uznaniowo, opierając się na własnym doświadczeniu, lub opierając się na danych statystycznych, na przykład pochodzących z sieciowych systemów *cache*, czy też zebranych na losowej próbie użytkowników. Kiedy popularne typy danych zostaną już ustalone, efektywność

ich kompresji przy pomocy wybranej metody zbadać można na dwa sposoby: analitycznie bądź empirycznie.

Podejście analityczne polega na zbudowaniu modelu spójnego z posiadaną definicją źródła generującego badany typ danych i określeniu relacji pomiędzy entropią źródła (wyliczoną w oparciu o zbudowany model), a długością danych zakodowanych badaną metodą. Rezultaty badań przy takim podejściu mają znaczenie teoretyczne, zwykle jednak nikłe znaczenie praktyczne, gdyż:

- model jest tylko przybliżeniem rzeczywistości, a im ogólniejszego przypadku dotyczy, tym mniej staje się dokładny;
- zwykle potrafimy tak określić jedynie asymptotyczne właściwości metody:
 - rzeczywiście kompresowane zbiory danych są zwykle dalece zbyt krótkie, by miały one w ich przypadku znaczenie,
 - często odnoszą się one do przypadku najgorszego, podczas gdy w praktyce bardziej interesują nas przypadki typowe.

Podejście empiryczne polega na przeprowadzeniu eksperymentalnej kompresji badaną metodą pewnej skończonej liczby plików (**korpusu**). Jest uzasadnione tym, że efektywność kompresji plików o podobnej charakterystyce jest zbliżona; nie ma zatem powodu testować wielu plików o podobnej charakterystyce, skoro dla każdego uzyska się zbliżony rezultat.

Podejście empiryczne nie jest wolne od problemów. Nie jest oczywiste według jakich kryteriów wytypować pojedynczy plik jako reprezentanta swojego typu danych. O charakterystyce pojedynczego pliku stanowią bowiem zarówno czynniki strukturalne (określone przez jego typ), jak i czynniki specyficzne wyłącznie dla tego pliku, odróżniające go od wszystkich pozostałych.

Nie są oczywiste również proporcje, w jakich poszczególne typy danych występować powinny w korpusie. Jeżeli kryterium miała by być popularność, to typy bardziej popularne powinny posiadać w nim większy udział. Dodając do tego podobieństwo występujące pomiędzy charakterystykami różnych typów spodziewać się należy, że najbardziej popularna charakterystyka typu danych zdominuje wtedy korpus na tyle, iż rezultaty dla mniej popularnych charakterystyk stracą na znaczeniu. Nie jest to pożądane skoro celem jest badanie uniwersalnej, a nie ograniczonej do jednego typu danych, efektywności metody.

W niniejszej pracy, podobnie jak w wielu innych pracach dotyczących nowych metod kompresji (patrz badania literaturowe w [ARBE97]), przyjęto podejście empiryczne. Aby uniknąć problemów związanych z doбором reprezentatywnych

plików, wykorzystano dwa ogólnie uznane korpusy testowe: *Calgary Corpus* i *Canterbury Corpus*.

Motywytem użycia korpusu *Calgary* [WITT87] jest długoletnia tradycja posługiwania się nim. Wyniki osiągane na tym korpusie podawali prawie wszyscy cytowani wcześniej autorzy prac dotyczących nowych metod kompresji z ostatniego dziesięciolecia, między innymi: [ABEL03], [BAKU00], [BLOO96], [BUNT96], [DEOR03], [HOAN95] i wielu innych.

Korpus *Canterbury* [ARBE97] jest również uznawany za standardowy, a ponadto nowocześniejszy, o bardziej zróżnicowanej zawartości i posiada mocne uzasadnienie naukowe. Jego autorzy przeprowadzili szeroko zakrojone badania statystyczne, których celem było wybranie spośród około 800, jedenastu plików najbardziej charakterystycznych dla swoich typów. Z dwóch testowanych korpusów ten właśnie uznać należy za bardziej reprezentatywny.

Wadą obu tych korpusów jest to, że niektóre spośród zawartych w nich plików mają bardzo małą długość. Na przykład plik *grammar.lsp* z korpusu *Canterbury* ma zaledwie 3721 bajtów. Precyzja pomiaru czasu kompresji jest ograniczona, co przy tak krótkich plikach powodowałoby wypaczenie wyników.

Rozwiązano ten problem w sposób zaproponowany w [GILC02], łącząc wszystkie pliki korpusu w jeden plik (nieskompresowane archiwum typu TAR), który następnie poddaje się testom. Dodatkowo, pozwala to ocenić jednocześnie łączną efektywność kompresji dla wszystkich typów plików tworzących korpus, jak i zdolność algorytmu do kompresji danych o zmieniającej się charakterystyce.

W ten sposób traci się jednak możliwość oceny efektywności i szybkości kompresji dla poszczególnych typów danych ujętych w korpusie z osobna. Różnice w kompresji danych różnego typu mogą mieć dużą wartość poznawczą. Dlatego w niniejszej pracy, dla pokazania wpływu charakterystyki typu danych na szybkość i efektywność kompresji, badania wykonano także na trzech dodatkowych zbiorach danych.

Problem doboru dodatkowych plików testowych rozwiązano w oryginalny sposób. Zamiast dobierać pliki na podstawie ich semantycznego typu, dobrano je ze względu na ich podstawową cechę strukturalną. Jako że celem testów ma być weryfikacja metody łączącej podejście predykcyjne i substytucyjne, za szczególnie nadającą się do tego zadania cechę uznano współczynnik substytucyjności.

W tej sytuacji, tak jak opisano to w punkcie 3.1.2, prostym sposobem klasyfikacji plików jest pogrupowanie ich według efektywności kompresji dwoma algorytmami różnego typu. W tym celu posłużono się programami: Zip (substytucyjny

algorytm Deflate, długość po zakodowaniu oznaczona L_{LZ}), oraz PPMd (predykcyjny algorytm PPMII, rząd 2, długość oznaczona L_{PPM}). Pliki podzielono na trzy grupy:

- I – o niskim współczynniku substytucyjności, dla których $L_{LZ} > 1,1L_{PPM}$,
- II – o wysokim współczynniku substytucyjności, dla których $1,1L_{LZ} < L_{PPM}$,
- III – o średnim współczynniku substytucyjności, dla których $L_{LZ} \approx L_{PPM}$.

Podział taki, choć prosty, ma tę zaletę, że przynajmniej dwie pierwsze grupy zawierają pliki o istotnie różnej charakterystyce. Nawet jeżeli wybierze się tylko po jednym pliku należącym do każdej z grup, to wyniki uzyskane dla nich będą stanowić lepszą informację o uniwersalnej przydatności testowanego algorytmu, aniżeli korpusy z większą liczbą plików, ale zdominowane przez jeden ich typ.

Przy podziale w oparciu o jedno tylko kryterium, nie można zakładać, że wszystkie pliki zakwalifikowane do danej grupy będą w podobny sposób poddawać się kompresji. W szczególności, wyraźne odstępstwa powodowane być mogą przez takie właściwości kompresowanego zbioru jak bardzo mały alfabet źródła (np. sekwencje DNA), czy występowanie korelacji specyficznego typu (np. dane sygnałowe). Zauważmy jednak, że pliki należące do pierwszej z wymienionych kategorii trudno określić jako popularne. Dążąc do otrzymania jak najmniejszego zestawu testowego, nie ma powodu uwzględniać tak egzotycznych plików. Z kolei dla plików należących do drugiej kategorii metody uniwersalne uzyskują niską efektywność kompresji. Można ją radykalnie podwyższyć stosując proste techniki przetwarzania wstępnego (co robią prawie wszystkie nowe programy kompresujące, na przykład Rar [ROSZ03], GRZip [GREB03], czy UHARC [HERK02]). Zważywszy, że celem nie jest analiza algorytmów przetwarzania wstępnego, dołączenie do zestawu pliku z danymi sygnałowymi nie jest uzasadnione. W konsekwencji, zaproponowany prosty podział uznać można za wystarczający dla dużej części popularnych typów danych.

Mimo że klasyfikacja plików oparta na współczynniku substytucyjności nie ma podstaw semantycznych, dla wielu popularnych typów danych można wskazać jako właściwą jedną z otrzymanych grup. W szczególności, większość plików zawierających tradycyjne dane tekstowe, jak na przykład: tekst w języku naturalnym, w tym w postaci XML z małym udziałem oznaczeń [SWAC03], HTML, kod źródłowy programów, tekst z formatowaniem binarnym (DOC), należy do grupy I.

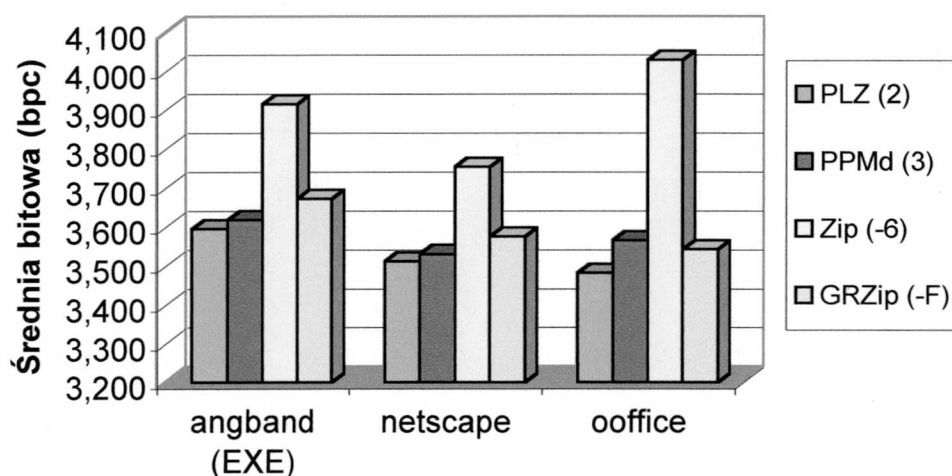
Większość plików zawierających dane wysoce powtarzalne, jak na przykład XML z dużym udziałem oznaczeń [SWAC03], duża część grafiki wektorowej (SVG, PS), prosta grafika rastrowa (BMP), czy własne formaty danych niektórych aplikacji, należy do grupy II. Grupa III to różnego rodzaju dane binarne, w szczególności obejmuje większość plików zawierających kod wykonywalny programów.

Do testów w sposób losowy wybrane zostały trzy pliki, których listę zamieszczono w tabeli 4.1 pod nagłówkiem *Dodatkowe pliki testowe*. Spełniają one z powodzeniem przedstawione wyżej warunki podziału: różnice w długości po kompresji pomiędzy algorytmem substytucyjnym i predykcyjnym wynoszą dla plików z grup I i II ponad 25%, czyli znacznie więcej niż założone 10%.

Tabela 4.1. Pliki wykorzystane w testach

Plik	Nazwa oryginalna	Opis	Wielkość	Źródło
Korpusy testowe				
Calgary Corpus.TAR		[WITT87]	3 152 384	www.data-comp ression.info
Canterbury Corpus.TAR		[ARBE97]	2 821 120	
Dodatkowe pliki testowe				
EXE	Angband.EXE	Plik wykonywalny IA-32	696 832	www.thango rodrim.net
TXT	Dowcipy.TXT	Dowcipy w języku polskim	536 049	www.dowcipy.pl
XML	Xerces.XML	Dokumentacja techniczna	1 841 811	xml.apache.org
Long.XML		Dane aukcyjne	21 480 778	XMLBenchmark

Wielkość podano w bajtach. Źródło tej i wszystkich pozostałych tabeli w tym rozdziale: badania własne.



Rys. 4.1. Średnia bitowa dla pliku EXE i plików z innych korpusów

Rezultaty kompresji metodą predykcyjno-substytucyjną trzech wybranych plików nie są ani najlepszymi ani najgorszymi zanotowanymi dla danej grupy. Przykładem tego jest rysunek 4.1, który pokazuje średnią bitową uzyskaną dla wybranego do testów pliku EXE i dwóch plików tego samego typu pochodzących z innych korpusów (*ooffice* z [DEOR03] i *netscape* z [GILC02]). Pamiętajmy także, że wyniki uzyskane na plikach dodatkowych stanowią jedynie swego rodzaju objaśnienie do wyników uzyskanych na omówionych już korpusach standardowych.

Ważnym kryterium doboru plików do korpusu jest ich długość. Różnice w długości po zakodowaniu pomiędzy porównywanymi metodami dla krótkich plików

są śladowe, a składniki nieistotne (np. wielkość nagłówka skompresowanego pliku) stają się dostrzegalne w rezultatach. Jeszcze gorzej zbyt mała długość pliku rzutuje na pomiary czasu. Czas uruchomienia programu może znacznie przekraczać czas samej kompresji wypaczając wyniki porównania.

Wielkość pliku powyżej kilkuset kilobajtów jest w zupełności wystarczająca do zadowalająco dokładnego pomiaru zarówno wielkości pliku po kompresji jak i czasu kompresji (dla wykorzystanego środowiska testowego). Pliki o wysokiej powtarzalności powinny być większe (powyżej 1 MB), gdyż kompresowane są szybko, a ich długość po zakodowaniu jest niewielka. Proponowane trzy pliki dodatkowe spełniają te wymagania.

Aby zbadać jak zwiększenie rozmiaru pliku wpływa na efektywność i szybkość kompresji należy użyć jednak znacznie większych plików. Niektórzy autorzy (na przykład G. Manzini i S. Deorowicz [DEOR03]) zaproponowali z tego powodu całe własne korpusy zawierające znacznie dłuższe pliki.

Wykonywanie wszystkich testów na tak dużych plikach jest oczywistym utrudnieniem badań, ze względu na wielokrotne wydłużenie czasu trwania testów. Dlatego do sprawdzenia jak wpływa zwiększenie rozmiaru pliku na efektywność i szybkość kompresji metodą predykcyjno-substytucyjną posłużono się pojedynczym dużym plikiem.

Aby mieć pewność, że odmienność wyników uzyskanych dla fragmentów badanego pliku o różnych długościach, bierze się właśnie z różnicy w długości, a nie z lokalnej zmiany charakterystyki danych, należy mieć pewność, że takie zmiany w testowanym pliku nie występują. Zamiast poszukiwać rzeczywistego pliku o takiej własności, prościej użyć pliku wygenerowanego automatycznie, o którym wiadomo, że każda jego część powstała na tych samych zasadach. Taką możliwość stwarza program XMLBenchmark [SCHM01], imitujący dane z internetowego systemu aukcyjnego. Przy pomocy tego programu wygenerowano plik *Long* o długości około 24 MB, która jest wystarczająca do przeprowadzenia zamierzonego badania.

4.1.2 Sposób prowadzenia badań

Doświadczenia mające na celu określenie efektywności i szybkości przeprowadzone zostaną przy pomocy programu PLZ, będącego autorską implementacją przedstawionych w rozdziale 3 algorytmów predykcyjno-substytucyjnych. Program ten powstawał równolegle z pisanie niniejszej pracy. Z tego powodu można przypuszczać, że nie stanowi on w pełni optymalnej implementacji

proponowanych algorytmów, szczególnie, jeżeli wziąć pod uwagę jak długo rozwijana była większość programów, z którymi będzie on dalej porównywany.

Program PLZ prawie w całości¹¹ powstał w języku Pascal, skompilowany zaś został przy użyciu *Borland Delphi 7*. Wybór takiego języka i kompilatora miał przyczyny pozamerytoryczne – zamierzeniem było zachowanie wysokiej przejrzystości całego programu, pojmowanej jako istotny walor edukacyjny. Należy domniemywać, że w porównaniu do napisanych w większości w języku C innych porównywanych programów, oznacza to dodatkowy czynnik spowalniający. Wynika to z pewnych braków języka Pascal¹², jak również z miejscami gorszej niż w innych kompilatorach optymalizacji¹³. Wszystkie wyniki szybkości uzyskiwane przez program PLZ należy zatem traktować jako dolną granicę szybkości zaimplementowanej w nim metody.

Jako długość po kompresji przyjęto całkowitą długość skompresowanego pliku. Jest to źródłem pewnej niedokładności wynikającej z wielkości nagłówka poprzedzającego skompresowane dane. Znacznie upraszcza to jednak procedurę badawczą, rzeczywista długość pliku po kompresji nie zawsze bowiem jest przez programy podawana.

