ROZDZIAŁ 4. KODOWANIE ARYTMETYCZNE

Jest to metoda, która pozwala uzyskać największą skuteczność kodowania przy założonym statystycznym modelu źródła informacji. Gdyby nie pewne ograniczenia patentowe, znalazłaby zapewne zastosowanie niemal we wszystkich współczesnych algorytmach kompresji danych. Stąd też wynika godne podkreślenia szczególne znaczenie kodowania arytmetycznego. W rozdziale tym przedstawione zostały podstawowe koncepcje kodera, jak również ich praktyczna implementacja i wybrane sposoby poprawy skuteczności kompresji danych.

4.1. Pomysł metody

Wyobraźmy sobie, że chcemy zakodować zbiór danych, który opisany jest przy pomocy dwuelementowego alfabetu: {0,1}, czyli kolejnymi symbolami pojawiającymi się w strumieniu wejściowym są sekwencje jednobitowych symboli (zero lub jedynka) - daje to średnią bitową 1bit/symbol. Stosując metodę Huffmana w takim przypadku nie uzyskamy żadnej kompresji, gdyż nie uda nam się przyporządkować tym symbolom sekwencji kodowych krótszych od jednego bitu. Kompresję można tutaj ewentualnie uzyskać poprzez rozszerzenie źródła do np. 256 elementowego alfabetu (liczby ośmiobitowe) i huffmanowskie kodowanie tak scharakteryzowanego zbioru danych. Trzeba jednak pamiętać, że jest to zabieg sztuczny, często zupełnie nie związany z charakterem danych, a więc nieefektywny. Aczkolwiek ze statystycznego punktu widzenia to rozszerzenie alfabetu może być skuteczną realizacją fazy modelowania, jeśli występuje pewna korelacja pomiędzy tak tworzonymi wartościami (pewne uzyskane w ten sposób 8-mio bitowe wartości wystąpią częściej w rozważanym zbiorze danych od innych). Jest to jeden ze sposobów zwiększania efektywności kodowania.

W metodzie kodowania arytmetycznego wykorzystano zupełnie inne podejście do problemu optymalnego kodowania źródeł informacji, który polega na przyporządkowaniu kompresowanemu zbiorowi danych jednej liczby ułamkowej (tzw. liczby kodowej) z przedziału [0,1), jednoznacznie dekodowalnej. Poszukuje się tej liczby poprzez sukcesywne zacieśnianie przedziału zwanego przedziałem kodu o początkowej postaci [0,1) w miarę postępu procesu kodowania, tak aby coraz dokładniej dookreślić liczbę kodową. Każda kolejna postać iteracyjnie modyfikowanego przedziału kodu zawiera się w przedziale kroku poprzedniego (jest jego podprzedziałem) wyznaczając jednocześnie nieprzekraczalne granice dla podprzedziałów wyznaczanych w następnych iteracjach. Końcowa postać przedziału kodu jest na tyle charakterystyczna dla kodowanego strumienia danych, że pozwala jednoznacznie odtworzyć oryginalną sekwencję danych. W metodzie tej zrywa się więc całkowicie z szukaniem optymalnych słów kodowych przypisanych osobno dla każdego symbolu, wprowadzając konstrukcję liczby ułamkowej, jakby jednego długiego słowa kodowego dla wszystkich danych wejściowych strumienia. Jej wartość jest modyfikowana w trakcie procesu kodowania przez kolejno pojawiające się dane w strumieniu wejściowym. Wpływ poszczególnych symboli na tę liczbę jest warunkowany ilością informacji związaną z wystąpieniem danego symbolu, a więc z oszacowanym prawdopodobieństwem wystąpienia symbolu w strumieniu wejściowym (zgodnie z przyjętymi statystycznymi modelami źródeł informacji). Metodę kodowania arytmetycznego można wiec zaliczyć podobnie jak metodę Huffmana i S-F do statystycznych metod kodowania entropijnego.

U podstaw arytmetycznej idei kodowania symboli leży więc prosty pomysł, aby jednoznacznie rozróżnić sekwencje symboli na podstawie pewnej liczby z zakresu [0,1), która

zostanie im przypisana. Liczb z zakresu [0,1) jest nieskończenie wiele, wydaje się więc możliwym przyporządkowanie unikalnej liczby dla dowolnej sekwencji wejściowej, czyli potencjał takiej metody jest odpowiedni. Ze względu na wykorzystywane dotychczas statystyczne modele źródeł nasuwa się przy tym odpowiednie narzędzie - dystrybuanta $F_S(a)$ zmiennej losowej *s* charakteryzującej źródło informacji *S* generujące kodowany ciąg symboli. Dzieli ona przedział wartości $\pi^{(0)} = [0,1)$ na podprzedziały, zależnie od alfabetu źródła $A_S = \{a_1, a_2, ..., a_n\}$, $a_i \in R$ oraz wartości prawdopodobieństw wystąpienia poszczególnych symboli alfabetu $P_S = \{p_1, p_2, ..., p_n\}$, gdzie $P(s = a_i) = P(a_i) = p_i$, $p_i \geq 0$ i $\sum_{i=1}^{n} p_i = 1$, w sposób następujący:

$$[F_S(a_{i-1}), F_S(a_i)) \quad \text{dla } 1 \leq i \leq n, \tag{4.1}$$

gdzie $F_S(a_i) = \sum_{j=1}^{i} p_j$, a $F_S(a_0) = 0$. Powyższy podział przedziału początkowego $\pi^{(0)}$ (przedziału odniesienia) na podprzedziały według prawdopodobieństwa występowania poszczególnych symboli alfabetu nosi nazwę linii prawdopodobieństw. Analogiczny podział aktualnego przedziału kodu $\pi^{(m)}$ dokonywany jest kolejno dla pojawiających się danych strumienia wejściowego.

Pierwszy symbol pojawiający się w kodowanym strumieniu $s_1 = a_k$, $1 \leq k \leq n$ powoduje wybór podprzedziału $\pi^{(1)} = [F_S(a_{k-1}), F_S(a_k))$ jako charakterystycznego dla kodowanej danej przedziału kodu. Drugi symbol z sekwencji powoduje teraz wchodzenie w głąb tego przedziału kodu, trzeci jeszcze głębiej, itd. Prowadzi to przy *m* danych w sekwencji wejściowej do ostatecznej postaci przedziału kodu $\pi^{(m)}$, który jest identyfikatorem rozważanej sekwencji i może być jednoznacznie zdekodowany. Dowolna liczba z tego przedziału jest szukaną arytmetyczną reprezentacją zbioru danych, charakteryzującą jednocześnie w sposób jednoznaczny cały zbiór danych, bez przydziału pojedynczych słów kodowych poszczególnym symbolom alfabetu źródła. Pozwala to uniknąć ograniczeń metod tworzących kod symboli.

