ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1999
Seria: INFORMATYKA z. 37 Nr kol. 1421

Roman STAROSOLSKI
Politechnika Śląska, Instytut Informatyki

SZYBKIE, BEZSTRATNE ORAZ ADAPTACYJNE METODY KOMPRESJI OBRAZÓW W ODCIENIACH SZAROŚCI

Streszczenie. Artykuł prezentuje wyniki badań mających na celu opracowanie szybkiego, bezstratnego i adaptacyjnego algorytmu kompresji obrazów w odcieniach szarości. Cel ten planowano osiągnąć poprzez odpowiedni dobór algorytmów stanowiących kolejne etapy kompresji obrazu oraz przez wprowadzenie modyfikacji do już istniejących algorytmów kompresji. Niniejszy artykuł zawiera opis zastosowanych algorytmów i wyniki badań nad wyborem algorytmów składowych kompresora.

FAST, LOSSLESS AND ADAPTIVE METHODS OF GRAYSCALE IMAGE COMPRESSION

Summary. This paper presents the research on fast, lossless and adaptive algorithms of grayscale image compression. We examine various algorithms as the consecutive components of the image compression process. We introduce improvements to the existing data compression algorithms. In this paper we describe the algorithms used and report the results of the selection of the component algorithms.

1. Wprowadzenie

Opisywane w niniejszym artykule badania stanowią część projektu badawczego pt. „Szybkie, bezstratne oraz adaptacyjne metody kompresji obrazów w odcieniach szarości" finansowanego przez Komitet Badań Naukowych (grant nr 8 T11C 029 12). Podstawowym celem projektu było opracowanie szybkiego, bezstratnego i adaptacyjnego algorytmu kompresji obrazów w odcieniach szarości. Cel ten planowano osiągnąć poprzez zmodyfikowanie istniejących algorytmów kompresji, jak również dzięki zastosowaniu dodatkowego etapu kompresji — poprzedzenie kodowania statystycznego przez szybszy i prostszy algorytm

kompresji, który ma za zadanie zmniejszyć ilość danych przetwarzanych przez kompresor statystyczny. Niniejszy artykuł zawiera opis zastosowanych algorytmów i wyniki badań nad wyborem algorytmów składowych kompresora. W wyniku opisanych w niniejszym artykule badań wyselekcjonowano dwa algorytmy. Badania wybranych algorytmów są tematem oddzielnych opracowań [14, 15].

Kompresja jest procesem usuwania nadmiarowości z plików danych. Nadmiarowość jest naturalna dla niektórych typów danych, takich jak teksty pisane, gdzie często powtarzają się niektóre wyrazy lub ich fragmenty, czy też obrazy, w których sąsiadujące piksele mają zazwyczaj zbliżoną barwę. Dzięki kompresji można oszczędniej wykorzystywać zarówno pamięci masowe, jak i szerokości pasm przenoszenia sieci komputerowych. Kosztem uzyskania tych oszczędności jest wykorzystanie pamięci operacyjnej i czasu procesora komputera.

Istnienie szybkich, adaptacyjnych algorytmów kompresji obrazów konieczne jest w systemach, które, dysponując ograniczonymi zasobami pamięci i mocy obliczeniowej, muszą przetwarzać dane napływające w czasie rzeczywistym, np. podczas akwizycji obrazu wideo. Szybki algorytm kompresji może także okazać się nieoceniony w systemach, gdzie kompresja i dekompresja dużej ilości danych odbywa się wielokrotnie i czas jej trwania decyduje o komforcie pracy użytkownika.

W wielu praktycznych zastosowaniach współczynnik kompresji uzyskiwany przez algorytm jest tylko jednym spośród kilku kryteriów wyznaczających jego przydatność. W systemach transmisji danych z urządzeń akwizycji obrazu (jak skaner czy kamera) do urządzeń, takich jak drukarka, lub przy transmisji obrazów w sieciach komputerowych wąskim gardłem jest szerokość pasma przenoszenia medium transmisyjnego. Zastosowanie odpowiednio szybkiego algorytmu kompresji pozwala na przesłanie większej ilości danych w jednostce czasu, przez co umożliwia zwiększenie wydajności całego systemu. Współczynnik kompresji jest również ważny, ale niewielka jego zmiana w wyżej wymienionych zastosowaniach wydaje się nieistotna.

Kolejnym ważnym parametrem jest złożoność pamięciowa algorytmu, która musi odpowiadać ograniczonym zasobom infrastruktury. Dla praktycznych zastosowań istotne jest także działanie algorytmu w najgorszym przypadku: pesymistyczna złożoność pamięciowa i obliczeniowa oraz maksymalna możliwa ekspansja danych.