Specyfikacja techniczna platformy testowej jest następująca: procesor Intel *Celeron* 1000 MHz, pamięć 256 MB SDR 100 MHz, dysk twardy Seagate *Barracuda* IV 40 GB ATA 100 (2 MB *cache*), system operacyjny Microsoft *Windows* 98 SE.

W przypadku pomiaru czasu działania pojawia się problem rozrzutu uzyskiwanych wyników. Ze względu na pamięć *cache* dysku i procesora, sposób działania systemu operacyjnego oraz mechanizm predykcji skoków, czas wykonywania tej samej operacji przy kolejnych uruchomieniach tego samego programu jest różny. Zwykle po pierwszym, wolniejszym uruchomieniu, kolejne są szybsze (efekt *cache*).

Przykładowe wyniki dla kolejnych 20 pomiarów szybkości kompresji pliku *TXT* przez programy PLZ i PPMd zawiera tabela 4.2. Jak widać, wśród pierwszych trzech pomiarów bardzo prawdopodobne są szybkości znacznie wolniejsze od późniejszych. Trudno jednoznacznie określić wszystkie tego przyczyny; zapewne jedną z nich jest to, że dla późniejszych uruchomień dane znajdują się już w pamięci *cache* dysku.

¹¹ Koder statystyczny napisany został częściowo w asemblerze, w istotny sposób nie wpływa to jednak na szybkość działania całego algorytmu, którego główny ciężar spoczywa w innych punktach.

¹² Na przykład – w porównaniu do języka C – brak makr czy funkcji typu *inline*.

¹³ Analiza kodu wykonywalnego ujawniła dalekie od optymalnego wykorzystanie rejestrów procesora, a w programie wykorzystującym tak duże struktury danych jak PLZ każde niepotrzebne odwołanie do pamięci powoduje zauważalny spadek szybkości.

Z jednej strony, w praktyce te same dane kompresuje się zwykle tylko jeden raz, należałoby więc wykonywać tylko jeden pomiar. Niestety, w przypadku początkowych pomiarów rozrzut wartości zmierzonych w kolejnych eksperymentach jest bardzo duży. Ponadto, aby mieć pewność, że warunki dla wszystkich testowanych programów są identyczne, należałoby wykonywać restart systemu po każdym pomiarze.

Tabela 4.2. Wartości przeciętne dla kolejnych 20 pomiarów szybkości kompresji (kB/s) pliku *TXT* przez programy PLZ i PPMd

Wszystkie 20 pomiarów	PLZ	PPMd
Min	2485	2382
Średnia	3609	4506
Mediana	3799	4786
Max	3805	4786
Odchylenie standardowe	465	702
Pierwsze 4 pomiary		
Min	2485	2382
Średnia	2851	3384
Mediana	2557	3184
Max	3804	4786
Odchylenie standardowe	639	1012
Kolejne 16 pomiarów		
Min	3792	4786
Średnia	3799	4786
Mediana	3800	4786
Max	3805	4786
Odchylenie standardowe	5	0

Z drugiej strony, celem jest badanie szybkości kompresji, a nie podsystemu wejścia-wyjścia, tudzież innych zewnętrznych przyczyn spowolnienia. Udział czasu przypadającego na tego rodzaju operacje należy więc minimalizować, poprzez ignorowanie początkowych pomiarów. To podejście ma także tę zaletę, że wyniki otrzymywane dla późniejszych pomiarów są znacznie bardziej powtarzalne w kolejnych eksperymentach, niż ma to miejsce w przypadku pomiaru pierwszego (patrz tabela 4.2).

W toku prowadzonych badań okazało się, że przynajmniej jeden spośród pierwszych 4 pomiarów jest zbliżony do wartości typowej dla pomiarów późniejszych, co pokazuje tabela 4.2. Prostem i nie wymagającym wykonania wielu pomiarów sposobem określenia szybkości reprezentatywnej dla danego programu i pliku testowego, będzie zatem znalezienie maksymalnej zaobserwowanej szybkości dla 4 następujących po sobie pomiarów. Jedyne zastrzeżenie do przyjętego sposobu pomiaru jest takie, że faworyzuje on w pewnym niewielkim stopniu algorytmy szybsze, gdyż dla nich opóźnienie związane na przykład z operacjami wejścia-wyjścia stanowi większą część czasu działania, aniżeli dzieje się to w przypadku algorytmów wolniejszych.

Z drugiej strony, podejście odwrotne faworyzowałoby algorytmy wolniejsze na niekorzyść szybszych.

4.2 Eksperymenty z różnymi parametrami użytkownika

4.2.1 Modyfikacja rzędu modelu statystycznego

Prezentowanie wyników przeprowadzonych testów rozpoczyna tradycyjne zestawienie zawierające wartość średniej bitowej uzyskanej dla poszczególnych plików korpusu *Calgary*.

Tabela 4.3. Średnia bitowa (w bpc) dla różnych rzędów modelu uzyskana programem PLZ na plikach tworzących *Calgary Corpus*

Plik	Rząd: 1	2	3	3+
<i>bib</i>	2,973	2,266	2,034	2,033
<i>book1</i>	3,474	2,847	2,408	2,364
<i>book2</i>	3,162	2,424	2,180	2,144
<i>geo</i>	4,626	4,560	4,409	4,409
<i>news</i>	3,192	2,616	2,375	2,364
<i>obj1</i>	3,832	3,794	3,632	3,592
<i>obj2</i>	2,545	2,320	2,185	2,136
<i>paper1</i>	3,142	2,616	2,364	2,324
<i>paper2</i>	3,249	2,680	2,408	2,380
<i>pic</i>	0,688	0,704	0,654	0,661
<i>progc</i>	2,898	2,565	2,351	2,309
<i>progl</i>	2,014	1,692	1,558	1,534
<i>progp</i>	1,905	1,690	1,564	1,524
<i>trans</i>	1,870	1,431	1,343	1,325
Średnia	2,826	2,443	2,248	2,221

Poszczególne kolumny tabeli zawierają wyniki uzyskane przez program PLZ dla następujących parametrów użytkownika:

- (1) rząd modelu PPM: 1, wielkość słownika rzędu 2: 8, wielkość słownika rzędu 0d: 4, minimalna długość substytucji: 4, estymacja długości zapisu wyłączona;
- (2) rząd modelu PPM: 2, wielkość słownika rzędu 2: 8, wielkość słownika rzędu 0d: 4, minimalna długość substytucji: 4, estymacja długości zapisu wyłączona;
- (3) rząd modelu PPM: 3, wielkość słownika rzędu 2: 4, wielkość słownika rzędu 0d: 2, minimalna długość substytucji: 4, estymacja długości zapisu wyłączona;
- (3+) rząd modelu PPM: 3, wielkość słownika rzędu 2: 4, wielkość słownika rzędu 0d: 2, minimalna długość substytucji: 4, estymacja długości zapisu włączona – zadowalająca długość substytucji 6.

O ile dla danego testu nie zostanie podane inaczej, w dalszej części pracy oznaczenia PLZ 1, 2, 3 i 3+ odnosiły się będą do programu PLZ w wymienionych wyżej konfiguracjach.

We wszystkich przypadkach program wywołano z opcją alokacji 32MB pamięci. Wielkość ta znacząco przekracza wielkości modeli tworzonych dla poszczególnych plików testowych, co gwarantowało, że w przypadku żadnego pliku nie doszło do wyczyszczenia modelu predykcyjnego bądź słownika substytucyjnego.

Z uwagi na to, że istotnym czynnikiem uwzględnionym w badaniach jest czas, modeli rzędu wyższego niż 3 nie badano, wychodząc z założenia, że czas kompresji w ich przypadku byłby zbyt duży dla rozpatrywanych prędkości transmisji.

Przypatrując się wynikom pomiarów, z pewnymi wyjątkami (pliki *geo*, *pic*), widoczna jest poprawa efektywności kompresji wraz ze wzrostem rzędu modelu. Za wyjątkiem wspomnianych dwóch plików, poprawie efektywności służy też estymacja długości zapisu. Jak widać, jej wpływ jest jednak ograniczony. Przyczyną tego jest niski rząd testowanego modelu. Wyników dla modeli rzędu 1 i 2 z estymacją długości zapisu nie przedstawiono, gdyż w ich przypadku uzyskiwana poprawa była bardzo niewielka.

Taki stan rzeczy należy interpretować w ten sposób, że właściwie dobrana minimalna długość substytucji, dla modeli niskich rzędów, wystarcza by odsetek podmian niekorzystnych ze względu na długość zapisu był niewielki. Znaczenie estymacji długości zapisu rośnie wraz z rzędem modelu [SWSZ02].

Gdy odniesie się uzyskaną średnią efektywność do wyników zamieszczonych w tabeli 2.5, to wyniki programu PLZ dla rzędu modelu 1 sytuują go w pobliżu takich algorytmów jak DMC i LZFG-PM, dla rzędu 2 – pomiędzy PPM rzędu 3 a oryginalnym BWT, natomiast dla rzędu 3 – blisko implementacji PPM* S. Bunton.

Sama efektywność kompresji nie może wystarczać do pełnego opisu pracy algorytmu. Więcej informacji zawiera tabela 4.4, w której zamieszczono: łączny czas przetwarzania danych przez algorytm (suma czasów kompresji i dekompresji), wielkość modelu PLZ (nie obejmuje struktur dodatkowych – bufora wejścia (słownikowego), wyjścia i tablic kontekstów, których wielkość jest taka sama dla wszystkich plików), uzyskaną średnią bitową (dla całego korpusu łącznie, a więc nie jest to wartość identyczna ze średnią z tabeli 4.3), oraz przyspieszenie transmisji dla założonej prędkości transmisji 512 kb/s, dobranej przypomnijmy tak, by nie deprecjonować znaczenia ani czasu, ani efektywności kompresji.

Zwiększenie rzędu modelu najmocniej wpływa na wielkość modelu: rośnie ona drastycznie szybko z każdym zwiększeniem rzędu o 1. Także włączenie estymacji długości zapisu powoduje jej wzrost, w tym wypadku jednak niewielki. Choć nie tak

gwałtownie jak rozmiar modelu, ale również istotnie wzrasta czas przetwarzania danych. Nadal wyraźnie, lecz w jeszcze mniejszym tempie wraz ze zwiększaniem rzędu modelu maleje średnia bitowa.

Miara łącząca czas i efektywność kompresji, współczynnik przyspieszenia transmisji, w przypadku każdego z ujętych w tabeli plików, osiąga maksimum dla rzędu modelu równego 2. Oznacza to, że powyżej tego poziomu rzędu modelu, poprawa efektywności okupiona jest zbyt dużym wydłużeniem czasu działania.

Dla modelu rzędu 3, użycie estymacji długości zapisu poprawia *WPT* dla wszystkich testowanych plików za wyjątkiem *XML*.

Tabela 4.4. Porównanie całościowe dla różnych rzędów modelu

Rząd modelu	Łączny czas przetwarzania (s)	Wielkość modelu (kB)	Średnia bitowa (bpc)	WPT dla 512 kb/s
<i>Calgary Corpus</i>				
1	1,610	86,3	2,923	2,453
2	2,083	1366,9	2,346	2,910
3	4,819	9669,8	2,138	2,675
3+	4,725	10323,4	2,111	2,714
<i>Canterbury Corpus</i>				
1	1,325	25,8	1,833	3,768
2	1,428	455,4	1,550	4,319
3	2,661	5115,1	1,431	4,081
3+	2,754	6143,7	1,387	4,141
<i>TXT</i>				
1	0,672	7,7	3,270	1,997
2	0,769	145,8	2,819	2,199
3	1,241	2334,8	2,554	2,089
3+	1,243	2491,2	2,522	2,106
<i>EXE</i>				
1	0,808	110,0	3,863	1,753
2	1,205	1502,7	3,552	1,761
3	2,021	4694,5	3,360	1,613
3+	2,011	4967,1	3,301	1,635
<i>XML</i>				
1	0,548	16,7	0,109	29,929
2	0,549	346,5	0,079	33,673
3	0,649	4089,4	0,079	30,070
3+	0,659	4392,5	0,080	29,683

4.2.2 Modyfikacja wielkości słowników i minimalnej długości substytucji

Istotną dla kompresji metodą predykcyjno-substytucyjną kwestią jest których słowników użyć i jaki powinny mieć one rozmiar. Każde zwiększenie rozmiaru słownika powoduje zwiększenie zapotrzebowania na pamięć, a także zwiększenie czasu

wyszukiwania frazy zgodnej. Interesujące jest zatem, przy tej samej wielkości słownika, dla słowników jakiego rzędu uzyska się największą liczbę podmienionych symboli. Tabela 4.5 pokazuje łączny odsetek podmienionych symboli w pliku *Calgary Corpus*, w zależności od rozmiarów słowników poszczególnych rzędów, przy ograniczeniu sumy rozmiarów wszystkich słowników do jednego (ujęto wszystkie możliwe kombinacje) i trzech (tylko najlepsze i najgorsze kombinacje) elementów.

Analiza tabeli 4.5 pozwala wysnuć następujące wnioski:

- Różnice w liczbie podmienianych przez komponent substytucyjny symboli, w zależności od rzędu użytego słownika, są duże; w skrajnym przypadku, najlepsza możliwa kombinacja podmienia 2,5 razy więcej symboli od najgorszej.

Tabela 4.5. Łączny odsetek podmienionych symboli (plik *Calgary Corpus*), w zależności od rozmiarów słowników poszczególnych rzędów

0c	0d	1	2	3	4	Usunięto
Dla sumy rozmiarów wszystkich słowników równej 1:						
0	0	0	1	0	0	33%
0	0	0	0	0	1	25%
0	0	1	0	0	0	25%
0	0	0	0	1	0	15%
Dla sumy rozmiarów wszystkich słowników równej 3:						
0	0	0	3	0	0	40%
0	1	0	2	0	0	39%
0	0	0	2	0	1	38%
0	0	1	2	0	0	38%
0	0	0	2	1	0	38%
1	0	0	2	0	0	38%
0	1	0	1	0	1	36%
0	2	0	0	1	0	16%
0	1	0	1	0	1	16%
2	0	0	0	1	0	16%

- Zwiększanie rozmiaru słownika jest zasadne, gdyż powoduje wzrost liczby podmienianych symboli: najlepsza kombinacja przy limicie sumy rozmiarów wszystkich słowników równej 3 jest o ponad 21% lepsza od najlepszej kombinacji przy limicie sumy rozmiarów wszystkich słowników równej 1.
- Najbardziej skuteczny, w sensie maksymalizowania liczby symboli, które można dzięki niemu podmienić, jest słownik rzędu 2. Spośród innych słowników, najlepszy rezultat daje łączenie go ze słownikiem rzędu 0d.

Ze względu na powyższe obserwacje, w testowanych konfiguracjach programu PLZ używa się tylko dwóch rodzajów słowników: rzędu 2 oraz 0d. Tabela 4.6 pokazuje, jak zmienia się szybkość i efektywność kompresji zmierzona dla korpusów testowych, w zależności od rozmiaru słownika rzędu 2 (przy rzędzie modelu PPM wynoszącym 2

i rozmiarze słownika rzędu 0d równym 2). Jak widać, zwiększanie rozmiaru słownika powoduje wzrost efektywności kompresji, któremu towarzyszy wzrost czasu przetwarzania. Choć czas dekompresji zmienia się tylko nieznacznie (i niekoniecznie rośnie), dla większych rozmiarów słownika znacznie wydłuża się czas kompresji. Lepszym sposobem poprawienia efektywności kompresji jest zwiększanie rzędu modelu PPM (patrz tabela 4.4), choć zwiększanie rozmiaru słownika ma tę zaletę, że nie wpływa istotnie na szybkość dekompresji. Najlepszy współczynnik przyspieszenia transmisji odnotowano dla rozmiaru słownika równego 8, dlatego słownika tej wielkości użyto w konfiguracjach PLZ dla rzędu modelu 1 i 2.