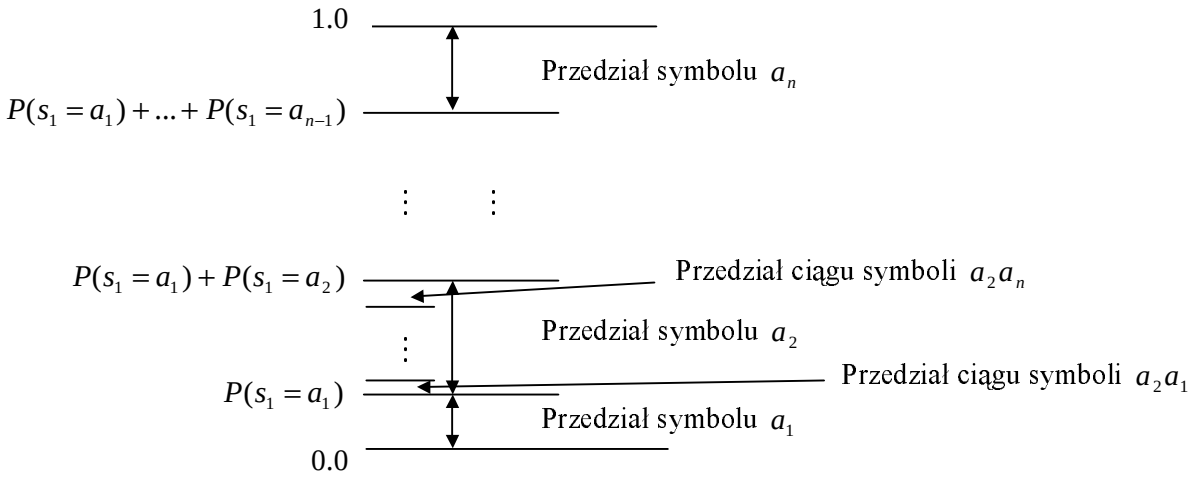
Podział kolejnych przedziałów kodu może się odbywać stale według tej samej linii prawdopodobieństw (zakładamy wtedy model źródła informacji bez pamięci), może także wykorzystywać inne linie prawdopodobieństw (przy modelu źródła informacji z pamięcią), konstruowane w oparciu o modele kumulowanych prawdopodobieństw warunkowych według zależności:

$$D_m(a_k \mid s_1, s_2, ..., s_{m-1}) \equiv \sum_{i=1}^{k-1} P(s_m = a_i \mid s_1, s_2, ..., s_{m-1}), \tag{4.2}$$

$$G_m(a_k \mid s_1, s_2, ..., s_{m-1}) \equiv \sum_{i=1}^{k} P(s_m = a_i \mid s_1, s_2, ..., s_{m-1}). \tag{4.3}$$

które pozwalają określić dolną i górną granicę nowego przedziału kodu $\pi^{(m)}$ wewnątrz aktualnego przedziału $\pi^{(m-1)}$ dla kodowanego właśnie symbolu $s_m = a_k$, poprzedzonego ciągiem $s_1, s_2, ..., s_{m-1}$ danych wejściowych.

Dokonujemy więc kolejnych podziałów przedziału $\pi^{(0)} = [0,1)$ w dziedzinie liczb rzeczywistych w sposób określony przez statystyczny model charakteryzujący źródło informacji. Pokazano to zostało na rys. 4.1.



Rys. 4.1. Prezentacja mechanizmu dzielenia przedziału początkowego $\pi^{(0)} = [0,1)$ w dziedzinie liczb rzeczywistych, w miarę kodowania kolejnych symboli. Mechanizm ten jest zdefiniowany przez model statystyczny źródła informacji.

Zarys samego pomysłu kodowania arytmetycznego można znaleźć już w pracach Shannona [1], a także w kilku pracach w latach sześćdziesiątych [2]. W 1976 roku rozwiązano problem skończonej precyzji kodera arytmetycznego [3][4], by wreszcie skonstruować algorytmy pozwalające na praktyczną aplikację kodera [5][6][7].

4.2. Schemat metody

Podstawowy algorytm metody kodowania arytmetycznego przedstawia się następująco:

Algorytm 4.1A. Metoda kodowania arytmetycznego - kodowanie

1. Szacowanie statystyki danych wejściowych - określenie prawdopodobieństwa wystąpienia każdego z symboli alfabetu.
2. Utworzenie linii prawdopodobieństw poprzez pokrycie przedziału początkowego $\pi^{(0)} = [0,1)$ podprzedziałami przyporządkowanymi każdemu z symboli alfabetu według zależności (4.1).
   Pokrycie to ma być zupełne (cały przedział ma być pokryty) i rozłączne (podprzedziały nie mogą zachodzić na siebie), a jednocześnie szerokość poszczególnych podprzedziałów ma być wprost proporcjonalna do prawdopodobieństwa wystąpienia symbolu. Kolejność pokrywania przedziału $\pi^{(0)}$ przez podprzedziały poszczególnych symboli jest dowolna, tzn. nie ma konieczności sortowania symboli w zależności od wartości prawdopodobieństwa. Najczęściej stosuje się więc metodę zachowania alfabetycznej kolejności symboli, czyli jeśli są to liczby, to zaczynając od dolnej granicy występuje najpierw podprzedział symbolu 0, potem 1, itd., a w przypadku liter - podprzedziały symboli A, B, itd. Ustawiamy licznik iteracji i=1, a także zmienne $dol\_prze^{(0)} = 0$, $gór\_prze^{(0)} = 1$.
3. Czytanie kolejnej danej s_i ze strumienia wejściowego oraz korekcja wartości dolnej i górnej granicy przedziału kodu.

Przyjmując że przeczytana dana odpowiada symbolowi a_k alfabetu źródła czyli $s_i = a_k$, modyfikacja przedziału kodu oznacza, iż nowe wartości granic ustalane są według zależności:

$$dol\_prze^{(i)} = dol\_prze^{(i-1)} + (gór\_prze^{(i-1)} - dol\_prze^{(i-1)}) \cdot dol\_sym(a_k)$$
$$gór\_prze^{(i)} = dol\_prze^{(i-1)} + (gór\_prze^{(i-1)} - dol\_prze^{(i-1)}) \cdot gór\_sym(a_k) \qquad (4.4)$$

przy czym dolna i górna granica symbolu to wartości stałe, ograniczające podprzedział danego symbolu w linii prawdopodobieństw określonej w p.2 tego algorytmu.

4. Powtarzanie p.3 aż do momentu pojawienia się ostatniego symbolu s_m w strumieniu wejściowym. Oczywiście przy przejściu do p .3 następuje inkrementacja: $i=i+1$.
5. Określenie liczby kodowej (*licz_kod*) jako liczby ułamkowej (zmiennoprzecinkowej) wybranej z ostatecznej postaci przedziału kodu określonego przez zmienne $dol\_prze^{(m)}$ i $gór\_prze^{(m)}$ po ostatniej iteracji, czyli $licz\_kod \in \pi^{(m)}$. Zakończenie.