1.1. Klasyfikacja algorytmów kompresji

Istnieje wiele kryteriów, które pozwalają na klasyfikację algorytmów kompresji. Na podstawie kryterium odwracalności procesu kompresji algorytmy kompresji można podzielić na **stratne** i **bezstratne**. Działanie algorytmów stratnych nie jest w pełni odwracalne i dlatego ich zastosowanie ograniczone jest do przypadków, gdy dopuszczalne są niewielkie różnice

pomiędzy tym co kompresowaliśmy, a tym co uzyskaliśmy po dekompresji. Nadają się one przede wszystkim do danych typu obraz lub dźwięk, gdzie, wykorzystując niedoskonałości zmysłów człowieka, sprawiają wrażenie bezstratności. Algorytmy bezstratne pozwalają natomiast na wielokrotną kompresję i dekompresję bez utraty informacji. Znajdują zastosowanie wszędzie tam, gdzie przetwarzane dane charakteryzują się pewną nadmiarowością, a ich rozmiar nie jest do zaniedbania. Doskonałym przykładem tego typu danych są obrazy i pliki tekstowe. Istnieje także klasa algorytmów wykorzystujących obie powyższe techniki. Są to algorytmy **LPL** (ang. lossy plus lossless). W ich przypadku dane najpierw kompresuje się stratnie, uzyskując silnie skompresowaną „miniaturę" oryginalnych danych, którą można np. wykorzystać do szybkiego przeglądania archiwów obrazów. Następnie wyznacza się straty, tj. różnice pomiędzy oryginalnymi danymi a tymi zdekompresowanymi po kompresji stratnej. Różnice kompresuje się algorytmem bezstratnym. Na podstawie danych skompresowanych stratnie i bezstratnie skompresowanych strat można odtworzyć oryginalne dane bezstratnie.

Spośród wielu bezstratnych algorytmów kompresji danych najczęściej stosowane i najlepiej poznane są dwie grupy algorytmów: algorytmy statystyczne i słownikowe. Algorytmy **statystyczne** na podstawie rozkładu prawdopodobieństw symboli przydzielają poszczególnym symbolom alfabetu kody o różnej długości. Za określenie prawdopodobieństwa wystąpienia danego symbolu odpowiedzialny jest tzw. **model danych. Rząd modelu danych** jest zerowy, jeżeli prawdopodobieństwo danego symbolu wyznacza się niezależnie od symboli poprzedzających go. Modele rzędów niezerowych wykorzystują prawdopodobieństwo warunkowe — tutaj rząd modelu określa, ile poprzednich symboli (określanych mianem **kontekstu**) bierze udział w wyznaczeniu prawdopodobieństwa wystąpienia symbolu aktualnego. Shannon wykazał, że do zakodowania symbolu s potrzeba $-log_2(p(s))$ bitów, gdzie $p(s)$ to prawdopodobieństwo wystąpienia symbolu s. **Kody prefiksowe** (kody Huffmana, Golomba itp.) przydzielają całkowitą liczbę bitów symbolom. Kody nieprefiksowe mogą wykorzystać pojedynczy bit jako fragment kodu dwu lub nawet większej liczby symboli; przykładem tutaj może być kodowanie arytmetyczne. W algorytmach **słownikowych** (inaczej podstawieniowych, ang. substitutional compressors) rolę modelu danych pełni tzw. słownik. Kompresor po napotkaniu w ciągu wejściowym frazy, która już znajduje się w słowniku, na wyjście wyprowadza zamiast frazy jej indeks. Rozróżniamy dwa rodzaje algorytmów słownikowych:

- W algorytmach LZ77 i pochodnych (np. LZSS) rolę słownika pełni bufor z określoną liczbą ostatnio przeczytanych symboli. Aby jednoznacznie zidentyfikować frazę, trzeba podać zarówno jej pozycję w buforze, jak i jej długość.
- Algorytmy LZ78 i pochodne (np. LZW), dokładniej opisane w dalszej części artykułu, korzystają ze słownika bardziej odpowiadającego intuicyjnemu rozumieniu słowa "słownik" — do jednoznacznego zidentyfikowania frazy wystarczy jej indeks.

Czasami możliwe jest łączenie kilku technik, np. po kompresji słownikowej kompresja statystyczna często jest w stanie dokonać dalszej redukcji rozmiaru danych.