Tabela 4.6. Rozmiar słownika rzędu 2 a szybkość i efektywność kompresji

$ D^2 $	<i>Calgary Corpus</i>				<i>Canterbury Corpus</i>			
	<i>BR</i> (bpc)	t_k (s)	t_d (s)	<i>WPT</i>	<i>BR</i> (bpc)	t_k (s)	t_d (s)	<i>WPT</i>
2	2,420	0,983	0,989	2,853	1,560	0,713	0,660	4,319
3	2,396	0,983	0,935	2,887	1,552	0,714	0,659	4,340
4	2,381	1,093	0,930	2,885	1,547	0,710	0,654	4,355
5	2,369	1,094	0,880	2,906	1,541	0,764	0,659	4,342
6	2,360	1,203	0,879	2,897	1,537	0,764	0,659	4,352
8	2,347	1,203	0,880	2,911	1,533	0,769	0,653	4,362
12	2,330	1,319	0,880	2,908	1,523	0,824	0,659	4,359
24	2,305	1,865	0,879	2,839	1,512	1,095	0,659	4,266
48	2,286	2,853	0,934	2,689	1,507	1,644	0,605	4,072
96	2,275	4,996	0,984	2,398	1,505	2,743	0,604	3,686

Minimalna długość substytucji powinna być dostosowana do kompresowanych danych: powinna być tym wyższa, im efektywniej dane te kompresowane są przy użyciu metody predykcyjnej. Tabela 4.7 zawiera rezultaty testów wykonanych na korpusach testowych, reprezentujących dane różnego typu. Użyto programu PLZ w konfiguracji zgodnej z PLZ2, oczywiście za wyjątkiem wielkości minimalnej długości substytucji.

Tabela 4.7. Minimalna długość substytucji a szybkość i efektywność kompresji

l_{min}	<i>Calgary Corpus</i>				<i>Canterbury Corpus</i>			
	<i>BR</i> (bpc)	t_k (s)	t_d (s)	<i>WPT</i>	<i>BR</i> (bpc)	t_k (s)	t_d (s)	<i>WPT</i>
4	2,346	1,204	0,875	2,910	1,550	0,819	0,654	4,317
5	2,356	1,258	0,881	2,891	1,548	0,824	0,655	4,301
6	2,378	1,264	0,934	2,858	1,559	0,875	0,654	4,253
7	2,405	1,320	0,984	2,812	1,555	0,879	0,655	4,260
8	2,432	1,374	0,984	2,776	1,567	0,878	0,659	4,231
12	2,517	1,478	1,039	2,670	2,059	1,209	0,875	3,204
16	2,575	1,483	1,040	2,618	2,108	1,264	0,879	3,128
24	2,639	1,534	1,044	2,556	2,131	1,259	0,874	3,102

Jak pokazują otrzymane wyniki, zarówno efektywność kompresji jak i szybkość działania spada wraz ze wzrostem minimalnej długości substytucji. W pewnym stopniu

wyniki te wyjaśniają niską efektywność estymacji długości zapisu dla niskich rzędów modelu: zdecydowana większość podmian skraca długość zapisu; gdyby było inaczej, zwiększenie minimalnej długości substytucji spowodowałoby poprawę efektywności kompresji.

4.2.3 Długość kompresowanego pliku a rezultaty kompresji

Model PLZ jest modelem adaptacyjnym, co oznacza, że jest budowany w trakcie kompresji w oparciu o charakterystykę przetwarzanych danych. Jeżeli zatem dane są homogeniczne, dla dłuższych plików powinniśmy zaobserwować lepszą efektywność kompresji. Model dla dłuższych plików będzie lepiej estymował prawdopodobieństwa kolejnych symboli, gdyż oparty jest na większej liczbie obserwacji, niż ma to miejsce w przypadku plików krótkich.

Aby pokazać, jak te teoretyczne przesłanki sprawdzają się w praktyce, w tabeli 4.8 zamieszczono rezultaty uzyskane na kolejnych częściach pliku *Long* kompresowanego programem PLZ 2.

Tabela 4.8. Rezultaty testów dla kolejnych części pliku *Long*

Część	Szybkość kompresji (kB/s)		Wielkość modelu (B)		Średnia bitowa (bpc)		WPT dla 512 kB/s	
	Część	Łącznie	Przyrost	Łącznie	Część	Łącznie	Część	Łącznie
1%	439	439	75480	75480	2,505	2,505	1,844	1,844
5%	1859	1129	18932	94412	2,489	2,492	2,737	2,495
25%	2883	2199	16684	111096	2,447	2,456	2,934	2,835
50%	3529	2710	6720	117816	2,447	2,452	2,990	2,910
75%	3602	2954	2848	120664	2,445	2,449	2,998	2,939
100%	3626	3097	1704	122368	2,434	2,445	3,012	2,957

Kolumna *Wielkość modelu* pokazuje, że duża część modelu powstaje już podczas kompresji początkowego 1% pliku *Long*. Rozbudowa modelu trwa w zasadzie jeszcze do 25% pliku, ale już z mniejszym natężeniem; dalsze powiększenie modelu jest już znikome.

Szybkość kompresji¹⁴ w pierwszej części pliku jest wolniejsza, ze względu na trwającą równolegle z kodowaniem danych rozbudowę modelu. Czas kompresji pierwszych kilku procent pliku jest nieproporcjonalnie wydłużony, gdyż właśnie na ten etap przypada główny ciężar tworzenia modelu.

Pomiary średniej bitowej potwierdzają, że efektywność kompresji poprawia się wraz z przechodzeniem do dalszych części pliku *Long*. Jako że model jest już w dużym

¹⁴ Uwaga. W odróżnieniu od pozostałych pomiarów czasu w tym rozdziale, szybkości podane w tabeli 4.8 mierzone były w głównej pętli programu. Nie ujmują one zatem wszystkich składników czasu wykonywania programu.

stopniu gotowy po przetworzeniu początkowego fragmentu danych, dalej poprawa ta jest ledwie zauważalna.

Jednoczesne przyspieszenie i poprawa efektywności kompresji powodują wyraźną poprawę *WPT* dla dalszych części pliku.

4.3 Porównanie z użytym algorytmem predykcyjnym

Podstawowym pytaniem dotyczącym metody predykcyjno-substytucyjnej jest czy rozszerzenie metody predykcyjnej o komponent substytucyjny przyniosło zamierzony pozytywny efekt. Ocena poprawy kompresji uzyskanej przez dodanie możliwości wykonywania substytucji będzie możliwa dzięki zestawieniu rezultatów uzyskanych przez algorytm predykcyjno-substytucyjny do rezultatów algorytmu użytego w jego komponencie predykcyjnym. Zestawienie takie ujęto w tabeli 4.9.

Tabela 4.9. Porównanie z użytym algorytmem predykcyjnym

Rząd modelu	Łączny czas przetwarzania (s)		Średnia bitowa (bpc)		<i>WPT</i> dla 512 kb/s	
	PPM	PLZ	PPM	PLZ	PPM	PLZ
<i>Calgary Corpus</i>						
1	1,193	1,610	3,725	2,923	1,994	2,453
2	1,259	2,083	2,933	2,346	2,490	2,910
3+	3,522	4,725	2,430	2,111	2,602	2,714
<i>Canterbury Corpus</i>						
1	0,972	1,325	3,049	1,833	2,422	3,768
2	1,043	1,428	2,437	1,550	2,975	4,319
3+	3,263	2,754	1,919	1,387	3,111	4,141
<i>TXT</i>						
1	0,650	0,672	3,569	3,270	1,865	1,997
2	0,658	0,769	3,050	2,819	2,124	2,199
3+	1,310	1,243	2,631	2,522	2,012	2,106
<i>EXE</i>						
1	0,771	0,808	4,901	3,863	1,429	1,753
2	1,093	1,205	4,054	3,552	1,608	1,761
3+	2,086	2,011	3,630	3,301	1,513	1,635
<i>XML</i>						
1	1,049	0,548	1,427	0,109	4,545	29,929
2	1,039	0,549	0,584	0,079	8,948	33,673
3+	1,623	0,659	0,447	0,080	8,701	29,683

Łączny czas przetwarzania danych w stosunku do algorytmu czysto predykcyjnego uległ zwiększeniu dla dwóch najniższych rzędów modelu. Wyjątkiem jest wysoce powtarzalny plik *XML*, w przypadku którego uzyskano przyspieszenie przetwarzania niezależnie od użytego rzędu modelu. Już jednak dla rzędu modelu równego 3, z jednym tylko wyjątkiem, metoda hybrydowa jest szybsza od

odpowiadającej jej rzędem modelu metody predykcyjnej. Takie wyniki wskazują na to, że wyższa szybkość komponentu substytucyjnego w stosunku do komponentu predykcyjnego przeważa wydłużenie czasu przetwarzania wynikające z wykonywania przełączania zwykle już przy rzędzie modelu równym 3.

W przypadku korpusu *Calgary* średnia bitowa uzyskana algorytmem predykcyjno-substytucyjnym jest bardzo zbliżona do średniej bitowej uzyskanej algorytmem predykcyjnym z jednostkowo większym rzędem modelu. Dla korpusu *Canterbury* poprawa średniej bitowej uzyskana algorytmem predykcyjno-substytucyjnym jest jeszcze wyższa.

W przypadku pliku *XML*, który predykcyjnie należałoby kompresować z użyciem modelu wysokiego rzędu, przewaga algorytmu predykcyjno-substytucyjnego pod względem efektywności kompresji jest ogromna.

Przechodząc do miary uwzględniającej zarówno szybkość jak i efektywność kompresji, należy stwierdzić iż w każdym badanym przypadku dodanie komponentu substytucyjnego pozwoliło osiągnąć poprawę współczynnika przyspieszenia transmisji.

Należy zatem uznać, że przyjęte w proponowanej metodzie podejście hybrydowe okazało się dobre. Nie tylko przyniosło poprawę efektywności kompresji dla wszystkich testowanych zbiorów danych, przy niewielkim spowolnieniu działania, ale także pozwoliło na efektywną kompresję tych typów danych (reprezentowanych przez plik *XML*), w przypadku których model predykcyjny niskiego rzędu jest niewystarczający.

4.4 Porównanie z innymi dostępnymi algorytmami

4.4.1 Testowane algorytmy

W trakcie testów nowy algorytm skonfrontowany został z 16 algorytmami różnego typu. Algorytmy wybrane do testów można podzielić na trzy grupy:

- algorytmy, które dla przyjętego przedziału prędkości transmisji (około 512 kB/s) osiągają najwyższe znane wartości *WPT*: GRZip Fast, Deflate, PPMII;
- algorytmy, które stosują metody hybrydowe odmiennego rodzaju: GRZip, LZIP3, UHARC;
- inne algorytmy, których załączenie w testach uznano za interesujące z racji ich wyróżniających właściwości (LZOP, LZMA), odmiennego od pozostałych sposobu działania (CTW, oparte na BWT), czy tylko ich popularności (Rar).

Wyboru testowanych algorytmów dokonano w przekonaniu że są to najlepsze dostępne algorytmy danej klasy (w chwili wyboru, to jest pod koniec października

2003), bazując na wynikach ujętych w literaturze [GILC02; DEOR03] oraz badaniach własnych Autora.

Wszystkie algorytmy testowano w postaci ich istniejących, w większości przypadków ogólnie dostępnych implementacji. Pojawia się pytanie, czy taki sposób postępowania jest najwłaściwszy. Wydaje się, że tak, gdyż:

- jest to podejście utarte w literaturze przedmiotu (przykładowo: [BLOO96], [BUNT96], [SZKA02a], [DEOR03]);
- większość algorytmów kompresji sformułowana jest bardzo szczegółowo, co mocno ogranicza swobodę programisty;
- w większości przypadków są to programy napisane przez samych autorów algorytmów; można zatem przypuszczać, że są w dużej mierze zoptymalizowane;
- alternatywą byłoby samodzielne zaimplementowanie poszczególnych algorytmów przez Autora niniejszej pracy, co byłoby: żmudne, trudne do wykonania (niektóre algorytmy znane są jedynie z ich istniejących implementacji), a ostateczny rezultat byłby zapewne znacznie gorszy od implementacji oryginalnych.¹⁵

Przyjęty sposób przeprowadzenia badań działa na niekorzyść implementacji nowego algorytmu, ponieważ:

- większość użytych w eksperymentach programów jest doskonała już od dłuższego czasu, stąd można domniemywać, iż poziom ich optymalizacji dalece przewyższa świeżo co napisany program PLZ;
- trzy najbardziej istotne dla wyników porównania programy: GRZip, PPMd oraz Zip, skompilowane zostały przy użyciu kompilatora *Intel C*; jest to kompilator powszechnie znany z bardzo dobrej optymalizacji, co potwierdzają pomiary czasu działania programów skompilowanych tym i innymi kompilatorami.

Algorytmy hybrydowe. Listę użytych w eksperymentach programów (tabela 4.10) otwiera GRZip. Zaimplementowano w nim substytucyjno-transformacyjną metodę hybrydową. W pracy przedstawiono wyniki dla obu dostępnych w programie algorytmów kompresji. Bardziej interesujący jest algorytm trybu *Fast*, gdyż łączy on dobrą efektywność kompresji z niezłą szybkością działania.

Algorytm LZIP3 testowano w oryginalnej implementacji C. Blooma. Trzeba tu zaznaczyć, że jest to prawdopodobnie najstarszy z testowanych programów, co może w pewnym stopniu usprawiedliwiać jego stosunkowo słabe rezultaty, aczkolwiek nie znaleziono lepszej implementacji tego algorytmu.

¹⁵ Mało prawdopodobne jest bowiem, iżby Autor niniejszej pracy zdołał lepiej zaimplementować poszczególne algorytmy, niż udało się to zrobić ich własnym autorom.

Algorytm hybrydowy użyty w programie UHARC (tryb „mx”) jest stanowczo zbyt powolny, by uzyskane przezeń rezultaty liczyły się, gdy stosuje się miarę uwzględniającą czas działania. Jednak wyniki tego algorytmu są interesujące, gdyż jest to algorytm hybrydowy o najwyższej efektywności kompresji. W badaniach nie uwzględniono implementacji ROLZ w programie RK; podobnie jak UHARC jest to algorytm bardzo powolny i osiągający zbliżoną do niego efektywność kompresji, głównym powodem pominięcia go w testach była niepewność co do możliwości wyłączenia wszystkich stosowanych przez program form przetwarzania wstępnego.

Tabela 4.10. Algorytmy i programy uwzględnione w testach

Typ	Algorytm	Program	Opcje	Źródło
Hybrydowe	LZP+BWT	GRZip 0.7.3		[GREB03]
	LZP+ST	GRZip 0.7.3	-f	
	LZP3	LZP3o2 1.1		www.cbloom.com
	LZP+PPM	UHARC 0.5np	-mx	[HERK02]
Substytucyjne	LZMA	7-zip 2.30	-m0=LZMA	[PAWŁ03]
	LZO	LZOP 1.01		[OBER03]
	Rar	Rar 3.20		[ROSZ03]
	Deflate	Zip 2.3	-4	www.info-zip.org
	Deflate	Zip 2.3	-6	
Predykcyjne	CTW	CTW 0.1	-d4	www.ele.tue.nl/ctw
	PPMII	PPMd v.I	-o2 -m32	ftp.elf.stuba.sk/ pub/pc/pack
	PPMII	PPMd v.I	-o3 -m32	
	PPMII	PPMd v.I	-o4 -m32	
	PPMII	PPMd v.I	-o8 -m32	
BWT	BWT+MTF	Bzip2 1.0.1		[SEWA00]
	BWT+WFC	SDC		[DEOR03]

Algorytmy substytucyjne. Algorytm LZMA osiąga najwyższą wśród algorytmów substytucyjnych efektywność kompresji. W eksperymentach użyto jego oryginalnej implementacji (7-zip).

Z kolei algorytm LZO jest uznawany za najszybszy algorytm substytucyjny ogólnego przeznaczenia. Interesujące było sprawdzenie czy bardzo wysoka szybkość usprawiedliwia niską efektywność kompresji tego algorytmu. W eksperymentach użyto jego oryginalnej implementacji (LZOP).

Program Rar jest jednym z najpopularniejszych uniwersalnych narzędzi kompresji. Użyty w nim algorytm substytucyjny jest dość efektywny, działając znacznie szybciej od LZMA. Testowano standardowy tryb działania (*Normal*). Tryb najlepszej kompresji wykorzystuje bowiem algorytm predykcyjny (ustępujący PPMd), z kolei tryb

najszybszy tylko nieznacznie poprawia szybkość działania przy bardzo ostrym spadku jej efektywności kompresji.