Algorytm 4.1B. Metoda kodowania arytmetycznego - dekodowanie

1. Pobranie ze zbioru danych informacji na temat prawdopodobieństwa występowania w zbiorze poszczególnych symboli alfabetu (muszą być dokładnie takie same jak w koderze).
2. Odtworzenie linii prawdopodobieństw (według tej samej zasady jak w koderze). Wczytanie liczby kodowej z pliku wejściowego jako $licz\_kod^{(1)}$ oraz ustawienie licznika rekonstruowanych symboli $i = 1$.
3. Dekodowanie symbolu poprzez wyznaczenie podprzedziału, w który wpada odczytywana liczba kodowa i określenie związanego z nim symbolu.
   Wykorzystywana jest tutaj linia prawdopodobieństw z p.2 tego algorytmu. Zdekodowanie kolejnego symbolu wyjściowego y_i (w odwracalnej kompresji $y_i = x_i$) jako a_k zachodzi więc wtedy i tylko wtedy, gdy dla źródła informacji S o n-elementowym alfabecie rzeczywistym: $y_i = a_k \Leftrightarrow licz\_kod^{(i)} \in [F_S(a_{k-1}), F_S(a_k))$, $1 \le k \le n$ (liczba kodowa wpada do podprzedziału symbolu a_k w linii prawdopodobieństw). Zakończenie po zdekodowaniu odpowiedniej liczby symboli lub też znaku końca zbioru. W innym przypadku:
4. Eliminacja wpływu zdekodowanego symbolu a_k na liczbę kodową poprzez następującą operację:

$$licz\_kod^{(i+1)} = \frac{licz\_kod^{(i)} - dol\_sym(a_k)}{gór\_sym(a_k) - dol\_sym(a_k)} \qquad (4.5)$$

5. Skok do p. 3 po inkrementacji $i = i + 1$.

Rozważmy przykład 4.1 ilustrujący sposób konstruowania liczby kodowej dla krótkiego strumienia danych.

PRZYKŁAD 4.1. Kompresja metodą kodowania arytmetycznego.

Następujący strumień wejściowy: ARYTMETYKA zostanie poddany procesowi kodowania według algorytmu 4.1A oraz dekodowania, zgodnie z algorytmem 4.1B. Szacowanie statystyki danych oraz tworzenie linii prawdopodobieństw przedstawione zostało w tabeli 4.1.

Tabela 4.1. Szacowanie prawdopodobieństw symboli oraz określenie linii prawdopodobieństw dla przykładu z tekstu.

Symbole alfabetu	Prawdopodobieństwo	Podprzedziały
A	2/10	$0.0 \le p < 0.2$
E	1/10	$0.2 \le p < 0.3$
K	1/10	$0.3 \le p < 0.4$
M	1/10	$0.4 \le p < 0.5$
R	1/10	$0.5 \le p < 0.6$
T	2/10	$0.6 \le p < 0.8$
Y	2/10	$0.8 \le p < 1.0$

Posługując się linią prawdopodobieństw można teraz, analizując kolejne dane ze strumienia wejściowego, przeprowadzać korekcję wartości dolnego i górnego progu według zależności (4.4), co przedstawiono w tabeli 4.2.

Tabela 4.2. Kodowanie danych z przykładu 4.1 metodą kodowania arytmetycznego.

Kolejne dane ze strumienia wejściowego	Dolna granica przedziału kodu	Górna granica przedziału kodu	Szerokość przedziału kodu
Inicjalizacja	0	1	1
A	0.0	0.2	0.2
R	0.1	0.12	0.02
Y	0.116	0.12	0.004
T	0.1184	0.1192	0.0008
M	0.11872	0.1188	0.00008
E	0.118736	0.118744	0.000008
T	0.1187408	0.1187424	0.0000016
Y	0.11874208	0.1187424	0.00000032
K	0.118742176	0.118742208	0.000000032
A	0.118742176	0.1187421824	0.0000000064

Według p.5 algorytmu kodowania 4.1A jako liczba kodowa może być wybrana dowolna liczba z przedziału [0.118742176, 0.1187421824). Przyjmijmy więc liczbę równą dolnej granicy ostatecznej wersji zawężanego przedziału - 0.118742176 jako nową reprezentację kodowanego zbioru danych. Proces dekodowania tej liczby i wiernego odtwarzania oryginalnej postaci zbioru przy pomocy algorytmu 4.1.B został przedstawiony w tabeli 4.3.

Jak widać z tabeli 4.3 po zdekodowaniu przedostatniej litery – ‘K’ wartość modyfikowanej liczby kodowej przyjmuje wartość 0. Oczywiście w takim przypadku kolejnym dekodowanym symbolem musi być ‘A’, przy czym powstaje pytanie jak długo, bowiem kolejne dekodowanie ‘A’ niczego nie zmienia (wartości graniczne oraz liczba kodowa są takie same). Do dekodera musi być przesłana dodatkowo informacja o ilości symboli w zbiorze (w naszym przypadku 10), tak że dekoder po określeniu dziesiątego symbolu kończy proces odtwarzania oryginalnego zbioru danych. Innym rozwiązaniem jest dołączenie, podobnie jak w metodzie Huffmana, znaku końca zbioru EOF do alfabetu opisującego dany zbiór i przydzielenie mu odpowiedniego podprzedziału na linii prawdopodobieństw. Zdekodowanie symbolu EOF kończy proces dekompresji.

Tabela 4.3. Dekodowanie liczby kodowej z przykładu 4.1 metodą kodowania arytmetycznego. Wejściowa liczba kodowa wynosi 0.118742176 i jest dolną granicą końcowej postaci przedziału kodu w koderze.

Liczba kodowa (modyfikowana)	Dekodowane symbole	Dolna granica podprzedziału symbolu	Górna granica podprzedziału symbolu	Szerokość podprzedziału symbolu
0.118742176	A	0	0.2	0.2
0.59371088	R	0.5	0.6	0.1
0.9371088	Y	0.8	1	0.2
0.685544	T	0.6	0.8	0.2
0.42772	M	0.4	0.5	0.1
0.2772	E	0.2	0.3	0.1
0.772	T	0.6	0.8	0.2
0.86	Y	0.8	1	0.2
0.3	K	0.3	0.4	0.1
0	A	0	0.2	0.2
0	...	...	...	...

Wracając do przykładu 4.1 i do zasady wyboru ostatecznej postaci liczby kodowej, wyrażonej w p.5 algorytmu 4.1A można zauważyć, że wybierając liczbę 0.11874218 jako nową reprezentację zbioru danych uzyskamy większą kompresję. Musimy bowiem zapamiętać lub przesłać do dekodera liczbę o ośmiu cyfrach po przecinku zamiast liczby o dziewięciu cyfrach po przecinku. Proces dekompresji w tym przypadku jest pokazany w tabeli 4.4. Tutaj także niezbędną jest dodatkowa informacja o zakończeniu procesu dekodowania. Gdyby nie było takiej informacji, po zdekodowaniu słowa ARYTMETYKA kolejnymi dekodowanymi symbolami byłyby 'T', 'A', 'T', 'A' itd.

Tabela 4.4. Dekodowanie liczby kodowej z przykładu 4.1 metodą kodowania arytmetycznego. Liczba kodowa wynosi 0.11874218 i należy do przedziału kodu.

Liczba kodowa (modyfikowana)	Dekodowane symbole	Dolna granica podprzedziału symbolu	Górna granica podprzedziału symbolu	Szerokość podprzedziału symbolu
0.11874218	A	0	0.2	0.2
0.5937109	R	0.5	0.6	0.1
0.937109	Y	0.8	1	0.2
0.685546	T	0.6	0.8	0.2
0.427725	M	0.4	0.5	0.1
0.27725	E	0.2	0.3	0.1
0.7725	T	0.6	0.8	0.2
0.8625	Y	0.8	1	0.2
0.3125	K	0.3	0.4	0.1
0.125	A	0	0.2	0.2
0.625	...	...	...	...