Na podstawie sposobu, w jaki dany algorytm adaptuje się do charakterystyki przetwarzanych danych, możemy wyróżnić algorytmy **adaptacyjne**, **statyczne** i **stałe** (ang. fixed). Algorytmy adaptacyjne budują model danych w miarę czytania danych od razu je kompresując. Dzięki temu dane skompresowane uzyskujemy na wyjściu kompresora z najwyżej niewielkim opóźnieniem w stosunku do danych czytanych przez kompresor. Algorytmy statyczne czytają dane dwukrotnie: za pierwszym razem budują model, a za drugim kompresują, korzystając z modelu zbudowanego na podstawie całego pliku danych. W tym przypadku skompresowane dane można uzyskać dopiero po jednokrotnym przeczytaniu przez kompresor całego pliku danych. Algorytmy stałe zakładają, że dane mają określoną charakterystykę i kompresują je za pomocą stałego modelu, którego zawartość nie zależy od rzeczywiście kompresowanych danych. Współczynnik kompresji uzyskiwany przez algorytm stały zależy nie tylko od entropii danych, ale również od tego, jak dobrze założony przez kompresor model odpowiada rzeczywistej charakterystyce danych.

Do charakteryzowania przetwarzanych danych często stosuje się pojęcie **entropii**. Entropia pliku danych to teoretyczny rozmiar tego pliku zakodowanego w sposób optymalny, czyli taki, że dany symbol s zakodowany jest na $-log_2(p(s))$ bitach, gdzie $p(s)$ to prawdopodobieństwo wystąpienia symbolu s. Definicja ta nic nie mówi o sposobie wyznaczania prawdopodobieństwa wystąpienia symbolu s. Zazwyczaj jest to prawdopodobieństwo warunkowe wyznaczone na podstawie znajomości skończonej i niewielkiej liczby symboli poprzedzających dany symbol — liczba symboli determinuje **rząd entropii**. Jeżeli prawdopodobieństwo wystąpienia danego symbolu wyznaczane jest niezależnie od pozostałych symboli, to mówimy wtedy o entropii rzędu zerowego. Entropię często wyraża się nie w bitach (liczba bitów potrzebnych do zakodowania całego pliku danych), ale w średniej liczbie bitów przypadających na pojedynczy symbol (np. bajt, piksel) pliku danych.

1.2. Szybka, bezstratna kompresja obrazów

Znanych jest wiele bezstratnych algorytmów kompresji obrazów, przegląd metod i standardów reprezentacji obrazów cyfrowych można znaleźć w monografiach [12, 13]. Bezstratne algorytmy kompresji obrazów są z reguły projektowane pod kątem uzyskania jak najlepszego współczynnika kompresji. Złożoność obliczeniowa i pamięciowa traktowana jest z mniejszą uwagą. Istnieją także algorytmy skonstruowane z myślą o uzyskaniu jak najmniejszej złożoności obliczeniowej przy możliwie niskiej degradacji współczynnika kompresji. Przykładem takiego algorytmu może być FELICS [3, 17], który uznawany jest obecnie [8] za bardzo szybki w porównaniu do innych bezstratnych algorytmów kompresji obrazów. Działa

on z prędkością algorytmu nie dedykowanego dla żadnego konkretnego rodzaju danych — Unix-owego programu compress wykorzystującego kodowanie słownikowe. Udane próby uzyskania algorytmów kompresujących z podobną prędkością, ale silniej (np. Progressive FELICS [4] czy LOCO-I [16]) sugerują, że można również skonstruować algorytmy kompresujące podobnie silnie, ale szybciej.

Nie spotyka się w literaturze opisów znacząco szybszych adaptacyjnych algorytmów kompresji szerokich klas obrazów. Istniejące szybsze algorytmy szybkość swą zawdzięczają korzystaniu ze stałego modelu danych. Zrezygnowanie z adaptacji do rzeczywistej charakterystyki danych powoduje, iż uzyskiwany przez kompresor współczynnik kompresji zbliżony jest do rezultatu kompresora adaptacyjnego tylko wtedy, gdy rzeczywiste dane odpowiadają założonemu modelowi. Dlatego też obszary zastosowań stałych algorytmów to wąskie klasy danych, np. obrazy termograficzne skanowane z wysoką rozdzielczością [1].

Algorytm będący przedmiotem badań (opisany dokładniej w dalszej części artykułu) jest w istocie rozszerzeniem często stosowanego, prostego schematu dekorelacja – kompresja statystyczna. W schemacie tym obraz jest przeglądany w kolejności rastra. Schemat ten polega na kompresji residuum algorytmem statystycznym. Residuum traktowane jest jako jednowymiarowy ciąg symboli. Kolejne symbole **residuum** wyznaczane są jako różnice pomiędzy rzeczywistą a przewidywaną jasnością piksela. Do przewidywania jasności danego piksela służy tzw. **predyktor**. Predyktor jest funkcją, do wyznaczenia wartości której można wykorzystać wszystkie już przetworzone piksele (lub mniejszą ich liczbę). Proces wyznaczania residuum nazywamy **dekorelacją**.