Algorytm Deflate to standard bezstratnej kompresji danych. Do testów wybrano jego implementację w programie Zip. Co prawda program PKZip uzyskiwał w testach wstępnych nieco lepszą efektywność kompresji, był jednak wolniejszy. Testowano dwa tryby działania: '4' i domyślny '6', które w testach wstępnych okazały się najbardziej obiecujące pod względem osiąganych wartości *WPT*. Uznano, że zamiast trybu '1' lepszym przykładem szybkiego algorytmu substytucyjnego będzie LZO, podobnie zamiast trybu '9', lepszym przykładem efektywnego algorytmu substytucyjnego będzie LZMA.

Algorytmy predykcyjne. CTW to przykład algorytmu predykcyjnego o odmiennej od PPM zasadzie działania. Przykład ten dobitnie pokazuje przepaść pomiędzy najlepszymi algorytmami, na których skupiono się w testach, a pominiętą resztą algorytmów. Dla przyspieszenia działania, wysokość drzew kontekstowych ograniczono do 4 poziomów.

PPMII to najlepszy obecnie algorytm wywodzący się z PPM, pod względem stosunku efektywności kompresji do czasu działania. W eksperymentach użyto rzędów modelu 2, 3 i 4, jako najbardziej interesujących dla przyjętych kryteriów porównawczych. Dodatkowo przetestowano model rzędu 8, dla pokazania jak spada szybkość kompresji dla modeli wyższych rzędów. Pominęto najbardziej obecnie zaawansowaną wersję algorytmu cPPMII, uznając UHARC za wystarczający przykład bardzo powolnego algorytmu o wysokiej efektywności kompresji. W eksperymentach użyto oryginalnej implementacji PPMII D. Szkarina (PPMd).

Algorytmy transformacyjne reprezentowane są przez dwie implementacje. Klasyczne połączenie transformacji BWT i MTF zbadano w postaci najpopularniejszego obecnie programu tego typu – Bzip2. Użyto domyślnej wielkości bloku. Najnowszy odpowiednik MTF – efektywny WFC przetestowano na przykładzie autorskiego programu SDC S. Deorowicza. Użyto domyślnego trybu pracy.

W programach, które posiadały opcje motywowanego semantycznie przetwarzania wstępnego, zostały one wyłączone, dla zachowania porównywalności rezultatów.

4.4.2 Porównanie efektywności kompresji

Pierwszą badaną cechą jest efektywność kompresji. Tabela 4.11 zawiera średnią bitową zmierzoną dla poszczególnych algorytmów i plików.

Tabela 4.11. Średnia bitowa (w bpc) osiągnięta przez testowane algorytmy dla poszczególnych plików

	Algorytm	<i>Calgary</i>	<i>Canterbury</i>	<i>Txt</i>	<i>Exe</i>	<i>Xml</i>
Nowe	PLZ 1	2,923	1,833	3,270	3,863	0,109
	PLZ 2	2,346	1,550	2,819	3,552	0,079
	PLZ 3	2,138	1,431	2,554	3,360	0,079
	PLZ 3+	2,111	1,387	2,522	3,301	0,080
Hybrydowe	GRZip	2,048	1,542	2,439	3,600	0,079
	GRZip -f	2,159	1,357	2,591	3,672	0,081
	LZP	2,557	2,099	3,019	4,029	0,098
	UHARC mx	1,880	1,168	2,433	3,100	0,067
Substytucyjne	LZMA	2,116	1,384	2,781	3,242	0,135
	LZOP	3,895	3,312	4,962	5,039	0,432
	Rar	2,298	1,630	2,925	3,547	0,119
	Zip -4	2,731	2,240	3,405	3,987	0,310
	Zip -6	2,616	2,096	3,246	3,914	0,277
Predykcyjne	CTW	2,095	1,539	2,466	3,633	0,356
	PPMd o2	2,674	1,795	3,004	3,834	0,543
	PPMd o3	2,257	1,554	2,613	3,617	0,387
	PPMd o4	2,055	1,484	2,427	3,502	0,344
	PPMd o8	1,939	1,472	2,318	3,409	0,273
BWT	Bzip2	2,191	1,619	2,590	3,810	0,114
	SDC	2,065	1,459	2,446	3,573	0,071

W przypadku korpusu *Calgary* (patrz także rysunek 4.2) najlepszą średnią bitową uzyskały algorytmy różnego rodzaju (UHARC, PPMd, SDC, CTW). Algorytmy PLZ (z wyjątkiem PLZ 1) uplasowały się w środkowej części rankingu.

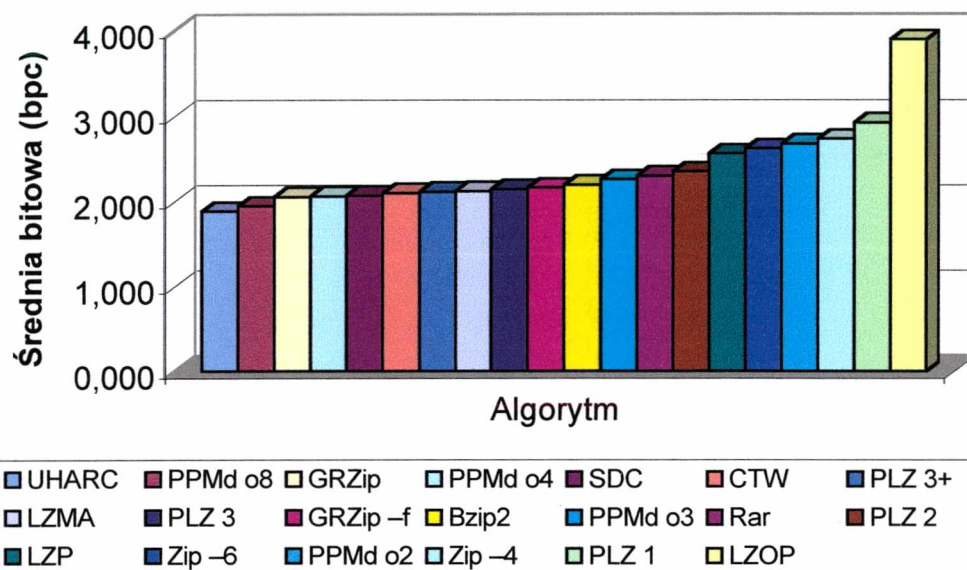
W przypadku korpusu *Canterbury* (patrz także rysunek 4.3) ponownie najlepszą średnią bitową uzyskał program UHARC. Na uwagę zasługuje przesunięcie się do czołówki rankingu algorytmów PLZ rzędu 3.

Sytuację tę wyjaśniają nieco rezultaty zanotowane dla poszczególnych dodatkowych plików testowych. W przypadku przykładowego pliku tekstowego *TXT*, najlepszy wynik zanotowano dla PPMd z modelem rzędu 8. Jest to wynik spodziewany, jako że wysoka efektywność algorytmów z rodziny PPM dla tekstu w języku naturalnym jest ich powszechnie znaną cechą. Co istotne, algorytmy PLZ przewyższają efektywnością algorytmy PPM o odpowiadającym im rzędzie modelu. Świadczy to o słuszności przyjętego modelu predykcyjno-substytucyjnego.

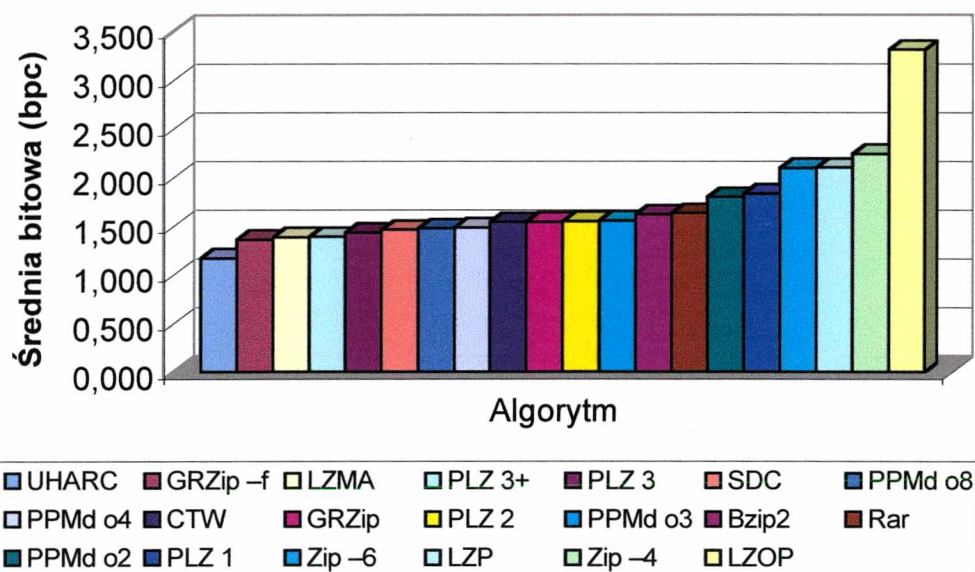
W przypadku plików *EXE* i *XML* na uwagę zasługuje bardzo wysoka efektywność osiągnięta przez PLZ o modelu rzędu 3, a w drugim przypadku także 2. Warto zwrócić

uwagę na bardzo dużą różnicę efektywności pomiędzy różnymi algorytmami występującą w przypadku pliku *XML*.

Korpus *Calgary* jest w większym stopniu zdominowany przez typowe dane tekstowe aniżeli korpus *Canterbury*. Stąd wyniki zanotowane w przypadku korpusu *Calgary* są znacznie bliższe wynikom dla pliku *TXT*, aniżeli ma to miejsce w przypadku korpusu *Canterbury*, w którym większą rolę odgrywają dane binarne.



Rys. 4.2. Średnia bitowa osiągnięta przez testowane algorytmy dla korpusu *Calgary*



Rys. 4.3. Średnia bitowa osiągnięta przez testowane algorytmy dla korpusu *Canterbury*

4.4.3 Porównanie szybkości działania

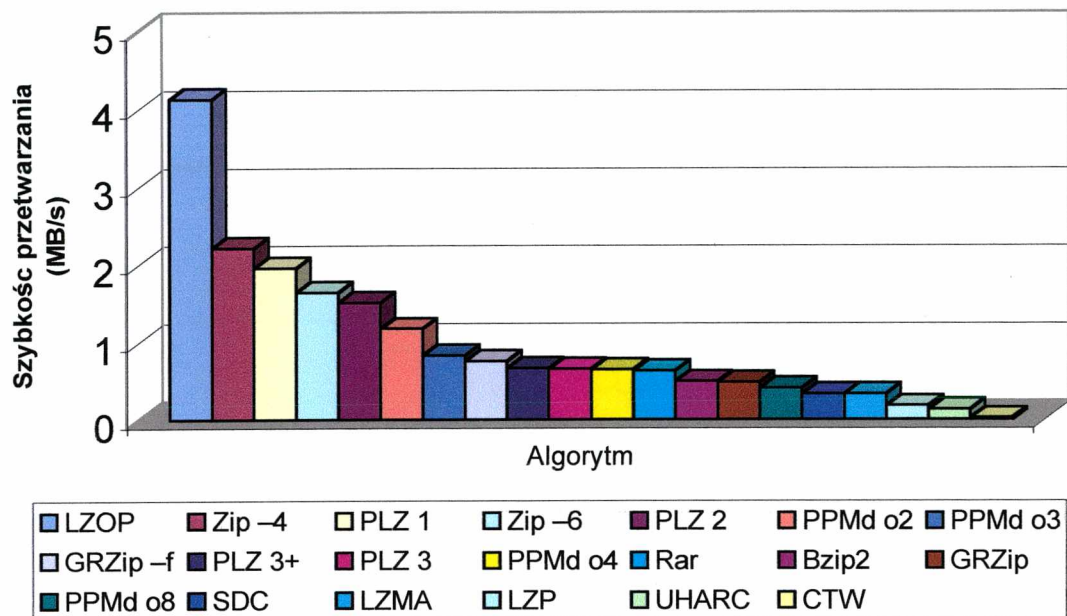
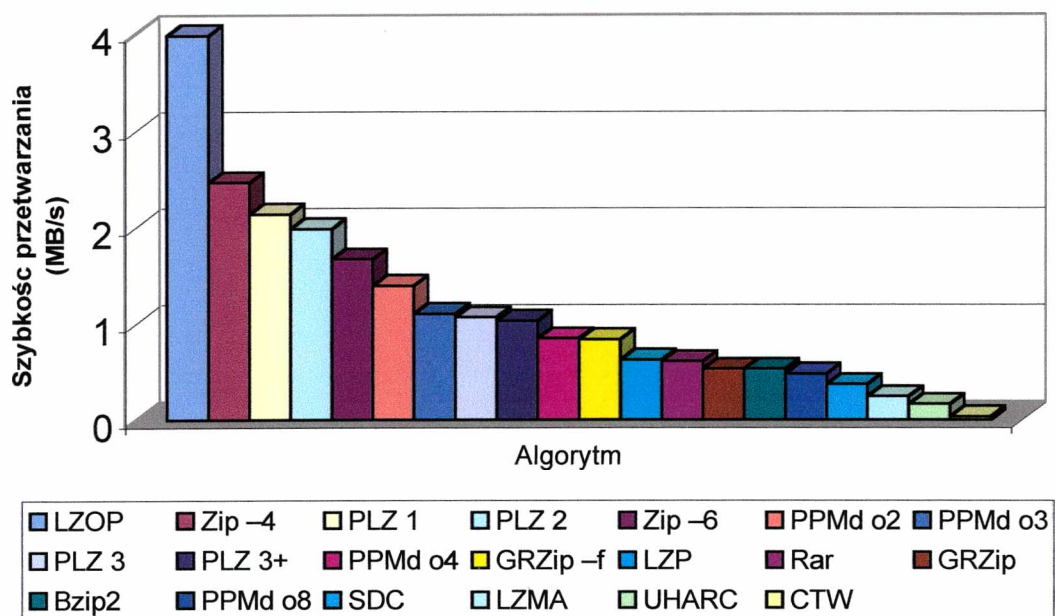
Drugą badaną cechą jest szybkość działania. Jako że w dalszych badaniach będziemy posługiwać się miarą transmisji danych, dla której stosunek czasu kompresji do dekompresji jest nieistotny, również pomiary szybkości przedstawione są dla obu tych operacji łącznie. Czas przetwarzania, w oparciu o który wyliczono szybkość działania testowanych algorytmów, jest zatem sumą czasu kompresji i czasu dekompresji danych. Tabela 4.12 zawiera szybkość przetwarzania danych zmierzona w MB/s dla poszczególnych algorytmów i plików.

W przypadku korpusów *Calgary* i *Canterbury* (patrz także rysunki 4.4 i 4.5), jedno z najlepszych wyników zaobserwowano dla PLZ o rzędzie modelu 1 i 2. Bezkonkurencyjny jest tu jednak LZOP, wyraźnie górujący szybkością działania nad pozostałymi algorytmami.

W przypadku pliku *TXT*, PLZ o modelu rzędu 2 ustępuje algorytmowi PPM o identycznym rzędzie modelu. Natomiast w przypadku pliku *EXE*, sytuacja jest odwrotna: PPM wyraźnie ustępuje PLZ.