4.3. Praktyczna realizacja (koder arytmetyczny w arytmetyce liczb całkowitych)

Przedstawiony algorytm arytmetycznego kodowania pozwala zaradzić wspomnianych wadom metody Huffmana, jednak w opisanej formie jest zupełnie niepraktyczny. Po pierwsze kosztowana arytmetyka zmiennoprzecinkowa i złożony algorytm daje efekt ogromnej czasochłonności metody. Ponadto co zrobić, gdy długość liczby kodowej tworzonej dla dużego zbioru danych zaczyna przekraczać długość rejestrów naszego komputera? Rodzą się też inne pytania i niejasności. Czy realizacja kodera wymaga procesorów zmiennoprzecinkowych? Czy maszyny o różniącym się formacie danych zmiennoprzecinkowych mogą przeprowadzić proces kodowania w sposób jednakowy? Czy zbiór zakodowany na jednej z nich może być zdekodowany na drugiej?

Otóż praktyczne implementacje idei kodera arytmetycznego są realizowane w arytmetyce liczb całkowitych przy pomocy 16 lub 32 bitowych rejestrów. Pomysł takiej realizacji jest przeniesieniem zasadniczej idei modyfikacji przedziału w realia binarnej reprezentacji danych oraz sukcesywnej analizy jedynie modyfikowanej części powstającej liczby kodowej. Przez rejestry przesuwają się kolejno coraz mniej znaczące bity granicznych wartości liczby kodowej, tak samo istotne wobec zmniejszającego się przedziału z liczbą kodową. Po przejściu przez rejestry nie podlegający już modyfikacji fragment liczby kodowej może być sukcesywnie zapisywany do pliku.

Na początek należy przyjąć, że początkowa postać przedziału kodu [0,1) może być określona przez granicą dolną 0.0 i granicę górną przedziału 0.9999..., co w notacji binarnej można zapisać jako 0.00000... i 0.11111 ... Warto zauważyć, że z lewej strony obie te liczby mają zerową część całkowitą i tak pozostanie do końca procesu kodowania niezależnie od długości i charakteru strumienia wejściowego. Wystarczy więc kontrolować jedynie części ułamkowe obu wartości. Z drugiej strony wiadomo, że dolna granica jest określona przez nieskończony ciąg zer, a wartość górnej granicy to ciągnący się w nieskończoność szereg dziewiątek (binarnie -jedynek).

Jeśli teraz do przechowywania aktualnych wartości granic przedziału przeznaczymy 16 bitowe rejestry *DÓŁ* i *GÓRA*, to szesnastkowo ich wartość początkowa będzie wynosić 0x0000 i 0xFFFF. Liczby te są domyślnie uzupełniane z lewej strony zerową częścią całkowitą i przecinkiem, a z prawej strony ciągiem odpowiednio zer i jedynek. Tak skonstruowana postać początkowa granic przedziału kodu będzie następnie modyfikowana iteracyjnie przy kodowaniu kolejnych symboli wejściowych. Jeśli zajdzie potrzeba przesunięcia wartości dolnej granicy w lewo, wówczas z rejestru wysunie się w lewo najbardziej znacząca pozycja liczby ułamkowej, a rejestr zostanie uzupełniony zerem. Przy przesunięciu w lewo najstarszej cyfry z rejestru górnej granicy przedziału, rejestr jest uzupełniany jedynkami (lub dziewiątkami przy notacji dziesiętnej).

Modyfikacja dolnej i górnej granicy przedziału odbywa się według zależności (4.4). Należy jednak pamiętać, że przy obliczaniu aktualnej szerokości przedziału musimy inkrementować różnicę granic przedziału: *DÓŁ* - *GÓRA* + 1. Jednocześnie wpisujemy do rejestru *GÓRA* obliczoną w kolejnej iteracji wartość górnej granicy przedziału zmniejszoną o jeden.

Pozostaje jedynie określić warunki, w których następuje przesunięcie zawartości rejestrów *DÓŁ* i *GÓRA* i uzupełnienie ich zawartości. Łatwo zauważyć, że w przypadku gdy na najbardziej znaczącej pozycji liczb ułamkowych dolnej i górnej granicy pojawi się ta sama cyfra, to nie ulegnie ona zmianie już do końca procesu kodowania strumienia wejściowego, bez względu na jego długość i charakter. Wynika to z faktu, iż przedział w procesie kodowania jest sukcesywnie zawężany i granice przedziału iteracji i są nieprzekraczalne dla iteracji $i+1$ i następnych. Można więc spokojnie wysunąć tę cyfrę z obu rejestrów i zapisać np. w pliku wyjściowym, pozostawiając jedynie tą część liczb ułamkowych stanowiących

wartości obu granic, które mogą ulec modyfikacji w kolejnych iteracjach algorytmu kodowania.

Ponadto, trzeba jeszcze zabezpieczyć się przed sytuacją, kiedy wysuwanie kolejnych znaczących cyfr z obu rejestrów zostanie zahamowane. Taki przypadek może wystąpić, kiedy na najstarszej pozycji w obu rejestrach występują różniące się o jeden cyfry, a proces zawężania przedziału w kolejnych iteracjach nie doprowadza do ich zrównoważenia. Wówczas modyfikowane części liczb ułamkowych mogą się w pewnym momencie przestać mieścić w rejestrach grożąc utratą informacji i całkowitą klęską procesu bezstratnego kodowania.

Rozwiązaniem jest wczesna detekcja takiej sytuacji i zapobieżenie jej propagacji poprzez prosty mechanizm wykrycia niedomiaru i jego usunięcia. Sytuacja niedomiaru występuje wtedy, gdy najstarsza cyfra w obu rejestrach różni się o jeden, a na następnej pozycji (tzw. pozycji niedomiaru) występuje 0 w rejestrze *GÓRA* i najbardziej znacząca cyfra danej notacji (np. 9 w zapisie dziesiętnym) w rejestrze *DÓŁ*. Usuwanie niedomiaru odbywa się wówczas w sposób następujący: przesuwane są o jeden w lewo wszystkie pozostałe pozycje rejestru oprócz najstarszych cyfr. Niewygodne cyfry 0 i 9 na pozycji niedomiaru odpowiednio w rejestrach *GÓRA* i *DÓŁ* są wtedy zapisane przez kolejne, mniej znaczące cyfry obu rejestrów. Najbardziej znaczące cyfry pozostają bez zmian, a najmłodsza pozycja w rejestrach zostaje odpowiednio uzupełniona. Każda taka operacja przesunięcia musi być oczywiście zarejestrowana i powtórzona podczas dekodowania.

Wersja algorytmu kodowania arytmetycznego w arytmetyce liczb całkowitych przedstawia się więc następująco:

Algorytm 4.2A. Metoda kodowania arytmetycznego w arytmetyce liczb całkowitych - kodowanie. Wykorzystano dziesiętną notację liczb.