2. Projekt kompresora

Kompresor przeznaczony jest do naturalnych obrazów w odcieniach szarości. W badaniach ograniczono się do obrazów w formacie 8 bitów na piksel, tzn. w 256 stopniach szarości. Kompresja odbywa się w kilku (maksymalnie trzech) etapach:

a) Zastosowanie prostego predyktora, korzystającego z 1 do 3 wcześniej zakodowanych pikseli do wyznaczenia residuum.

b) Kompresja residuum za pomocą algorytmu RLE lub szybkiego kodowania słownikowego. Ten etap ma na celu istotne zmniejszenie ilości danych, tak by ostatni, najwolniejszy, fragment algorytmu miał do przetworzenia mniejszą ilość danych.

c) Kompresja statystyczna (najbardziej kosztowny etap).

W etapie b) można dokonać kompresji za pomocą odmiany algorytmu RLE o 4-bitowych kodach lub kompresji słownikowej. Przebadano kompresory słownikowe LZ78

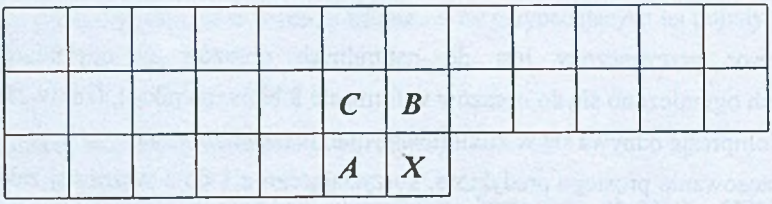
i LZW, w których struktura słownika została zrealizowana w postaci uproszczonego drzewa wektorowego. Do podstawowych algorytmów LZ78 i LZW wprowadzono nowy mechanizm adaptacji. W etapie c) został wykorzystany, uznawany za szybki, kompresor statystyczny algorytmu FELICS [3]. Algorytm kompresora został uzupełniony o kilka oryginalnych modyfikacji umożliwiających zwiększenie szybkości kompresji oraz szybszą adaptację kompresora do charakterystyki danych na początku kompresji. Koncepcja ogólnego algorytmu przewiduje kompresję z wykorzystaniem wszystkich trzech etapów lub mniejszej ich liczby.

Algorytmy zostały rozbudowane o mechanizm ograniczający maksymalną ekspansję danych. Algorytmy kompresujące przetwarzają całe wiersze obrazu. W przypadku gdy dochodzi do ekspansji danych, wyprowadzany jest specjalny znacznik i nieskompresowane symbole. Takie rozwiązanie pociąga za sobą kilka konsekwencji:

- Przyspieszenie działania, gdyż odwołania do procedur systemu operacyjnego odbywają się rzadziej niż raz na każdy symbol.
- Zapisując skompresowane dane wierszami, możemy zmarnować do 7 bitów na pojedynczy wiersz obrazu wejściowego.
- Komplikuje się nieco algorytm dekompresora, ponieważ musi on dekodować zadaną liczbę symboli zamiast dekodowania bloku skompresowanych symboli o zadanym rozmiarze.

Do skompresowanych danych nie są dołączane sumy kontrolne ani korekcyjne, ale format skompresowanych danych zawiera niewielką ilość dodatkowej informacji umożliwiającej wykrycie błędów.

2.1. Dekorelacja



Rys. 1. Położenie pikseli używanych w funkcji predyktora
Fig. 1. Locations of pixels used by the predictor function

Wartość funkcji predyktora dla danego piksela X wyznaczana jest na podstawie maksimum 3· najbliższych sąsiadów (lewy — A, lewy górny — C i górny — B). Predyktory wyliczane są (tabela 1) jedynie przez proste operacje stałopozycyjne (dodawania, odejmowania, przesunięcia).

Tabela 1

Predyktory stosowane w badaniach

$Pred0(X) = 0$	$Pred4(X) = A+B-C$	$Pred8(X) = 0.75A+0.75B-0.25C$
$Pred1(X) = A$	$Pred5(X) = A+(B-C)/2$	$Pred9(X) = 2B-A$
$Pred2(X) = B$	$Pred6(X) = B+(A-C)/2$	$Pred10(X) = 1.5B-0.5A$
$Pred3(X) = C$	$Pred7(X) = (A+B)/2$	$Pred11(X) = 0.25A+0.75B$

Do badań użyto standardowych predyktorów algorytmu Lossless JPEG (*Pred0* – *Pred7*) [6]. *Pred8* to często używany predyktor [7, 18]. Predyktory *Pred9*, *Pred10* i *Pred11* zostały tak zaprojektowane, by ich wartość zależała w większym stopniu od górnego sąsiada aktualnego piksela, a w mniejszym od sąsiada lewego.