Tabela 4.12. Szybkość działania zmierzona dla testowanych algorytmów na poszczególnych plikach (MB/s)

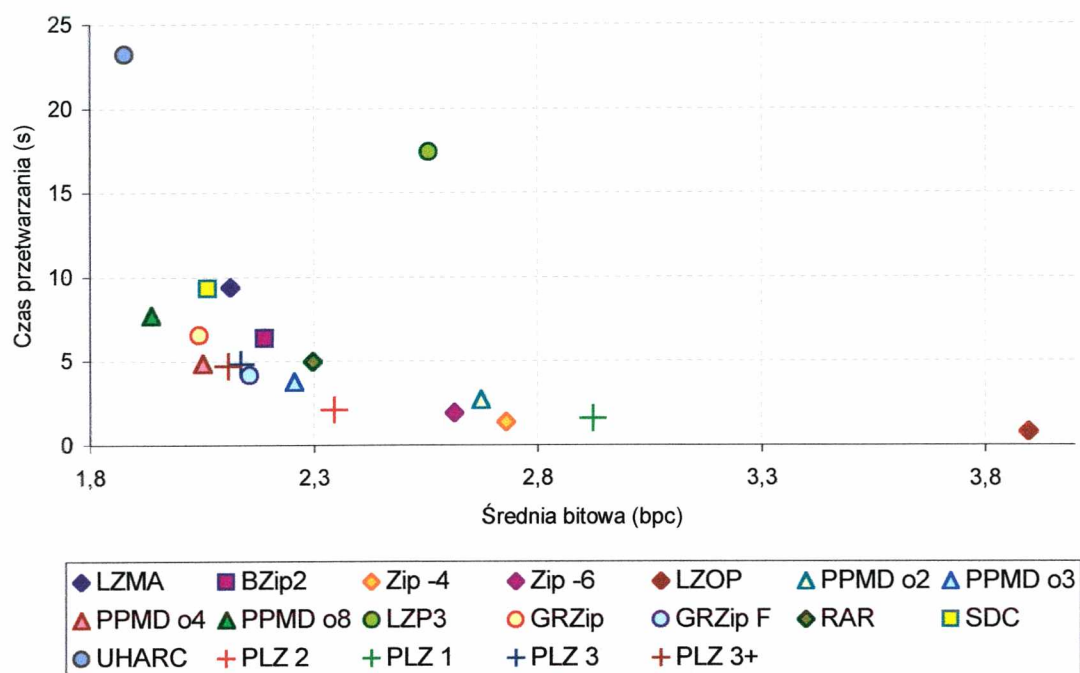
	Algorytm	<i>Calgary</i>	<i>Canterbury</i>	<i>Txt</i>	<i>Exe</i>	<i>Xml</i>
Nowe	PLZ 1	1,958	2,129	0,798	0,862	3,361
	PLZ 2	1,513	1,976	0,697	0,578	3,355
	PLZ 3	0,654	1,060	0,432	0,345	2,838
	PLZ 3+	0,667	1,024	0,431	0,347	2,795
Hybrydowe	GRZip	0,483	0,525	0,327	0,303	2,820
	GRZip –f	0,757	0,831	0,444	0,409	2,794
	LZP	0,181	0,620	0,338	0,027	2,812
	UHARC mx	0,136	0,161	0,105	0,076	0,963
Substytucyjne	LZMA	0,336	0,240	0,245	0,335	0,838
	LZOP	4,128	3,980	1,087	1,412	3,385
	Rar	0,631	0,605	0,445	0,509	1,604
	Zip –4	2,214	2,457	0,888	0,983	2,799
	Zip –6	1,639	1,662	0,702	0,847	2,818
Predykcyjne	CTW	0,024	0,031	0,025	0,024	0,042
	PPMd o2	1,175	1,388	0,751	0,364	1,981
	PPMd o3	0,833	1,097	0,614	0,296	2,098
	PPMd o4	0,646	0,843	0,446	0,265	2,109
	PPMd o8	0,410	0,476	0,297	0,236	1,982
BWT	Bzip2	0,496	0,525	0,325	0,326	0,230
	SDC	0,338	0,367	0,258	0,223	0,267

Rys. 4.4. Szybkość działania zmierzona dla testowanych programów na korpusie *Calgary*Rys. 4.5. Szybkość działania zmierzona dla testowanych programów na korpusie *Canterbury*

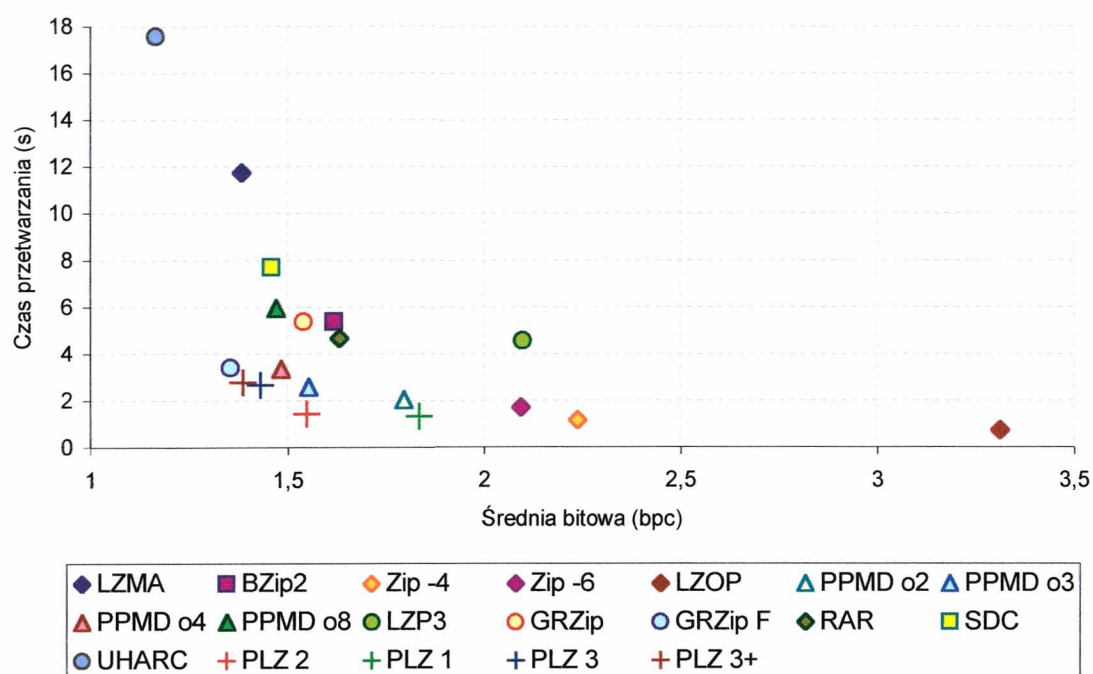
Mocno nadmiarowy plik *XML* większość algorytmów kompresuje znacznie szybciej. Wyjątkiem są algorytmy oparte na metodach transformacyjnych. Pliki takie jak *XML* są trudne do posortowania, co znacznie wydłuża czas działania algorytmów opartych na *BWT*. Za to szybkość algorytmu predykccyjno-substytucyjnego na takich danych jest wyśmienita: jedynie minimalnie ustępuje on najszybszemu LZOP.

4.4.4 Porównanie dwukryterialne

Znając szybkość działania i efektywność kompresji poszczególnych algorytmów można przystąpić do ich oceny uwzględniającej oba te kryteria jednocześnie.



Rys. 4.6. Czas przetwarzania a efektywność kompresji *Calgary Corpus*



Rys. 4.7. Czas przetwarzania a efektywność kompresji *Canterbury Corpus*

Rysunki 4.6 i 4.7 pokazują czas przetwarzania i średnią bitową testowanych algorytmów (pominięto drastycznie odstający CTW). Im bliżej umiejscowiono dany algorytm bliżej punktu przecięcia obu osi, tym bliższy jest on optimum.

W przypadku obu korpusów algorytmy PLZ2, PLZ3 i PLZ3+ są optymalne w sensie Pareto. Oznacza to, że nie ma innych algorytmów, które osiągałyby tę samą efektywność kompresji przy krótszym czasie przetwarzania (lub lepszą efektywność kompresji przy tym samym czasie przetwarzania). W przypadku korpusu *Canterbury*, optymalny w sensie Pareto jest również algorytm PLZ1.

Powyższe obserwacje uznać można za wystarczające do pozytywnej weryfikacji przydatności nowej metody. Wyniki eksperymentów pokazały niezbicie, iż istnieje niepusty zbiór sytuacji, w których najlepszym dostępnym algorytmem jest tu zaproponowany algorytm predykcyjno-substytucyjny.

4.4.5 Porównanie przyspieszenia transmisji dla założonej prędkości

Pokazano już, że nowy algorytm jest optymalny w sensie Pareto. Wiadomo więc, że istnieje hipotetyczna sytuacja, w której stanowi on najlepsze dostępne rozwiązanie. Z praktycznego punktu widzenia interesujący może być jednak tylko pewien wycinek rzeczywistości, w którym stosunek wagi przywiązywanej do czasu działania, do wagi przywiązywanej do efektywności kompresji wyznacza moc obliczeniowa używanego sprzętu i przepustowość dostępnego łącza.

U podstaw tworzenia nowego algorytmu leżało przekonanie, że docelowo będzie to algorytm przede wszystkim szybki, lecz nie ustępujący wiele efektywności istniejącym rozwiązaniom. Dobrą miarą nadającą się do zbadania jego praktycznej przydatności będzie współczynnik przyspieszenia transmisji przy założonej prędkości transmisji 512 kb/s.

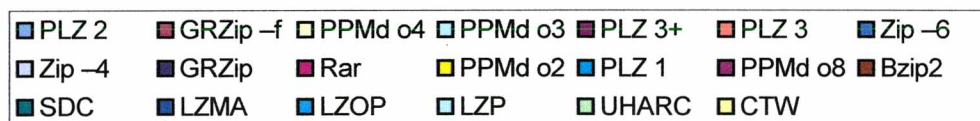
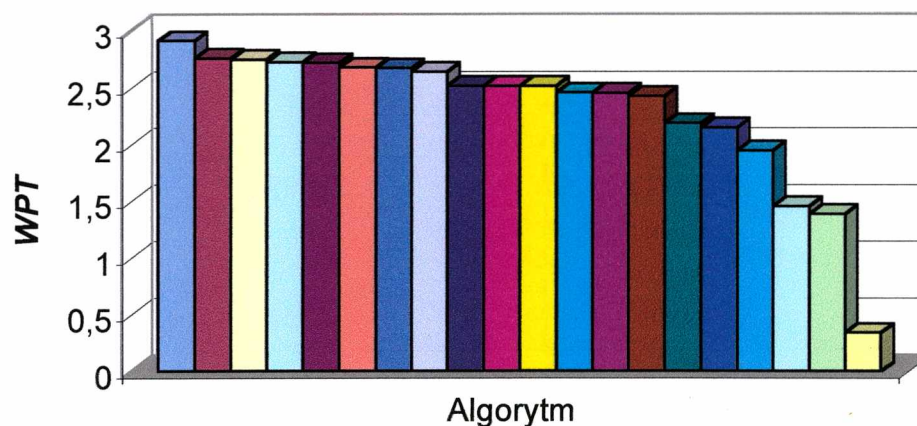
Przypomnijmy, że prędkość 512 kb/s została dobrana do specyfikacji komputera testowego tak, by nie akcentować przesadnie ani czasu działania ani efektywności kompresji. Kształtowanie się współczynnika przyspieszenia transmisji przy innych prędkościach transmisji zostanie omówione w dalszej części pracy.

Choć używa się tu miary odnoszącej się do transmisji danych, znaczenie uzyskanych pomiarów bynajmniej nie ogranicza się do sytuacji transmisji danych z założoną prędkością. Odpowiada ona bowiem każdej rzeczywistej sytuacji, także dostępu do przechowywanych danych, w której użytkownik ceni sobie czas wykonywania i efektywność kompresji w takich samych proporcjach jak przy transmisji danych z taką właśnie prędkością.

Wartości *WPT* wyliczone dla zmierzonej średniej bitowej i czasu działania, dla poszczególnych algorytmów i plików, przedstawia tabela 4.13.

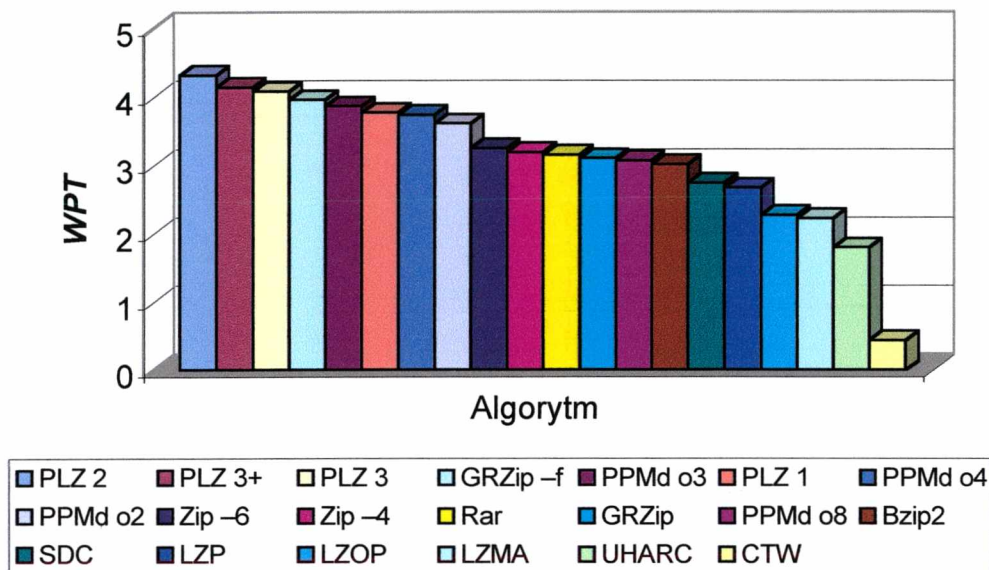
Tabela 4.13. Współczynnik przyspieszenia transmisji testowanych algorytmów przy prędkości transmisji 512 kb/s

	Algorytm	<i>Calgary</i>	<i>Canterbury</i>	<i>Txt</i>	<i>Exe</i>	<i>Xml</i>
Nowe	PLZ 1	2,453	3,768	1,997	1,753	29,929
	PLZ 2	2,910	4,319	2,199	1,761	33,673
	PLZ 3	2,675	4,081	2,089	1,613	30,070
	PLZ 3+	2,714	4,141	2,106	1,635	29,683
Hybrydowe	GRZip	2,514	3,104	1,951	1,477	29,989
	GRZip -f	2,756	3,959	2,086	1,587	29,599
	LZP	1,451	2,671	1,724	0,341	27,886
	UHARC mx	1,383	1,794	1,070	0,798	13,056
Substytucyjne	LZMA	2,147	2,222	1,604	1,637	10,470
	LZOP	1,944	2,271	1,438	1,446	13,403
	Rar	2,513	3,155	1,917	1,716	17,838
	Zip -4	2,637	3,191	1,962	1,733	15,864
	Zip -6	2,668	3,250	1,965	1,729	17,014
Predykcyjne	CTW	0,337	0,436	0,345	0,316	0,621
	PPMd o2	2,512	3,610	2,120	1,491	9,746
	PPMd o3	2,721	3,866	2,266	1,462	12,374
	PPMd o4	2,744	3,735	2,185	1,438	13,313
	PPMd o8	2,451	3,068	1,934	1,399	14,692
BWT	Bzip2	2,423	3,012	1,876	1,452	3,336
	SDC	2,183	2,738	1,763	1,331	3,923



Rys. 4.8. Współczynnik przyspieszenia transmisji testowanych algorytmów przy prędkości transmisji 512 kb/s dla korpusu *Calgary*

W przypadku korpusu *Calgary* najlepszym z testowanych algorytmów okazał się PLZ o modelu rzędu 2 (rysunek 4.8). Identycznie jest w przypadku korpusu *Canterbury*, gdzie dodatkowo drugie i trzecie miejsce w rankingu przypadło PLZ o modelu rzędu 3, odpowiednio: z estymacją długości zapisu i bez niej (rysunek 4.9).



Rys. 4.9. Współczynnik przyspieszenia transmisji testowanych algorytmów przy prędkości transmisji 512 kb/s dla korpusu *Canterbury*

Jak należało się spodziewać, najwyższą wartość *WPT* dla pliku *TXT* odnotowano w przypadku programu PPMd. Ale już bezpośrednio za nim uplasował się program PLZ z modelem rzędu 2. Świadczy to o tym, że swą niższą efektywność kompresji w przypadku danych tego rodzaju, PLZ rekompensuje szybkością kompresji.

W przypadku pliku *EXE*, najwyższą wartość *WPT* osiąga program PLZ z modelem rzędu 2 i 1. Nieco gorsze rezultaty odnotowano dla szeregu algorytmów substytucyjnych. Algorytmy predykcyjne wyraźnie im w tym przypadku ustępują.

Plik *XML* stanowi prawdziwe pole do popisu dla metody predykcyjno-substytucyjnej. Siedem najlepszych wyników odnotowano dla siedmiu testowanych algorytmów hybrydowych. W ich gronie, na największą uwagę zasługuje PLZ z modelem rzędu 2, który jest wyraźnie lepszy od pozostałych.