1. Jak w algorytmie 4.1A
2. Pokrycie przedziału prawdopodobieństwa [0,1) jak w algorytmie 4.1.A. Ustawiamy wartości rejestrów $DÓŁ = 0...0$, $GÓRA = 9...9_{(10)}$ oraz zerujemy licznik niedomiaru L=0.
3. Czytanie z wejścia kolejnej danej s_i odpowiadającej symbolowi a_k alfabetu źródła.
4. Modyfikacja zawartości rejestrów:

$$\begin{aligned} SZEROKOŚĆ &= GÓRA - DÓŁ + 1 \\ DÓŁ &= DÓŁ + SZEROKOŚĆ \cdot dol\_sym(a_k) \\ GÓRA &= DÓŁ + SZEROKOŚĆ \cdot gór\_sym(a_k) - 1 \end{aligned} \tag{4.6}$$

5. Korekcja wartości rejestrów *DÓŁ* i *GÓRA*. Sprawdzana jest najbardziej znacząca cyfra obu rejestrów:

   a) jeśli najbardziej znacząca cyfra w rejestrach *DÓŁ* i *GÓRA* jest taka sama, przesuwamy o jedną pozycję cyfry obu rejestrów wysyłając na wyjście najbardziej znaczącą cyfrę i uzupełniając rejestr *DÓŁ* cyfrą 0 oraz rejestr *GÓRA* cyfrą $9_{(10)}$. Jeśli licznik niedomiaru jest różny od zera, wówczas jest wysyłana na wyjście informacja o zawartości licznika (najczęściej w realizacjach koderów w konwencji dwójkowej jeśli z najstarszej pozycji wysuwany jest bit *b*, to na wyjście wysyłana jest także liczba bitów (1-*b*) równa zawartości licznika [6]). Ustawiamy L=0.

b) w przypadku, gdy najbardziej znaczące cyfry w *DÓŁ* i *GÓRA* różnią się o jeden i drugie w kolejności znaczenia cyfry obu rejestrów są następujące: 0 w *DÓŁ* i $9_{(10)}$ w *GÓRA*, wówczas następuje przesunięcie wszystkich cyfr obu rejestrów, z wyjątkiem dwu najbardziej znaczących, o jedną pozycję w lewo i odpowiednie uzupełnienie rejestrów, jak w p. 5a). Inkrementowany jest licznik niedomiaru: $L=L+1$.

6. Skok do p.3 aż do momentu pojawienia się ostatniego symbolu w strumieniu wejściowym.
7. Wysłanie na wyjście wszystkich znaczących cyfr (różnych od zera), jakie pozostały w rejestrze *DÓŁ*. Zakończenie.

Aby uzyskać bardziej praktyczny opis algorytmu w notacji binarnej wystarczy zastąpić $9_{(10)}$ przez $1_{(2)}$. Przy pomocy arytmetyki całkowitoliczbowej można więc zrealizować ideę kodowania arytmetycznego. Pokazuje to następujący przykład:

PRZYKŁAD 4.2. Realizacja całkowitoliczbowa kodera arytmetycznego.

Należy skompresować ciąg symboli ARYTMETYKA metodą kodowania arytmetycznego z wykorzystaniem arytmetyki całkowitoliczbowej. Dostępne są rejestry o pojemności pięciu cyfr w notacji dziesiętnej.

Kolejne etapy algorytmu kodowania według algorytmu 4.2A przedstawiono w tablicy 4.5.

Tabela 4.5. Kodowanie danych z przykładu 4.2 metodą kodowania arytmetycznego według implementacji całkowitoliczbowej.

Kolejne symbole wejściowe i działania powodowane ich wystąpieniem	Zawartość rejestru *DÓŁ*	Zawartość rejestru *GÓRA*	Szerokość przedziału określonego przez rejestry *DÓŁ* i *GÓRA*	Strumień wyjściowy
Inicjalizacja	00000	99999	100000	-
A	00000	19999	020000	
R	10000	11999		
Wysuń 1	00000	19999	020000	1
Y	16000	19999		
Wysuń 1	60000	99999	040000	11
T	84000	91999	008000	
M	87200	87999		
Wysuń 8 i 7	20000	99999	080000	1187
E	36000	43999	008000	
T	40800	42399		
Wysuń 4	08000	23999	016000	11874
Y	20800	23999		
Wysuń 2	08000	39999	032000	118742
K	17600	20799	003200	
A	17600	18239		
Wysuń 1	76000	82409		1187421
Wysuń 7 i 6				118742176

O ile przedstawiony mechanizm modyfikacji zawartości rejestrów umożliwia generację takiego samego kodu jak algorytm 4.1A, przy analogicznych operacjach, to jednak odtworzenie oryginalnego strumienia symboli z kodu wyjściowego wymaga nieco innej pracy dekodera, niż to wynika z algorytmu 4.1B. Dekodowanie według tego algorytmu wydaje się operacją prostszą niż proces kodowania, lecz równie niepraktyczną. Potrzeba wówczas tylko jednej zmiennej zawierającej odczytany kod wejściowy i linii prawdopodobieństw umieszczonej w wygodnej do porównań strukturze. Wpadająca w konkretny podprzedział linii prawdopodobieństw liczba kodowa jednoznacznie detekuje symbol, którego wpływ na liczbę kodową zostaje usunięty prostą operacją według równania (4.5). Nie potrzeba odtwarzać wartości dolnej i górnej granicy przedziału kodu i śledzić ich w procesie dekodowania kolejnych symboli.

W praktycznej aplikacji potencjalnie nieskończenie długi ciąg cyfr liczby kodowej może być wczytywany sekwencyjnie do rejestru o skończonej, określonej długości (*n* cyfr danej notacji). Dla ustalenia uwagi rejestr ten będziemy nazywać *KOD*. W trakcie dekompresji liczba kodowa jest przesuwana przez rejestr aż do ostatnich cyfr znaczących. W takim rozwiązaniu nie jest znana cała liczba kodowa na danym etapie algorytmu dekodowania, lecz jedynie jej fragment, znajdujący się aktualnie w rejestrze *KOD*. Nie można więc wyznaczać kolejnych symboli dekodowanych z kodu wejściowego poprzez rzutowanie liczby kodowej na linię prawdopodobieństw. Aby więc określić te symbole, trzeba dokładnie naśladować proces kodowania, w tym przede wszystkim kolejne modyfikacje rejestrów *DÓŁ* i *GÓRA* z przesuwaniem i uzupełnianiem ich zawartości jak w koderze. Ponadto operacje przesuwania i uzupełniania są wykonywane także na rejestrze *KOD*, z tym że jest on uzupełniany na pozycji najmniej znaczącej kolejnymi cyframi znaczącymi liczby kodowej, czytanymi np. z pliku. We wszystkich trzech rejestrach najbardziej znaczące cyfry wysuwane w kolejnych operacjach na rejestrach nie są już użyteczne. Cała informacja w nich zawarta została bowiem odczytana i wykorzystana do odtworzenia kolejnych symboli oryginalnej reprezentacji.