Jeżeli w funkcji predyktora występuje odejmowanie, to wartość predyktora może przekroczyć dopuszczalny zakres wartości symboli. Należy wtedy użyć najbliższej dopuszczalnej wartości symbolu.

Nie dla każdego piksela obrazu możliwe jest wyznaczenie wartości wszystkich predyktorów. Dlatego też lewy górny piksel obrazu dekorelowany jest zawsze predyktorem *Pred0*. Dla pozostałych pikseli pierwszego wiersza obrazu stosujemy *Pred1* (o ile nie chcemy użyć *Pred0* do dekorelacji całego obrazu), a dla pierwszej kolumny obrazu *Pred2* (*Pred0* jeżeli chcemy użyć *Pred0* lub *Pred1* dla całego obrazu).

Kompresji podlega symbol residuum R będący różnicą pomiędzy przewidywaną a rzeczywistą jasnością piksela:

$R=X-Pred(X)$.

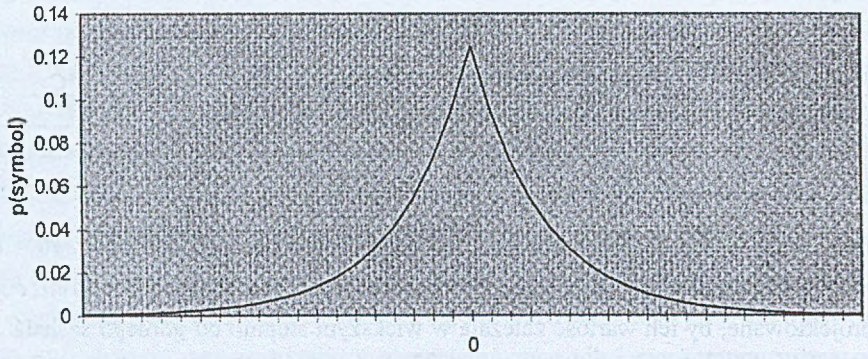
Ponieważ zarówno X, jak i $Pred(X)$ są wartościami z zakresu $\langle 0, 255\rangle$, zatem R może przyjmować wartości z zakresu $\langle -255, 255\rangle$. Taki symbol wymagałby 9 bitów do zakodowania binarnego, powodując ekspansję (8-bitowych) danych jeszcze przed rozpoczęciem kompresji. Dla konkretnej wartości predyktora R może przyjmować tylko 256 wartości (z przedziału $\langle -Pred(X), 255-Pred(X)\rangle$). Podczas dekompresji obraz jest odtwarzany z użyciem wartości predyktora i zdekompresowanego symbolu residuum:

$X=R+Pred(X)$.

W tym przypadku wartości residuum znajdują się w tym samym przedziale o długości 256 (dla konkretnej wartości predyktora suma mieści się w przedziale $\langle 0, 255\rangle$). Dzięki temu można zastąpić wyżej wymienione wzory przez:

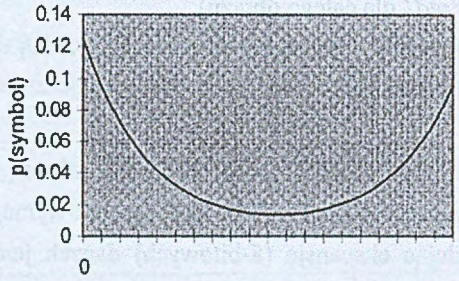
$R=(X-Pred(X)) \bmod 256$ oraz $X=(R+Pred(X)) \bmod 256$.

Operacja dzielenia modulo jest obliczeniowo bardzo prosta, jeżeli użyjemy 8-bitowych zmiennych, to na procesorach wykonujących obliczenia w kodzie uzupełnieniowym do dwóch dzielenie modulo 256 wykona się automatycznie bez dodatkowych kosztów.



Rys. 2. Rozkład prawdopodobieństwa symboli po dekorelacji
Fig. 2. The probability distribution after the decorrelation

Kompresor statystyczny i rodzina kodów, które zostaną użyte do kompresji residuum, wymagają, by rozkład prawdopodobieństwa symboli był malejący (najlepiej wykładniczy). Po dekorelacji rozkład jest zbliżony do symetrycznego rozkładu wykładniczego (rys. 2). Po dzieleniu modulo przestaje nim być (rys. 3).