Aby odpowiedzieć na pytanie o ile lepszy okazał się algorytm PLZ 2 od dotąd istniejących, przygotowano tabelę 4.14. Poszczególne pozycje tej tabeli odpowiadają procentowej poprawie *WPT* osiągniętej przez PLZ 2 dla programu wskazanego przez wiersz tabeli w przypadku pliku testowego określonego przez kolumnę tabeli.

Tabela 4.14. Procentowa poprawa WPT poprzez użycie PLZ 2 w stosunku do innych testowanych algorytmów przy założonej prędkości transmisji 512 kB/s

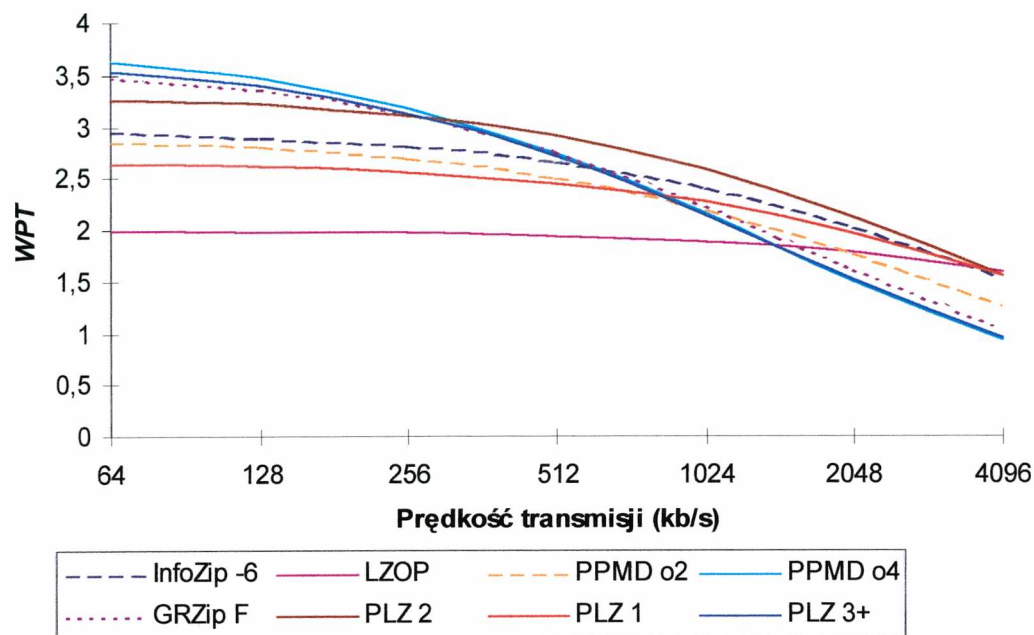
	Algorytm	<i>Calgary</i>	<i>Canterbury</i>	<i>Txt</i>	<i>Exe</i>	<i>Xml</i>
Hybrydowe	GRZip	15,75%	39,14%	12,71%	19,23%	12,28%
	GRZip –f	5,59%	9,09%	5,42%	10,96%	13,76%
	LZP	100,55%	61,70%	27,55%	416,42%	20,75%
	UHARC mx	110,41%	140,75%	105,51%	120,68%	157,91%
Substytucyjne	LZMA	35,54%	94,37%	37,09%	7,57%	221,61%
	LZOP	49,68%	90,21%	52,92%	21,76%	151,24%
	Rar	15,80%	36,89%	14,71%	2,62%	88,77%
	Zip –4	10,35%	35,35%	12,08%	1,62%	112,26%
	Zip –6	9,07%	32,89%	11,91%	1,85%	97,91%
Predykcyjne	CTW	763,50%	890,60%	537,39%	457,28%	5322,4%
	PPMd o2	15,84%	19,64%	3,73%	18,13%	245,51%
	PPMd o3	6,95%	11,73%	-2,97%	20,49%	172,13%
	PPMd o4	6,05%	15,64%	0,64%	22,46%	152,93%
	PPMd o8	18,73%	40,78%	13,70%	25,88%	129,19%
BWT	Bzip2	20,10%	43,39%	17,22%	21,28%	909,38%
	SDC	33,30%	57,74%	24,73%	32,31%	758,35%

4.4.6 Porównanie dla różnych prędkości transmisji

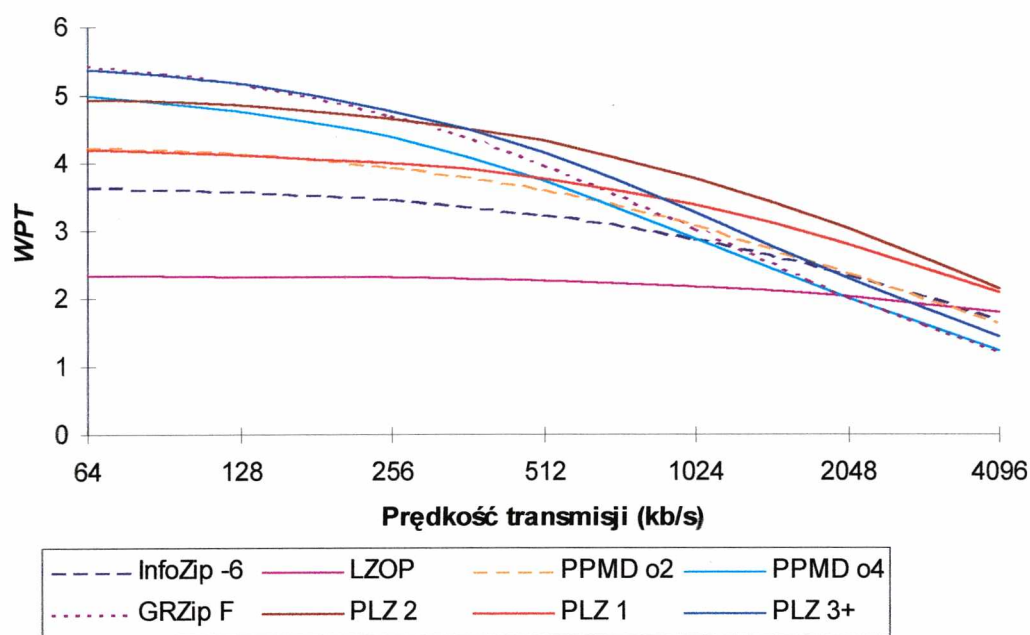
Interesującą kwestią pozostaje jak kształtuje się *WPT* w zależności od przyjętej prędkości transmisji. Wyjaśnić ją, przynajmniej dla zbadanego zakresu prędkości, pomogą wyniki podane niżej. Należy przy tym pamiętać, że zarówno w przypadku programu PLZ jak i najbardziej wartościowych spośród porównywanych algorytmów, programy i ich opcje dobrano tak, by uzyskiwane wyniki były jak najlepsze dla założonej prędkości transmisji 512 kb/s. Oznacza to, że dla niektórych (w szczególności najniższych) spośród prędkości rozpatrywanych poniżej, lepsze wyniki można uzyskać stosując inne, nieujęte tu programy lub kombinacje parametrów użytkownika.

Ze względu na konieczność zachowania przejrzystości rysunków, umieszczono na nich tylko 8 spośród testowanych algorytmów, a mianowicie takie których rezultaty uznano za najbardziej interesujące; pominięto algorytmy wyraźnie odstające niekorzystnie oraz te, których rezultaty w dużej części nakładały się na inne podobnego typu, tu ujęte. Rysunek 4.10 pokazuje wyniki uzyskane dla korpusu *Calgary*, natomiast rysunek 4.11 wyniki uzyskane dla korpusu *Canterbury*.

Oba rysunki pokazują, że proponowane algorytmy sprawdzają się także dla innych prędkości transmisji. Dla dużej części testowanego zakresu prędkości, bardzo dobre wyniki uzyskuje PLZ 2. Dla prędkości wyższych na znaczeniu zyskuje PLZ 1; dla prędkości niższych zaś PLZ 3+.



Rys. 4.10. Przyspieszenie transmisji przy różnych założonych prędkości transmisji (*Calgary Corpus*)



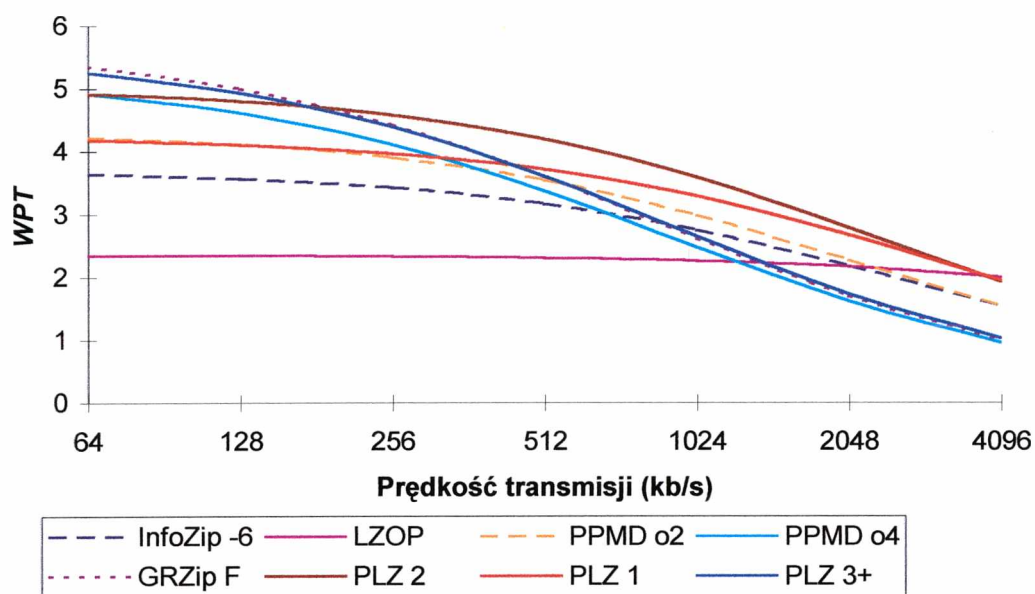
Rys. 4.11. Przyspieszenie transmisji przy różnych założonych prędkości transmisji (*Canterbury Corpus*)

4.4.7 Porównanie dla różnych środowisk testowych

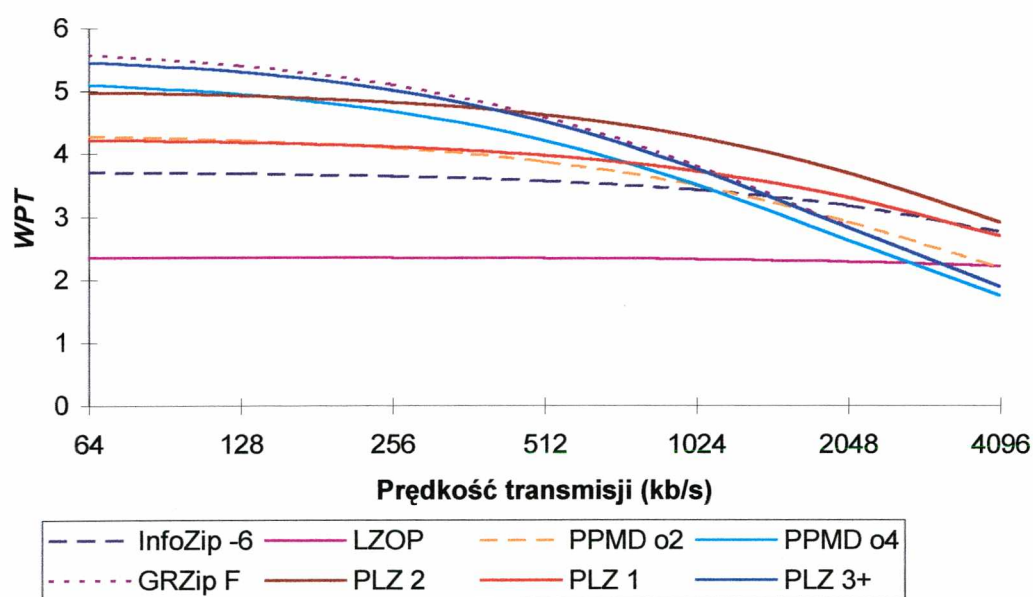
Dotychczasowe eksperymenty prowadzone były tylko w jednym środowisku testowym. Powstaje zatem pytanie, w jakim stopniu uzyskane wyniki można uogólniać na sprzęt o innej mocy obliczeniowej.

Gdyby szybkość działania poszczególnych algorytmów zależała wyłącznie od częstotliwości, z jaką taktowany jest główny procesor, wystarczyłoby uzyskane wyniki odpowiednio przeskalować. Jednak ze względu na to, że zmiana wydajności dotyczy różnych elementów środowiska testowego (procesora, pamięci) w różnym stopniu, tak samo różnie może one wpłynąć na czas działania poszczególnych algorytmów.

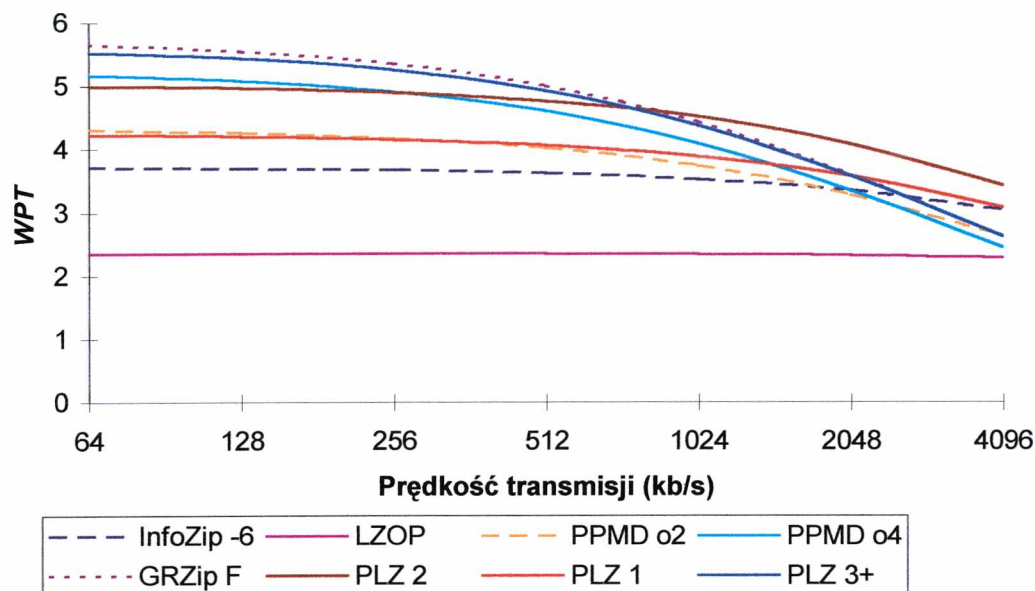
Poniżej zamieszczono rezultaty pomiarów współczynnika przyspieszenia transmisji wykonanych na różnych platformach sprzętowych z użyciem korpusu *Canterbury*.



Rys. 4.12. Przyspieszenie transmisji w środowisku Celeron 800 (*Canterbury Corpus*)



Rys. 4.13. Przyspieszenie transmisji w środowisku Athlon 1600 (*Canterbury Corpus*)



Rys. 4.14. Przyspieszenie transmisji w środowisku Athlon 2400 (*Canterbury Corpus*)

Rysunek 4.12 prezentuje wyniki uzyskane na platformie testowej o następującej specyfikacji technicznej: procesor Intel *Celeron* 800 MHz, pamięć 128 MB SDR 100 MHz, dysk twardy Seagate 15 GB, system operacyjny Microsoft *Windows* XP.

Specyfikacja techniczna platformy testowej, dla której uzyskane wyniki przedstawia rysunek 4.13 jest następująca: procesor AMD *Athlon XP* 1600 (1400 MHz), pamięć 224 MB DDR 266 MHz, dysk twardy Western Digital 20 GB, system operacyjny Microsoft *Windows* XP.

Z kolei rysunek 4.14 prezentuje wyniki uzyskane na platformie testowej o następującej specyfikacji technicznej: procesor AMD *Athlon XP* 2400 (2000 MHz), pamięć 512 MB DDR 400 MHz, dysk twardy Seagate *Barracuda* 80 GB, system operacyjny Microsoft *Windows* XP.