Mając do dyspozycji zawartość tych trzech rejestrów można określić zakodowany symbol poprzez operację rzutowania tzw. lokalnej liczby kodowej (*lok_licz_kod*) na linię prawdopodobieństwa. Lokalna liczba kodowa jest wyznaczana na podstawie aktualnych zawartości trzech rejestrów i odpowiada położeniu zawartości rejestru *KOD* w granicach wyznaczonych przez *DÓŁ* i *GÓRA*.

Algorytm dekodowania w wersji umożliwiającej praktyczną realizację przedstawia się więc następująco:

Algorytm 4.2B. Metoda kodowania arytmetycznego w arytmetyce liczb całkowitych - dekodowanie

1. Jak w algorytmie 4.1B.
2. Pokrycie przedziału prawdopodobieństwa [0,1) jak w algorytmie 4.1.B. Ustawiamy wartości rejestrów $DÓŁ = 0...0$, $GÓRA = 9...9_{(10)}$.
3. Wczytanie do rejestru *KOD n* pierwszych cyfr liczby kodowej ze strumienia wejściowego.
4. Obliczenie pomocniczej liczby kodowej z zależności:

$$lok\_licz\_kod = \frac{KOD - DÓŁ}{GÓRA - DÓŁ + 1}. \tag{4.7}$$

Dekodowanie symbolu poprzez wyznaczenie takiego podprzedziału linii prawdopodobieństw, w który wpada pomocnicza liczba kodowa, a następnie określenie związanego z nim symbolu a_k.

5. Modyfikacja zawartości rejestrów *DÓŁ* i *GÓRA* według zależności (4.6).
6. Korekcja zawartości rejestrów *DÓŁ, GÓRA* i *KOD*. Sprawdzana jest najbardziej znacząca cyfra obu rejestrów *DÓŁ* i *GÓRA*:

   a) jeśli najbardziej znacząca cyfra w rejestrach *DÓŁ* i *GÓRA* jest taka sama, przesuwamy o jedną pozycję cyfry trzech rejestrów usuwając z nich najbardziej znaczącą cyfrę i uzupełniając rejestr *DÓŁ* cyfrą 0, rejestr *GÓRA* cyfrą $9_{(10)}$, a na najmniej znaczącą pozycję rejestru *KOD* wpisujemy kolejną cyfrę ze strumienia wejściowego.

   b) w przypadku, gdy najbardziej znaczące cyfry w *DÓŁ* i *GÓRA* różnią się o jeden i drugie w kolejności znaczenia cyfry obu rejestrów są następujące: $9_{(10)}$ w *DÓŁ* i 0 w *GÓRA*, wówczas następuje przesunięcie wszystkich cyfr trzech rejestrów, z wyjątkiem najbardziej znaczących, o jedną pozycję w lewo i odpowiednie uzupełnienie rejestrów, jak w p. 6a). Ilość przesunięć jest sterowana wartości licznika niedomiaru odczytaną ze strumienia wejściowego.
7. Skok do p.4 aż do momentu pojawienia się ostatniego symbolu w strumieniu wejściowym.
8. Zakończenie procesu dekodowania po zdekodowaniu odpowiedniej liczby symboli lub też znaku końca zbioru.

Proces dekodowania zakodowanej sekwencji ARYTMETYKA z przykładu 4.2 według powyższego algorytmu przedstawia tabela 4.6.

Tabela 4.6. Dekodowanie danych z przykładu 4.2 według implementacji całkowitoliczbowej.

Zawartość rejestru *KOD*	Zawartość rejestru *DÓŁ*	Zawartość rejestru *GÓRA*	Szerokość przedziału określonego przez rejestry *DÓŁ* i *GÓRA*	Lokalna liczba kodowa	Dekodowane symbole i działania pomocnicze
11874	00000	99999	100000	0.11874	**A**
	00000	19999	20000	0.5937	**R**
	10000	11999			Usuń 1
18742	00000	19999	20000	0.9371	**Y**
	16000	19999			Usuń 1
87421	60000	99999	40000	0.685525	**T**
	84000	91999	8000	0.427625	**M**
	87200	87999			Usuń 8 i 7
42176	20000	99999	80000	0.2772	**E**
	36000	43999	8000	0.772	**T**
	40800	42399			Usuń 4
21760	08000	23999	16000	0.86	**Y**
	20800	23999			Usuń 2
17600	08000	39999	32000	0.3	**K**
	17600	20799	3200	0	**A**

4.4. Modelowanie statystyczne

Stosowana w algorytmie kodowania arytmetycznego linia prawdopodobieństw może być wyznaczona w sposób statyczny, na podstawie ilości wystąpień poszczególnych symboli alfabetu w kompresowanym pliku danych. Wymaga to zastosowania algorytmu dwuprzebiegowego oraz umieszczenia dodatkowej informacji o statystyce symboli w strumieniu wyjściowym. Informacja ta jest niezbędna w procesie dekompresji. Okazuje się, że w przypadku wszystkich metod entropijnego kodowania efektywność wzrasta, jeśli prawdopodobieństwo wystąpienia poszczególnych symboli jest bardziej zróżnicowane. Innymi słowy, lepiej określony model statystyczny, dokładniej opisujący lokalne zależności w sygnale kodowanym tworzy silniej zróżnicowaną mapę zapisywanej informacji, która jest wynikiem pojawienia się kolejnych symboli w kodowanej sekwencji danych. Lepszy model statystyczny można to zrealizować przy pomocy algorytmu adaptacyjnego, a ponadto zwiększyć skuteczność kodowania poprzez zastosowanie modeli statystycznych wyższych rzędów.

Algorytm adaptacyjny

Koder arytmetyczny w swojej najprostszej postaci może być zrealizowany tak jak koder Huffmana w wersji statycznej, czyli w pierwszym etapie zbudowany zostaje model statystyczny na podstawie częstości wystąpienia poszczególnych symboli w całym zbiorze danych przeznaczonym do kompresji. Najoszczędniej jest następnie zapisać w kodowanym strumieniu wyjściowym tablicę liczby wystąpień symboli, aby dekoder mógł powtórzyć ten sam proces budowania modelu będącego fundamentem w procedurze rekonstrukcji danych oryginalnych.

Przed kodowaniem oraz przesłaniem liczby zliczeń poszczególnych symboli korzystnie jest je przeskalować tak, aby mieściły się na przykład w jednym bajcie każda. Takie spłaszczenie dynamiki zliczeń nie wnosi zazwyczaj istotnych strat w efektywności algorytmu kodowania pozwalając w oszczędny sposób zapisać informacje dla dekodera.

Ponieważ wadą modelu statycznego oprócz konieczności dopisania dodatkowych danych jest także niedostosowanie do lokalnej charakterystyki zbioru danych, często opłaca się konstruować model adaptacyjnie zaczynając od początkowej postaci statystyki, pozwalającej możliwie szybko uzyskać dużą efektywność modelu. Takie rozwiązanie działając oczywiście z pewną inercją znacznie lepiej opisze lokalne własności danych zwiększając zróżnicowanie prawdopodobieństw wystąpienia poszczególnych symboli. Efektem jest krótszy strumień kodu na wyjściu, czyli wzrost efektywności kompresji.