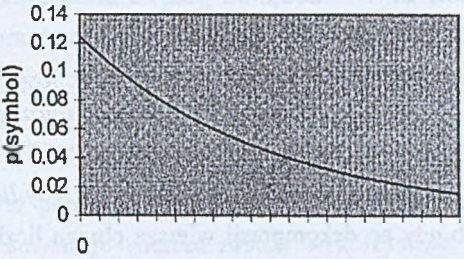


Rys. 3. Rozkład prawdopodobieństwa po dzieleniu modulo
Fig. 3. The probability distribution after the modulo reduction

Musimy więc zmienić kolejność symboli, by ponownie uzyskać rozkład malejący. Robimy to kolejno układając symbole: pierwszy, ostatni, drugi, przedostatni i tak dalej:

$$R2 = \begin{cases} R*2 & dla\ R<128 \\ (256-R)*2-1 & dla\ R>=128 \end{cases}$$

Powyższe wyrażenie jest łatwe do wyliczenia, może być wyznaczane jeszcze szybciej po umieszczeniu w tablicy (dziedzina przekształcenia to tylko 256 punktów). W ten sposób uzyskamy rozkład prawdopodobieństwa zbliżony do wykładniczego (rys. 4).



Rys. 4. Powrót do rozkładu malejącego
Fig. 4. Back to the descending probability distribution

Do dekorelacji obrazu o długości wiersza *dim_x* potrzebny jest bufor wyjściowy o długości *dim_x*. Wartość predyktora wyznaczana jest na podstawie wartości sąsiednich pikseli — musimy więc przechowywać przynajmniej *dim_x*+2 piksele obrazu. Ponieważ założyliśmy przetwarzanie całymi wierszami, będziemy w pamięci przechowywać 2 wiersze obrazu (aktualny i poprzedni): *dim_x**2.

2.2. Kompresja RLE

Algorytm RLE zastępuje ciągi powtarzających się symboli parami *<liczba powtórzeń, powtarzany symbol>*. Ponieważ kompresujemy digitalizowane obrazy naturalnych scen, w przetwarzanych danych zawsze będzie występował szum. Szum ten powoduje, iż ciągi powtarzających się symboli będą zazwyczaj bardzo krótkie. Będą także występowały ciągi pojedynczych różnych symboli. Reprezentacja takich ciągów za pomocą ciągów par *<jedno powtórzenie, symbol>* może prowadzić do ekspansji danych. Dlatego też klasyczny schemat RLE, w którym liczba powtórzeń zapisana jest na 8-u bitach, został zmodyfikowany w następujący sposób: podczas kompresji rozpoznajemy zarówno ciągi powtarzających się symboli, jak i ciągi różnych symboli. Rozpoznany ciąg kodujemy jako *<kod_rle, powtarzany symbol>* lub *<kod_rle, ciąg różnych symboli>*. *Kod_rle* umożliwia wyznaczenie zarówno liczby symboli w ciągu, jak i rodzaju ciągu (różne symbole / powtórzenia symbolu).

W przyjętym rozwiązaniu *kod_rle* ma długość czterech bitów, jego wartość *<0, 15>* interpretowana jest następująco:

wartości *<0, x>* — ciąg powtórzeń o długości *kod_rle* +2,
wartości *<x+1, 15>* — ciąg różnych symboli o długości *kod_rle* –*x*.

Aby umożliwić rozdzielenie fazy kompresji RLE od kompresji statystycznej, kody RLE wiersza obrazu i symbole zapisywane są w oddzielnych obszarach pamięci. Dzięki temu prostszy jest także zapis symbolu do pamięci (symbol zawsze zajmuje 1 cały bajt). Postać

wiersza obrazu po kompresji jest następująca:

<liczba symboli (2 bajty), symbole, kody_rle>

lub gdy doszło do ekspansji danych:

< liczba symboli (2 bajty), nieskompresowany wiersz obrazu>

gdzie *liczba symboli* równa się długości wiersza obrazu.

Błąd w skompresowanych danych może zostać wykryty, gdy *liczba symboli* jest większa od szerokości obrazu lub gdy po dekompresji wiersza obrazu liczba przeczytanych symboli różna jest od *liczby symboli*.

Maksymalna możliwa ekspansja danych wynosi 2 bajty na wiersz obrazu. Do kompresji wiersza o długości *dim_x* potrzebny jest bufor wyjściowy o długości *dim_x*+2 i pomocniczy bufor na *kody_rle* o długości *dim_x*/2.

2.3. Kompresja słownikowa

Pierwotnie planowano użycie jedynie algorytmu LZ78. Ostatecznie użyto również algorytmu LZW. Po kompresji LZW nie ma bezpośredniej możliwości dalszej kompresji słownikowej. Oba algorytmy to bezstratne kompresory słownikowe, w wersji podstawowej na początku działania adaptacyjne, później stałe. Algorytm LZW, zastosowana struktura danych słownika i nowy mechanizm adaptacji są tematem oddzielnej pracy [15] — tutaj zostały opisane w sposób mniej szczegółowy.