Tak jak należało się tego spodziewać, przedział prędkości transmisji, w którym najlepszym dostępnym rozwiązaniem jest algorytm predykcyjno-substytucyjny w stosunku do rysunku 4.11 uległ przesunięciu na wykresie w lewo dla pierwszego (wolniejszego) środowiska, w prawo zaś dla dwóch pozostałych (szybszych).

Przypatrując się przebiegowi krzywych reprezentujących poszczególne algorytmy, wydaje się że kluczowy wpływ na zmiany ich wzajemnego położenia ma różnica pomiędzy szybkością procesora a szybkością pamięci dla poszczególnych środowisk testowych. Im różnica ta jest mniejsza, tym stosunkowo lepiej wypadają algorytmy szybkie, w których przypadku „wąskim gardłem” jest szybkość dostępu do danych. Im różnica ta jest większa, tym stosunkowo lepiej wypadają algorytmy skomplikowane, w których przypadku „wąskim gardłem” jest szybkość wykonywania obliczeń.

Podsumowanie

W niniejszej pracy podjęto temat bezstratnych metod kompresji danych. Jest to ważny obszar badań naukowych, metody te pozwalają bowiem zmniejszyć objętość przechowywanych i przesyłanych danych. Przynosi to wymierne korzyści, które wyrażają się przede wszystkim w poprawie parametrów użytkowych systemu, osiągniętej dzięki lepszemu wykorzystaniu istniejącej infrastruktury technicznej.

Szczególnie interesującym zagadnieniem, do którego wielu badaczy nie przykładają należytej wagi, jest w jakim stopniu korzyść z zastosowania kompresji (zmniejszenie objętości danych) równoważy jej podstawowy koszt (czas wykonywania kompresji i dekompresji). W związku z tym, w niniejszej pracy algorytmy kompresji oceniano z jednoczesnym uwzględnieniem obu tych kryteriów.

Dostępne są bardzo szybkie algorytmy substytucyjne, których efektywność jest jednak niska. Dostępne są też wysoce efektywne algorytmy predykcyjne, które są jednak bardzo powolne w działaniu.

Istniała zatem naturalna potrzeba zbadania, czy właściwie zbudowany algorytm łączący oba te podejścia może przewyższyć każde z nich w sytuacji, w której liczy się zarówno efektywność kompresji, jak i czas jej wykonywania.

Na zasadność połączenia metody substytucyjnej i predykcyjnej wskazywały przede wszystkim trzy przesłanki: łatwość ich połączenia, różnica w sposobie modelowania przez nie źródła danych oraz różnica w szybkości ich działania.

Za lepszą formę realizacji metody predykcyjno-substytucyjnej uznano schemat jednoprzebiegowy, w którym dla każdego kodowanego elementu zbioru danych wykonuje się naprzemiennie wybór metody składowej i kodowanie z jej użyciem, w odróżnieniu od schematu dwuprzebiegowego, w którym to cały zbiór danych kompresowany jest najpierw metodą substytucyjną, a następnie predykcyjną.

Jedną z niewątpliwych zalet schematu jednoprzebiegowego jest możliwość zaimplementowania w nim estymacji długości zapisu.

Najważniejsze elementy algorytmu predykcyjno-substytucyjnego to: algorytm przełączający pomiędzy dwiema dostępnymi metodami, komponent substytucyjny, komponent predykcyjny oraz koder statystyczny.

W przypadku algorytmu przełączającego, klasyczną koncepcję długości substytucji, dla której odbywa się przełączanie między metodami, rozszerzono o pojęcia minimalnej i zadowalającej długości substytucji, które uzasadniono odwołując się do pewnych własności źródła danych, występujących przy realistycznych założeniach. Opracowano szybki sposób estymacji długości zapisu gwarantujący optymalność wyboru metody w sensie lokalnym. Polega on na estymacji opłacalności podmiany już po predykcyjnym zakodowaniu frazy, co przekłada się na tylko niewielkie spowolnienie algorytmu w stosunku do jego wersji bez estymacji długości zapisu.

Komponent substytucyjny określić można jako „kontekstowy LZ77”. Szczególnie ważnym oryginalnym pomysłem jest w nim użycie słowników rzędu 0 o specjalnym trybie działania, służących poprawie efektywności kompresji. Są to: słownik *cache* oraz słownik dystansów, zawierające odpowiednio absolutne i relatywne odwołania do fraz, które zostały ostatnio podmienione, niezależnie od ich kontekstu.

Komponent predykcyjny opiera się na algorytmie PPMC (z modelem niskiego rzędu), wzbogaconym o pewną liczbę usprawnień mających za zadanie poprawić jego szybkość działania. Częściowo zrezygnowano w nim z wykluczeń symboli; dodano drugorzędną estymację ucieczek (uzyskano poprawę efektywności kompresji), ale wykonywaną tylko dla kontekstów „niepewnych”. Istotnym osiągnięciem jest zmieniony sposób przechowywania modelu w pamięci – jest to poprawa nie tylko w stosunku do oryginalnego PPMC, ale także w stosunku do ostatnich osiągnięć w tej dziedzinie wniesionych z algorytmem PPMII.

Koder statystyczny jest implementacją algorytmu D. Subbotina, poszerzoną o pewne dodatkowe funkcje wymagane ze względu na zastosowany sposób kodowania przełącznika i estymację długości zapisu.

Aby zbadać nowy algorytm empirycznie, zaimplementowano go w postaci doświadczalnego programu nazwanego PLZ. Analiza dwukryterialna zmierzonych czasów działania i efektywności kompresji wykazała, że metoda predykcyjno-substytucyjna jest optymalna w sensie Pareto.

Najważniejszym punktem weryfikacji było zbadanie praktycznej przydatności poszczególnych algorytmów, za której miarę przyjęto współczynnik przyspieszenia transmisji dla założonej prędkości transmisji, przy której nie traci na znaczeniu ani efektywność kompresji, ani czas przetwarzania danych.

Dla przekroju popularnych typów danych reprezentowanego przez oba korpusy testowe, nowy algorytm okazał się najlepszym dostępnym rozwiązaniem. Dla nowszego i bardziej reprezentatywnego korpusu *Canterbury* uzyskano wyraźną poprawę w porównaniu do najlepszego testowanego algorytmu hybrydowego i predykcyjnego,

a jeszcze większą w porównaniu do najlepszego testowanego algorytmu substytucyjnego i transformacyjnego.

Przechodząc do rezultatów uzyskanych dla plików o znanym typie danych, w przypadku przykładowego pliku tekstowego uzyskano wyniki zbliżone do najlepszych z testowanych metod predykcyjnych, a wyraźnie lepsze od najlepszego testowanego algorytmu substytucyjnego. Dla przykładowego pliku binarnego uzyskano wyniki zbliżone do najlepszych z testowanych metod substytucyjnych, a wyraźnie lepsze od najlepszego testowanego algorytmu predykcyjnego. Największą poprawę zaobserwowano dla przykładowego pliku o dużej nadmiarowości, gdzie w porównaniu do najlepszego algorytmu niehybrydowego była ona prawie dwukrotna. Można zatem powiedzieć, że nowa metoda jest bardziej uniwersalna od istniejących metod czysto predykcyjnych lub substytucyjnych, które choć osiągają dobre rezultaty dla jednego typu danych, dla innego okazują się być wyraźnie gorsze.

Wydaje się, że nowa metoda powinna znaleźć szerokie zastosowanie zarówno w przechowywaniu, jak i transmisji danych. Za podstawowy punkt jej zastosowania uznać należy uniwersalne narzędzia kompresji; najprostsze takie narzędzie można stworzyć rozwijając eksperymentalny program PLZ. Wyjątkowo dobra wydajność nowej metody dla danych w wysokim stopniu powtarzalnych czyni ją interesującą kandydatką do użycia w formatach wymiany danych programów użytkowych. Możliwe jest także wbudowanie jej na zasadzie rozszerzenia do klientów poczty elektronicznej czy przeglądarek internetowych.

Szerokie zastosowania nowej metody staną się możliwe dzięki powstającej predykcyjno-substytucyjnej bibliotece kompresji, wzorem takich bibliotek jak *zlib* umożliwiającej prostą implementację kompresji w programach dowolnego przeznaczenia.

Istotną wartość poznawczą mają szczegółowe rezultaty badawcze uzyskane w trakcie opracowywania oraz weryfikowania nowej metody. Zaliczyć do nich należy opracowanie między innymi:

- oryginalnego sposobu efektywnego kodowania stanu przełącznika i parametrów substytucji;
- szybkiego sposobu wykonywania estymacji długości zapisu;
- szeregu usprawnień komponentu predykcyjnego mających za zadanie poprawę szybkości jego działania;
- sposobu działania słowników, który pozwala maksymalizować ich wydajność, ze szczególnym uwzględnieniem specjalnych słowników rzędu 0;

a także następujące wyniki badań empirycznych:

- zbadanie wpływu na szybkość i efektywność kompresji:
 - rzędu modelu komponentu predykcyjnego,
 - minimalnej długości substytucji,
 - rozmiarów i rzędów słowników komponentu substytucyjnego;
- porównanie algorytmów dla zbiorów danych różnego typu i długości;
- porównanie algorytmów dla różnych założonych prędkości transmisji, odpowiadających sytuacjom, w których czynnikowi czasu przypisuje się różną wartość.

Choć uzyskane rezultaty są bardzo dobre, z pewnością możliwa jest ich dalsza poprawa. Interesujące wydaje się być także opracowanie modyfikacji algorytmów, które byłyby najlepsze także dla znacznie niższych, niż badana, prędkości transmisji. W szczególności ciekawa jest możliwość opracowania algorytmów, o bardzo wysokiej efektywności kompresji, które opierałyby swe działanie na zasadzie wyłączania z pasmem pośrednim.

Literatura

- [ABEL03] Abel, J. 2003. *Record Preprocessing for Data Compression*. Preprint. www.data-compression.info/JuergenAbel/Preprints/Preprint_Record_Preprocessing.pdf.
- [ABRA69] Abramson, N. 1969. *Teoria informacji i kodowania*. Państwowe Wydawnictwo Naukowe, Warszawa.
- [ALSB75] Alsberg, P. A. 1975. *Space and Time Savings Through Large Data Base Compression and Dynamic Restructuring*. Proceedings of the IEEE 63, 8 (sierpień), s. 1114.
- [APFR85] Apostolico, A., Fraenkel, A. S. 1985. *Robust Transmission of Unbounded Strings Using Fibonacci Representations*. Technical Report CS85-14, The Weizmann Institute of Science, Rehovot.
- [ARBE97] Arnold, R., Bell, T. C. 1997. *A corpus for the evaluation of lossless compression algorithms*. Proc. IEEE Data Compression Conference, Snowbird, UT, ss. 201–210.
- [BAKU00] Balkenhol, B., Kurtz, S. 2000. *Universal data compression based on the Burrows-Wheeler transformation: Theory and practice*. IEEE Transactions on Computers, 49, 10 (październik), ss. 1043–1053.
- [BLOO96] Bloom, C. 1996. *LZP: a new data compression algorithm*. Proc. IEEE Data Compression Conference, s. 425. <http://www.cbloom.com/lzp.zip>.
- [BLOO98] Bloom, C. 1998. *Solving the Problems of Context Modeling*. <http://www.cbloom.com/ppmz.zip>.
- [BOTT98] Bottou, L., Haffner, P., Howard, P., Simard, P., Bengio, Y., LeCun, Y. 1998. *High Quality Document Image Compression with DjVu*. Journal of Electronic Imaging. <http://www.research.att.com/~haffner/bib2web/publis.html>.
- [BRIS96] Brislawn, C. 1996. *The FBI Fingerprint Image Compression Standard*. <http://www.c3.lanl.gov/~brislawn/FBI/FBI.html>.
- [BUNT94] Bunton, S. 1994. *A characterization of the 'Dynamic Markov Compression' FSM with finite conditioning contexts*. Technical Report UW-CSE-94-11-03, University of Washington.
- [BUNT96] Bunton, S. 1996. *On-Line Stochastic Processes in Data Compression*. Rozprawa doktorska. University of Washington.
- [BUWH94] Burrows, M., Wheeler, D. J. 1994. *A block-sorting data compression algorithm*. SRC Research Report 124. Digital Equipment Corporation, Palo Alto, CA.
- [CAPP85] Cappellini, V. (Red.) 1985. *Data Compression and Error Control Techniques with Applications*. Academic Press, London.
- [CHEN01] Cheney, J. 2001. *Compressing XML with Multiplexed Hierarchical PPM Models*. Proc. IEEE Data Compression Conference.
- [CLEA95] Cleary, J. G., Teahan, W. J., Witten, I. H. 1995. *Unbounded length contexts for PPM*. Proc. IEEE Data Compression Conference, ss. 52–61.
- [CLWI84] Cleary, J. G., Witten, I. H. 1984. *Data compression using adaptive coding and partial string matching*. IEEE Transactions on Communications, 32 (4), ss. 396–402.
- [COHO87] Cormack, G. V., Horspool, R. N. S. 1987. *Data compression using dynamic Markov modelling*. The Computer Journal, 30 (6), ss. 541–550.

- [CORM85] Cormack, G. V. 1985. *Data Compression In a Database System*. Comm. of the ACM 28, 12 (grudzień), s. 1336.
- [CROC00] Crochemore, M., Mignosi, F., Restivo, A., Salemi, S. 2000. *Data compression using antidictionaries*. Proceedings of the IEEE, 88 (11), Special issue 'Lossless data compression', ss. 1756–1768.
- [DADY99] Dąbrowski, A., Dymarski, P. (Red.) 1999. *Podstawy transmisji cyfrowej*. Oficyna Wydawnicza Politechniki Warszawskiej, Warszawa.
- [DANI02] Daniłowicz, Cz. (Red.) 2002. *Multimedialne i sieciowe systemy informacyjne*. Oficyna Wydawnicza Politechniki Wrocławskiej, Wrocław.
- [DEBO99] de Bonet, J. S. 1999. *Data Compression Techniques for Branch Prediction*. <http://www.debonet.com/research/publications/1999/DeBonet-BranchPrediction.pdf>.
- [DEGA96] Deutsch, P., Gailly, J-L. 1996. *ZLIB Compressed Data Format Specification version 3.3*. RFC1950. <http://www.ietf.org/rfc/rfc1950.txt>.
- [DEOR03] Deorowicz, S. 2003. *Universal lossless data compression algorithms*. Rozprawa doktorska. Politechnika Śląska, Gliwice
- [DEUT96a] Deutsch, P. 1996. *DEFLATE Compressed Data Format Specification version 1.3*. RFC1951. <http://www.ietf.org/rfc/rfc1951.txt>.
- [DEUT96b] Deutsch, P. 1996. *GZIP file format specification version 4.3*. RFC1952. <http://www.ietf.org/rfc/rfc1952.txt>.
- [DOUG93] Douglass, F. 1993. *The Compression Cache: Using On-line Compression to Extend Physical Memory*. Proceedings of USENIX Conference, ss. 519–529.
- [DUMO01] Dumont, R., Pellacini, F., Ferwerda, J. A. 2001. *Perceptually-based texture caching for hardware rendering*. Proc. of 12th Eurographics Workshop on Rendering, ss. 246–256.
- [ELIA75] Elias, P. 1975. *Universal Codeword Sets and Representations of the Integers*. IEEE Transactions on Information Theory, 21(2), ss. 194–203.
- [FALL73] Faller, N. 1973. *An Adaptive System for Data Compression*. Record of the 7th Asilomar Conference on Circuits, Systems, and Computers, Piscataway, NJ, IEEE Press, ss. 593–597.
- [FANO49] Fano, R. M. 1949. *Transmission of Information*. M. I. T. Press, Cambridge, Mass.
- [FIEL97] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Berners-Lee, T. 1997. *Hypertext Transfer Protocol – HTTP/1.1*. RFC2068. <http://www.ietf.org/rfc/rfc2068.txt>.
- [FIGR89] Fiala, E. R., Greene, D. H. 1989. *Data compression with finite windows*. Communications of the ACM, 32, 4 (kwiecień), ss. 490–505.
- [FRSI96] Friend, R., Simpson, W. 1996. *PPP Stac LZS Compression Protocol*. RFC1974. <http://www.ietf.org/rfc/rfc1974.txt>.
- [GAAD95] Gailly, J-L., Adler, M. 1995. *Zlib Compression Library*. <http://www.gzip.org/zlib/>.
- [GALL78] Gallagher, R. G. 1978. *Variations on a Theme by Huffman*. IEEE Transactions on Information Theory, 24, 6 (listopad), ss. 668–674.
- [GILC02] Gilchrist, J. 2002. *Archive Comparison Test 2.0*, maj. <http://compression.ca/>.
- [GOLO66] Golomb, S. W. 1966. *Run-Length Encodings*. IEEE Transactions on Information Theory, 12 (lipiec), ss. 399–401.
- [GOYA99] Goyal, K. B., Ramamritham, K., Datta, A., Thomas, H. M. 1999. *Indexing and Compression in Data Warehouses*. Proc. of Int. Workshop on Design and Management of Data Warehouses.