Przy ustalaniu początkowej statystyki dobrze jest skorzystać z wszelkiej wiedzy dostępnej a priori na temat kompresowanego zbioru danych. Zamiast więc typowej równomiernej statystyki (waga 1 przypisana każdemu współczynnikowi z alfabetu) można zacząć od średniej statystyki kompresowanego zbioru, statystyki wynikającej z zasad alfabetu, itd. Model początkowy jest następnie modyfikowany poprzez inkrementację wag symboli pojawiających się kolejno w strumieniu wejściowym. Oczywiście, ta sama inicjacja modelu oraz jego sukcesywna modyfikacja musi być dokładnie powtórzona w procesie dekompresji. Można także podczas inicjalizacji wyobrazić sobie bardzo ubogą linię prawdopodobieństw, zawierającą prawdopodobieństwa zaledwie kilku podstawowych symboli, analogicznie do adaptacyjnego algorytmu Huffmana, do której wprowadzane są kolejne symbole w miarę ich pojawiania się w strumieniu wejściowym.

Statystyczne modele Markowa

Przy modelowaniu źródła informacji przez model Markowa liczba informacji związana z wystąpieniem poszczególnych symboli alfabetu obliczana jest przy pomocy zestawu prawdopodobieństw warunkowych. Taki też model jest wykorzystywany w praktyce do konstrukcji optymalnego kodu w algorytmie kodowania arytmetycznego, według zależności (4.2) i (4.3). Ze względu na złożoność takiego modelu, który nawet przy rzędzie 1 zawiera 256 linii prawdopodobieństw (przy założeniu jednobajtowych danych), praktycznie konieczne jest użycie adaptacyjnego algorytmu budowania i korekcji modelu.

Najprostszy przypadek modelu rzędu 1 polega na wyznaczeniu wielu linii prawdopodobieństw zamiast jednej do konstrukcji kodu arytmetycznego. Linie te są wybierane do kolejnych modyfikacji przedziału kodowania w zależności od kontekstu, tj. wartości bezpośrednio poprzedzającej dany symbol. Chodzi tutaj o dokładniejszą charakterystykę lokalnej statystyki danych w kodowanym strumieniu. Naturalnym jest występowanie korelacji pomiędzy sąsiednimi wartościami w tymże strumieniu, a więc model uwzględniający te zależności potrafi dokładniej określić rzeczywistą informację zawartą w pliku. Wiedząc przykładowo dla zbiorów danych tekstowych, że ostatnią zakodowaną literą była L należy wybrać teraz do kodowania kolejnego symbolu linię charakteryzującą wystąpienia innych liter po literze L. Wiadomo, że dużo częściej po L występuje A niż Z, więc w linii(L) szerokość podprzedziału przypadająca na symbol A będzie dużo większa niż podzakres wartości prawdopodobieństwa przypadający na symbol Z.

Istotnym problemem w tak złożonym modelu jest wiarygodne określenie tychże linii prawdopodobieństw. Jeśli postać początkową zbioru linii można utworzyć na podstawie wiedzy dostępnej *a priori*, np. na podstawie zasad słownictwa danego języka, to zadanie jest proste, a model od samego początku procesu kodowania może być efektywny. Gorzej jest w przypadku, kiedy nie mamy informacji wstępnych na temat kompresowanego zbioru danych lub też wiemy, że własności źródła informacji generującego ten zbiór są daleko niestacjonarne. W takim przypadku trzeba rozpocząć budowanie modelu od początku, czyli najczęściej równomiernego rozkładu prawdopodobieństw pomiędzy poszczególne symbole i konteksty. Aby jednak taki model zaadoptować to własności statystycznych zbioru (tj. odpowiednio zróżnicować prawdopodobieństwa warunkowe wystąpienia poszczególnych symboli) potrzeba zakodować pewną liczbę symboli. Dopiero wtedy model zacznie spełniać swoją funkcję wiarygodnego opisania informacji zawartej w zbiorze wejściowym. Wiarygodny statystycznie model prawdopodobieństw warunkowych zacznie więc skutecznie kształtować wyjściowy strumień kodu dopiero po zakodowaniu w sposób mniej efektywny nieraz dużej partii zbioru.

Zjawisko występujące w pierwszej fazie kodowania, kiedy to złożony model statystyczny w wersji inicjacyjnej nie został jeszcze wiarygodnie zweryfikowany (określony) na podstawie danych wejściowych, nazywane jest rozrzedzeniem kontekstu (ang. context dilution). Brak jest wtedy dostatecznej liczby danych, aby dobrze estymować prawdopodobieństwa warunkowe opisujące stany modelu, czyli występuje brak określoności pełnego modelu kontekstowego (tj. jego mała wiarygodność statystyczna). Jedynie fragmenty modelu mogą być wiarygodne, a efektem jest mniejsza efektywność rozrzedzonego kontekstu w procesie kodowania.

Przy doborze optymalnego rozmiaru kontekstu, czyli najbardziej efektywnego z punktu widzenia skuteczności kompresji rzędu modelu trzeba uzyskać kompromis pomiędzy

rozmiarem nośnika modelu, odpowiadającego rzeczywistym korelacjom w zbiorze danych a wiarygodnością jego określenia. Rozważmy prosty przykład:

PRZYKŁAD 4.3. Kodowanie arytmetyczne zbioru tekstowego z modelowaniem kontekstu.

Plik tekstowy z 8-mio bitowymi znakami w kodzie ASCII poddany został bezstratnej kompresji przy pomocy kodera arytmetycznego z modelem statystycznym dowolnego rzędu. Długość pliku oryginalnego wynosi 1,1 megabajta (1137436 bajty). Uzyskana skuteczność kompresji dla modeli rzędu od 1 do 10 została przedstawiona w tabeli 4.7. Ponadto obliczono wartość entropii tego samego zbioru danych przyjmując analogiczne modele źródeł z pamięcią, czyli średnią entropię warunkową. Wartość entropii bezwarunkowej tego zbioru wynosi 3.056.

Tabela 4.7. Kompresja zbioru tekstowego koderem arytmetycznym z modelem statystycznym dowolnego rzędu. Zamieszczono wartości średnich bitowych skompresowanego pliku o długości oryginalnej 1.1 MB, a także wartości średniej entropii warunkowej przy analogicznym rzędzie modelu Markowa. W ostatnim wierszu znajdują się efekty kompresji pliku o długości 50 kilobajtów (47647 bajtów), który jest pierwszą częścią tegoż zbioru tekstowego 1.1 MB (zresztą dość jednorodnego).

Rząd modelu	1	2	3	4	5	6	7	8	9	10
Średnia entropia warunkowa	2.03	1.55	1.25	0.97	0.56	0.34	0.25	0.24	0.23	0.19
Średnia bitowa pliku 1.1MB	2.04	1.61	1.31	1.11	1.02	0.96	0.92	0.85	0.84	0.84
Średnia bitowa pliku 50KB	2.49	2.08	1.90	1.81	1.81	1.82	1.83	1.81	1.83	1.85

Przedstawione rezultaty testów pokazują kilka interesujących faktów, które dobrze charakteryzują problem związku statystycznego modelowania ze skutecznością kompresji. Widać wyraźnie, że w kompresowanym zbiorze istnieją silne zależności pomiędzy danymi i zwiększanie rzędu modelu w koderze arytmetycznym powoduje sukcesywny wzrost efektywności kodowania. Proces ten jednak nasyca się przy modelach rzędu 9 i 10. Widać także, iż uzyskiwane średnie bitowe tym są bliższe wartości entropii, im mniejszy jest rząd modelu statystycznego. Znaczy to, że model statystyczny jest lepiej określony dla niskich rzędów ze względu na małą ilość stanów takiego modelu. Zwiększając rząd obejmujemy więc coraz szerszą gamę zależności pomiędzy danymi, ale nie jesteśmy w stanie jej wykorzystać ze względu na słabość tak szerokiego modelu statystycznego. Wartość średniej bitowej kodu oddala się więc stopniowo od granicznej wartości entropii dochodząc w pewnym momencie (dla rzędu modelu równego 9) do granic możliwości takiego modelu przy określonej, skończonej długości kodowanego strumienia.