2.3.1. Kompresja LZ78

Algorytm LZ78 można w uproszczeniu przedstawić za pomocą pseudokodu z rys. 5.

```
opróżnij słownik
f := łańcuch pusty
while not eof(plik wejściowy)
    pobierz symbol S
    if fS znajduje się w słowniku
        f := fS
    else
      wyprowadź parę <kod frazy f, S>
      dodaj fS do słownika
      f := łańcuch pusty
    endif
endwhile
```

Rys. 5. Kodowanie LZ78
Fig. 5. The LZ78 coding

W przyjętym rozwiązaniu słownik oparty na uproszczonym drzewie trie ma dużą złożoność pamięciową (struktura danych słownika jest wspólna dla LZ78 i LZW). Kod frazy ma długość $\lceil \log_2(\text{aktualna długość słownika}) \rceil$. Założono maksymalny rozmiar słownika 2^{16} fraz, zatem pojedynczy symbol może zostać zapisany na 24 bitach (pusta fraza na 16 bitach, symbol na 8).

Skompresowany wiersz poprzedzany jest bitem o wartości 0. Jeżeli nie wszystkie bity ostatniego bajta skompresowanego wiersza zostały wykorzystane, to niewykorzystane bity wypełniane są zerami. Skompresowany wiersz zawsze zajmuje całkowitą liczbę bajtów. W przypadku gdy następuje ekspansja kompresowanego wiersza, wyprowadzany jest specjalny znacznik (bajt o wartości 11111111b) i nieskompresowany wiersz.

Wykrycie błędu możliwe jest wtedy, gdy pierwszy bit wiersza jest równy 1, a wśród pozostałych występuje przynajmniej jeden bit różny od 1. Błąd zostanie wykryty również wówczas, gdy dekompresowana fraza będzie dłuższa niż jeszcze nie zdekompresowany fragment wiersza lub gdy po dekompresji wiersza nie wszystkie niewykorzystane bity ostatniego bajta skompresowanego wiersza będą równe 0.

Maksymalna ekspansja danych wynosi 1 bajt na wiersz obrazu. Do kompresji wiersza o długości *dim_x* potrzebny jest bufor wyjściowy o długości *dim_x**3+1 oraz obszar pamięci przechowujący słownik.

2.3.2. Kompresja LZW

Algorytm LZW to modyfikacja algorytmu LZ78 polegająca na wstępnym wypełnieniu słownika jednoznakowymi frazami z wszystkimi symbolami alfabetu. Dzięki tej modyfikacji wyprowadzane są tylko kody fraz — algorytm LZW kompresuje silniej niż LZ78. Dekompresja nie jest już tak prosta jak w algorytmie LZ78 (dokładniejszy opis można znaleźć np. w pracy [10]).

2.3.3. Mechanizm adaptacji

Słownik, w miarę postępowania kompresji, wypełnia się nowymi frazami aż osiągnie maksymalny rozmiar — do momentu wypełnienia słownika algorytm jest adaptacyjny. W przypadku gdy słownik jest pełny, można zaprzestać dodawania nowych fraz, tzn. „**zamrozić słownik**". Po zamrożeniu słownika dla reszty pliku algorytm będzie stały. Można także powtórnie zainicjalizować słownik i budować go od nowa dla nowych danych. W istniejących implementacjach (np. w kompresorze Unix compress) łączy się oba rozwiązania. Po wypełnieniu słownika zamraża się go i monitoruje się chwilowy współczynnik kompresji. Powtórną inicjalizację słownika wykonuje się, gdy zostanie wykryte pogorszenie się wartości chwilowego współczynnika kompresji.

W przyjętym rozwiązaniu, po dodaniu frazy do słownika, jeżeli słownik jest pełny (osiągnął już maksymalny rozmiar), jest z niego usuwana jedna fraza, która nie jest prefiksem żadnej innej. W przypadku algorytmu LZW usuwana fraza musi mieć przynajmniej 2 symbole. Umożliwienie kasowania takich fraz nieznacznie skomplikowało i rozbudowało strukturę danych słowika, który oprócz fraz musi przechowywać tablicę indeksów fraz nadających się do usunięcia.

Po wypełnieniu słownika kompresor może dodawać nową frazę do słownika za każdym razem, gdy wyprowadza znalezioną frazę lub rzadziej. Po dodaniu frazy do słownika losowana jest liczba zaniechań kolejnych operacji aktualizacji słownika. Mechanizm ten, również stosowany w przypadku kompresora statystycznego, jest dokładniej opisany w dalszej części artykułu, w punkcie „Kompresja statystyczna".