- [GRAB99] Grabowski, Sz. 1999. *Text Preprocessing For Burrows-Wheeler Block Sorting Compression*. VII Konferencja „Sieci i Systemy Informatyczne – teoria, projekty, wdrożenia”, Łódź, ss. 229–239.
- [GREB03] Grebnow, I. 2003. *GRZip – Block Sorting Compressor 0.7.3*. Program kompresujący. <http://magics.h10.ru/main.php?page=GRZip&folder=projects>.
- [GRSH91] Graefe, G., Shapiro, L. D. 1991. *Data Compression and Database Performance*. Proc. ACM/IEEE-CS Symp. on Applied Computing, Kansas City, MO.
- [GRTA94] Grumbach, S., Tahi, F. 1994. *A new challenge for compression algorithms: genetic sequences*. Journal Inf. Process. Manage, 30, ss. 875–886.
- [GUBE94] Gutmann, P. C., Bell, T. C. 1994. *A hybrid approach to data compression*. Proc. IEEE Data Compression Conf., ss. 225–233.
- [HERK02] Herklotz, U. 2002. *UHARC – High Compression Multimedia Archiver 0.5np*. Program kompresujący. Wersja 0.4 do pobrania z: <ftp://ftp.elf.stuba.sk/pub/pc/pack>.
- [HOAN95] Hoang, D. T., Long, P. M., Vitter, J. S. 1995. *Multiple-Dictionary Coding Using Partial Matching*. Proc. IEEE Data Compression Conference, ss. 272–281.
- [HOWA93] Howard, P. G. 1993. *The Design and Analysis of Efficient Lossless Data Compression Systems*. Rozprawa doktorska. Brown University.
- [HUFF52] Huffman, D. A. 1952. *A Method for the Construction of Minimum-Redundancy Codes*. Proc. IRE 40, 9 (wrzesień), ss. 1098–1101.
- [IDG99] Praca zbiorowa 1999. *Vademecum teleinformatyka*, Część 1. Wydawnictwo Idg, Warszawa.
- [INFO99] Info-Zip 1999. Zip 2.3. Program kompresujący. <http://www.info-zip.org/pub/infozip/>.
- [JPEG93] JPEG 1993. *Digital Compression and Coding of Continuous-Tone Still Images*. ISO 10918.
- [JUKI03] Jukin, W. 2003. *Compressors Comparison Test*, wydanie 6.5 (listopad). <http://compression.graphicon.ru/ybs/best.htm>.
- [KIYA03] Kieffer, J., Yang, E. 2002. *Structured Grammar-Based Codes For Universal Lossless Data Compression*. Communications in Information and Systems 2, 1 (czerwiec), ss. 29–52.
- [KRTR81] Krichevsky, R. E., Trofimov, V. K. 1981. *The Performance of Universal Encoding*. IEEE Trans. Inform. Theory, IT-27 (marzec), ss. 199–207.
- [KRUP98] Krupiczka, A. 1998. *Kompresja dźwięku – rozwój metod*. W: [SKAR98], ss. 39–68.
- [LANG84] Langdon, G.G. 1984. *On Parsing Versus Mixed-Order Model Structures for Data Compression*, IBM Research Report RJ-4163 (46091) 1/18/84, IBM Research Laboratory, San Jose, CA.
- [LEHI87] Lelewer, D. A., Hirschberg, D. S. 1987. *Data compression*. ACM Computing Surveys, 19, 261.
- [LYBR81] Lynch, C. A., Brownrigg, E. B. 1981. *Application of Data Compression to a Large Bibliographic Data Base*. Proc. Int. Conf. on Very Large Data Bases, s. 435.
- [MAHO00] Mahoney, M. V. 2000. *Fast Text Compression with Neural Networks*. Proceedings of the 13th International FLAIRS Conference, ss. 230–234.
- [MAHO02] Mahoney, M. V. 2002. *The PAQ1 Data Compression Program*. Draft. <http://www.cs.fit.edu/~mmahoney/compression/paq1.pdf>.
- [MAHO99] Mahoney, M. V. 1999. *Text Compression as a Test for Artificial Intelligence*. Proceedings of the 16th National Conference on Artificial Intelligence and 11th Conference on Innovative Applications of Artificial Intelligence. AAAI Press, s. 970.

- [MART79] Martin, G. N. N. 1979. *Range encoding: an algorithm for removing redundancy from a digitized message*. The Video & Data Recording conference, Southampton.
- [MOFF90] Moffat, A. 1990. *Implementing the PPM data compression scheme*. IEEE Trans. Comm. 38, 11, ss. 1917–1921.
- [MORR68] Morrison, D. R. 1968. *PATRICIA – A practical algorithm to retrieve information coded in alphanumeric*. Journal of the ACM, 15, ss. 514–534.
- [NAKA94] Nakano, Y., Yahagi, H., Okada, Y., Yoshida, S. 1994. *Highly efficient universal coding with classifying to subdictionaries for text compression*. Proc. IEEE Data Compression Conference, ss. 234–243.
- [OBER03] Oberhumer, M. 2003. *LZOP 1.01*. Program kompresujący. <http://www.lzop.org/>.
- [PALL97] Pall, G. 1997. *Microsoft Point-To-Point Compression (MPPC) Protocol*. RFC2118. <http://www.ietf.org/rfc/rfc2118.txt>.
- [PALP98] Palpanas, T. 1998. *Web Prefetching Using Partial Match Prediction*. Tech. Report CSRG-376. University of Toronto.
- [PASC76] Pasco, R. 1976. *Source Coding Algorithms for Fast Data Compression*. Rozprawa doktorska. Stanford University.
- [PAWL03] Pawłow, I. 2003. *7-zip 2.30*. Program kompresujący. <http://www.7-zip.org/>.
- [PIND98] Pindraszak, D. 1998. *Archiwizatory. Kompresja danych*. Mikom, Warszawa.
- [PKWA02] PKWare 2002. *PKZip 6.00*. Program kompresujący. <http://www.pkware.com/>.
- [POST81] Postel, J. 1981. *Internet Protocol*. RFC791. <http://www.ietf.org/rfc/rfc0791.txt>.
- [PRZE98] Przelaskowski, A. 1998. *Miary jakości*. W: [SKAR98], ss. 111–142.
- [RADZ97] Radzikowski, W. 1997. *Badania operacyjne w zarządzaniu przedsiębiorstwem*. Toruńska Szkoła Zarządzania, Toruń.
- [RAND96] Rand, D. 1996. *The PPP Compression Control Protocol (CCP)*. RFC1962. <http://www.ietf.org/rfc/rfc1962.txt>.
- [RAY95] Ray, G., Haritsa, J. R., Seshadri, S. 1995. *Database Compression: A Performance Enhancement Tool*. Advances in Data Management '95, Proceedings of the 7th International Conference on Management of Data (COMAD). Computer Society of India, Tata McGraw-Hill, Pune, Indie.
- [RICE79] Rice, R. F. 1979. *Some Practical Universal Noiseless Coding Techniques*. Technical Report, JPL Publication 79-22, Jet Propulsion Laboratory, Pasadena, CA.
- [RISS76] Rissanen, J. J. 1976. *Generalized Kraft Inequality and Arithmetic Coding*. IBM Journal of Research and Development, 20 (maj), ss. 198–203.
- [ROSZ03] Roszał, E. 2003. *Rar 3.20*. Program kompresujący. <http://www.rarlab.com/>.
- [RUBI76] Rubin, F. 1976. *Experiments in Text File Compression*. Communications of the ACM, 19, 11, ss. 617–623.
- [RUK72] Ruth, S. S., Keutzer, P. J. 1972. *Data compression for business files*. Datamation 18 (wrzesień), s. 62.
- [RYAB84] Ryabko, B. Y. 1984. *Twice Universal Coding*. Problems of Information Transmission, 20(3), ss. 24–28.
- [S3I98] S3 Inc. 1998. *S3TC DirectX 6.0 Standard Texture Compression*. S3 Inc.
- [SAYO02] Sayood, K. 2002. *Kompresja danych – wprowadzenie*. Wydawnictwo RM, Warszawa.
- [SCHI97] Schindler, M. 1997. *A fast block-sorting algorithm for lossless data compression*. Proc. IEEE Data Compression Conference, s. 469.
- [SCHM01] Schmidt, A. R., Waas, F., Kersten, M. L., Florescu, D., Manolescu, I., Carey, M. J., Busse, R. 2001. *The XML Benchmark Project*. <http://monetdb.cwi.nl/xml/Benchmark/benchmark.html>.

- [SCHR96] Schryver, V. 1996. *PPP BSD Compression Protocol*. RFC1977. <http://www.ietf.org/rfc/rfc1977.txt>.
- [SEVE83] Severance, D. G. 1983. *A Practitioner's Guide to Data Base Compression*. Information Systems 8, 1, ss. 51–62.
- [SEWA00] Seward, J. 2000. *Bzip2 1.0.1*. Program kompresujący. <ftp://ftp.elf.stuba.sk/pub/pc/pack>.
- [SHAC98] Shacham, A., Monsour, R., Pereira, R., Thomas, M. 1998. *IP Payload Compression Protocol (IPComp)*. RFC2393. <http://www.ietf.org/rfc/rfc2393.txt>.
- [SHAN48] Shannon, C. E. 1948. *A Mathematical Theory of Communication*. Bell System Technical Journal, 27, ss. 379–423, 623–656.
- [SIMP94] Simpson, W. (Red.) 1994. *The Point-to-Point Protocol (PPP)*. RFC1661. <http://www.ietf.org/rfc/rfc1661.txt>.
- [SKAR98] Skarbek, W. (Red.). 1998. *Multimedia. Algorytmy i standardy kompresji*. Akademicka Oficyna Wydawnicza PLJ, Warszawa.
- [STSZ82] Storer, J. A., Szymanski T. G. 1982. *Data Compression Via Textual Substitution*. Journal of the ACM 29 (4), ss. 928–951.
- [SUBB99] Subbotin, D. 1999. *Carryless rangecoder*. <http://hem.spray.se/mikael.lundqvist/range.tgz>.
- [SWAC01] Swacha, J. 2001. *Formaty graficzne*. Software 2.0, 79 (lipiec), ss. 60–66.
- [SWAC02a] Swacha, J. 2002. *Reprezentacja binarna dokumentu XML z wykorzystaniem XML Schema*. Studia Informatica Nr 16, ZN 337. Wydawnictwo Naukowe Uniwersytetu Szczecińskiego, Szczecin, ss. 43–54.
- [SWAC02b] Swacha, J., Bandosz, M., Radliński, Ł. 2002. *Zaawansowane multimedia na stronach WWW*. W: [DANI02], ss. 411–418.
- [SWAC03] Swacha, J. 2003. *Kompresja dokumentów XML metodą substytucyjno-predykcyjną*. Studia Informatica 17, ZN 359. Wydawnictwo Naukowe Uniwersytetu Szczecińskiego, Szczecin, ss. 155–168.
- [SWAC04] Swacha, J. 2004. *The Predictive-Substitutional Compression Scheme and its Effective Implementation*. Proc. IEEE Data Compression Conference.
- [SWAC99] Swacha, J. 1999. *Popularne standardy kompresji danych*. Pro Dialog, 9, ss. 23–32.
- [SWSZ02] Szyjewski, Z., Swacha, J. 2002. *A New Method of Predictive-substitutional Data Compression*. Internet Technologies, Applications and Societal Impact. Kluwer Academic Publishers, Boston, ss. 123–134.
- [SZKA02a] Szkarin, D. 2002. *PPM: One Step to Practicality*. Proc. Data Compression Conf., ss. 202–211.
- [SZKA02b] Szkarin, D. 2002. *PPMd var. I*. Program kompresujący. <ftp://ftp.elf.stuba.sk/pub/pc/pack>.
- [SZKA03] Szkarin, D. 2003. *Durilca 0.2*. Program kompresujący. <ftp://ftp.elf.stuba.sk/pub/pc/pack>.
- [SZPA99] Szpankowski, W. 1999. *Average Case Analysis of Algorithms*. W: Algorithms and Theory of Computation Handbook, CRC Press, Boca Raton.
- [TEAH98] Teahan, W. J., Inglis, S., Cleary, J. G., Holmes, G. 1998. *Correcting English text using PPM models*. Proc. IEEE Data Compression Conference, ss. 289–298.
- [TECL97] Teahan, W. J., Cleary, J. G. 1997. *Applying compression to natural language processing*. Zgłoszone na ANLP'97. <http://citeseer.nj.nec.com/teahan97applying.html>.
- [TERA93] Teuhola, J., Raita, T. 1993. *Application of a finite-state model to text compression*. The Computer Journal, 36(7), ss. 607–614.

- [TJAL97] Tjalkens, T. J., Volf, P. A. J., Willems, F. M. J. 1997. *A Context-Tree Weighting Method for Text-Generating Sources*. Proc. IEEE Data Compression Conference, s. 472.
- [TOHA01] Tolani, P. M., Haritsa, J. R. 2001. *XGRIND: A Query-friendly XML Compressor*. Proceedings of 18th IEEE Int. Conf. on Data Engineering (ICDE), San Jose, USA.
- [VEVE98] Velasco, J. R., Velasco Lucianez, L. A. 1998. *Benefits of compression in HTTP applied to caching architectures*. Proceedings of the Third International WWW Caching Workshop.
- [VOWI98a] Volf, P. A. J., Willems, F. M. J. 1998. *Switching between two universal source coding algorithms*. Proc. IEEE Data Compression Conf., ss. 491–500.
- [VOWI98b] Volf, P. A. J., Willems, F. M. J. 1998. *The switching method: elaborations*. Proceedings of the 19th Information Theory in Benelux, ss. 13–20.
- [W3C99] W3C 1999. *WAP Binary XML Content Format*. W3C NOTE 24, czerwiec. <http://www.w3.org/TR/wbxml>.
- [WELC84] Welch, T. A. 1984. *A technique for high performance data compression*. IEEE Computer, 17, 6 (czerwiec), ss. 8–19.
- [WILL91] Williams, R. 1991. *LZRW4: Ziv and Lempel Meet Markov*. <http://www.cs.pdx.edu/~idr/unbzip2.cgi?compression/lzrw4.html.bz2>.
- [WILL95] Willems, F. M. J., Shtarkov, Y. M., Tjalkens, T. J. 1995. *The Context-Tree Weighting Method: Basic properties*. IEEE Transactions on Information Theory, 41, 3, ss. 653–664.
- [WILS94] Wilson, W. J. 1994. *Chinks in the armor of public key cryptosystems*. Technical Report 94/3, University of Waikato, Hamilton, New Zealand.
- [WITT87] Witten, I., Bell, T., Cleary, J. 1987: Calgary Corpus. University of Calgary, Canada. <http://www.data-compression.info/Corpora/CalgaryCorpus/calgarycorpus.htm>.
- [WITT99] Witten, I. H., Bray, Z., Mahoui, M., Teahan, W. J. 1999. *Text mining: A new frontier for lossless compression*. Proc. IEEE Data Compression Conference, ss. 198–207.
- [ZILE77] Ziv, J., Lempel, A. 1977. *A Universal Algorithm for Sequential Data Compression*. IEEE Trans. Inform. Theory 23, 3 (maj), ss. 337–343.
- [ZILE78] Ziv, J., Lempel, A. 1978. *Compression of Individual Sequences via Variable-Rate Coding*. IEEE Trans. Inform. Theory 24, 5 (wrzesień), ss. 530–536.