W przypadku krótszych zbiorów danych zjawisko rozrzedzenia kontekstu występuje znacznie szybciej. Widać to na przykładzie kompresji pierwszej części pliku 1.1 MB z przykładu 4.3 (wydzielonych zostało pierwszych 50 KB pliku). Ponieważ jest to dość jednorodny plik, można przyjąć że informacja jest w przybliżeniu równomiernie rozłożona wzdłuż pliku zachowując swój charakter, a więc model zależności pomiędzy danymi sięga także w pierwszej, wydzielonej części zbioru, rzędu 9-10. Jednak koder arytmetyczny już przy rzędzie 4 uzyskał minimalną wartość średniej bitowej kodu, a dalsze zwiększanie rozmiaru kontekstu modelu statystycznego nie tylko nie przyniosło poprawy skuteczności

kodowania, ale nawet ją pogorszyło przy jednocześnie znacznie zwiększonych kosztach czasowych.

Aby zapobiec zjawisku rozrzedzenia kontekstu przy jednoczesnym uwzględnieniu w modelu realnych zależności pomiędzy danymi leżącymi w nieco dalszym sąsiedztwie stosuje się różne metody modelowania kontekstu oraz jego kwantyzacji. Modelowanie kontekstu polega na tworzeniu kontekstu o rozmiarze i kształcie zapewniającym pokrycie fragmentu strumienia danych dotychczas kodowanego we wszystkich istotnych obszarach zawierających informacje o kodowanym aktualnie symbolu. W następującym potem procesie kwantyzacji kontekstu tworzona jest pewna pośrednia reprezentacja na bazie zbioru wszystkich danych skorelowanych z daną aktualnie kodowaną, która może być opisana modelem statystycznym o znacząco mniejszej liczbie stanów. Ta nowa reprezentacja opiera się więc zazwyczaj na przyczynowym otoczeniu o znacznie mniejszym rozmiarze i ma silnie zredukowany alfabet symboli. Jednak na jej postać mają wpływ wszystkie dane objęte procesem modelowania z właściwie dobranymi wagami (np. metodą regresji liniowej). Reprezentacja ta staje się kontekstem dla modelu statystycznego, użytego w koderze.

Oznaczmy rząd kontekstu danych skorelowanych przez m_1 (rząd kontekstu modelowanego), rząd kontekstu reprezentacji pośredniej przez m_2 (rząd kontekstu skwantowanego), a alfabety danych należących do obu kontekstów odpowiednio przez A_1 i A_2, przy czym leżą one w przestrzeni metrycznej. Redukcję rzędu kontekstu z m_1 do m_2 nazwiemy kwantyzacją rozmiaru kontekstu, a redukcję alfabetu z A_1 do A_2- redukcją alfabetu.

W prostym przykładzie można sobie wyobrazić, że pośrednia reprezentacja przy kodowaniu każdej kolejnej danej jest tworzona poprzez liniową kombinację trzech wartości bezpośrednio poprzedzających ją w strumieniu, a alfabet wejściowego źródła informacji jest 256-elementowy. Wagi bliższych sąsiadów są większe, a dla dalszych - stopniowo maleją. W takim przypadku $m_1 = 3$ jest redukowany do $m_2 = 1$, czyli uzyskaliśmy zmniejszenie rzędu modelu prawdopodobieństw warunkowych w koderze z 3 na 1. Jeśli współczynniki wag w funkcji określającej pośrednią reprezentacje zostały tak dobrane, że alfabet nie uległ zmianie, czyli $A_1 = A_2$, wtedy mamy zmniejszenie ilości stanów modelu statystycznego z $|A_1|^{m_1} = 256^3$ do $|A_2|^{m_2} = 256^1 = 256$. Zmniejszyło się więc zagrożenie rozrzedzeniem kontekstu, nawet w przypadku krótkich zbiorów. Możemy teraz zmniejszyć jeszcze bardziej liczbę stanów modelu poprzez zmniejszenie czterokrotnie liczby symboli w alfabecie A_2 w prostej operacji kwantyzacji wtórnej (skalarnej, równomiernej) wartości symboli tego alfabetu. Jeśli więc oznaczymy przez x_k kodowany aktualnie symbol, to kwantyzacja modelowanego kontekstu rzędu 3, zarówno co do rozmiaru jak i alfabetu wygląda następująco:

$$x'_{k-1} = \text{round}\left(\frac{a_1 x_{k-1} + a_2 x_{k-2} + a_3 x_{k-3}}{4} \right), \tag{4.8}$$

gdzie a_i - współczynniki określające wagę informacji wnoszonej przez element x_{k-i} modelowanego kontekstu. Ostateczna postać elementów kontekstu kwantowanego x'_{k-1} - o alfabecie liczącym $|A_2| = 64$, na który mają wpływ trzy symbole poprzedzające x_k - jest wykorzystywana do wyboru właściwej linii prawdopodobieństw, a następnie modyfikacji modelu statystycznego rzędu 1. Tak kwantowany kontekst może dać lepsze wyniki kompresji niż standardowy kontekst pierwszego rzędu x_{k-1} o alfabecie z 256-ciu elementami.

Zalety statystycznego modelowania źródła informacji poprzez odpowiedni dobór kontekstu i optymalizację alfabetu modelu pośredniego są widoczne szczególnie przy kompresji obrazów, stąd też bardziej złożone metody definiujące kontekst początkowy oraz schematy kwantyzacji kontekstu zostały przedstawione w rozdziale 7.

Bibliografia

1. C.E. Shannon, *A Mathematical Theory of Communication*, Bell System Technical Journal, 27:379-423, 623-656, 1948.
2. F. Jelinek, *Probabilistic Information Theory*, New York: McGraw-Hill, 1968.
3. R. Pasco, *Source Coding Algorithms for Fast Data Compression*, rozprawa doktorska w Stanford University, 1976.
4. J.J. Rissanen, *Generalized Kraft Inequality and Arithmetic Coding*, IBM Journal of Research and Development, 23(2):149-162, 1979.
5. J.J. Rissanen, G.G. Langdon, *Arithmetic Coding*, IBM Journal of Research and Development, 20:198-203, 1976.
6. I.H. Witten, R.M. Neal, and J. Cleary, *Arithmetic coding for data compression*, Commun. ACM, 30(6):520-540, 1987.
7. M. Nelson, *The Data Compression Book*, M&T Books, rozdz. 5 i 6, 1991.