2.3.4. Struktura danych słownika

Do przechowywania fraz zastosowany został słownik w postaci uproszczonego drzewa wektorowego [11], stosowane jest też określenie pochodzące z języka angielskiego: drzewo trie [2]. Struktura ta pozwala na zmodyfikowanie podstawowych algorytmów LZW i LZ78 — zmniejszenie liczby wyszukiwań w słowniku z jednego dla każdego symbolu kompresowanego ciągu do jednego na wyprowadzony indeks frazy. Frazy umieszczane w słowniku przez algorytmy LZW i LZ78 mają jedną, istotną ze względu na strukturę danych słownika, cechę: jeżeli w słowniku jest fraza *f*, to muszą się tam znajdować wszystkie jej prefiksy. Ta cecha pozwala na uproszczenie struktury drzewa trie — wystarczy tylko jeden rodzaj węzła i nie musimy uzupełniać alfabetu o symbol „koniec frazy".

W celu umożliwienia wymaganego przez mechanizm adaptacji szybkiego wyszukiwania i usuwania fraz nie będących prefiksami innych fraz, czyli będących liśćmi drzewa trie, strukturę słownika uzupełniono o tablicę liści.

Węzły drzewa trie umieszczono w tablicy. Maksymalną liczbę fraz przechowywanych w słowniku ograniczono do 2^{16}, dzięki czemu wskaźniki można było zastąpić 16-bitowymi indeksami. Węzeł drzewa zawiera, oprócz wskaźników na synów także, przydatne jedynie przy dekompresji pole zawierające symbol reprezentowany przez aktualny węzeł, oraz potrzebne do szybkiego usuwania liści pola: wskaźnik na rodzica, licznik synów i wskaźnik (indeks) do tablicy liści.

Wadą zastosowanej struktury danych słownika jest jej duża złożoność pamięciowa. W przypadku 8-bitowego alfabetu każda fraza (węzeł) zajmuje 521 bajtów. Warto zauważyć, że w dekompresji nie jest wykorzystywane drzewo trie. Złożoność pamięciowa słownika do dekompresji spada do 9 bajtów na frazę.

2.4. Kompresja statystyczna

Algorytm kompresji statystycznej jest tematem oddzielnej pracy [14]; tutaj został opisany w sposób mniej szczegółowy.

Zastosowano zmodyfikowany kompresor statystyczny pierwszego rzędu z algorytmu FELICS [3]. Kompresor ten koduje symbole o malejącym rozkładzie prawdopodobieństwa za pomocą parametrycznej rodziny kodów.

Reakcję na ekspansję kompresowanego wiersza rozwiązano w taki sam sposób, jak w przypadku algorytmu LZ78 — maksymalna ekspansja danych wynosi 1 bajt na wiersz obrazu. Skompresowany wiersz uzupełniany jest bitami o wartości 0 do granicy pełnego bajta.

Wykrycie błędu możliwe jest wtedy, gdy pierwszy bit wiersza jest równy 1, a wśród pozostałych bitów pierwszego bajta występuje przynajmniej jeden bit różny od 1. Błąd może również zostać wykryty, gdy po dekompresji wiersza nie wszystkie niewykorzystane bity ostatniego bajta skompresowanego wiersza będą równe 0.

2.4.1. Rodzina kodów

Zastosowano parametryczną rodzinę kodów zbudowaną w oparciu o zmodyfikowane kody Golomba i Rice'a. Kody Golomba i Rice'a (tabela 2) charakteryzowane są przez tylko jeden parametr.

Tabela 2

Kody Golomba i Rice'a

Golomb / Rice		m=1 k=0	m=2 k=1	m=3	m=4 k=2
x =	0	0•	0•0	0•0	0•00
	1	10•	0•1	0•10	0•01
	2	110•	10•0	0•11	0•10
	3	1110•	10•1	10•0	0•11
	4	11110•	110•0	10•10	10•00
	5	111110•	110•1	10•11	10•01
	6	1111110•	1110•0	110•0	10•10
	7	11111110•	1110•1	110•10	10•11
	8	111111110•	11110•0	110•11	110•00

Oznaczany przez m parametr kodu Golomba jest całkowitą liczbą dodatnią. Aby zakodować nieujemną, całkowitą liczbę x kodem Golomba z parametrem m, najpierw kodujemy $\lfloor x/m \rfloor$ unarnie, po czym kodujemy x mod m za pomocą zmodyfikowanego kodu binarnego. Zmodyfikowany kod binarny (ang. adjusted binary code) użyty do kodowania liczb z zakresu

<0, N-1> przyporządkowuje części liczb z tego zakresu kody o długości $\lfloor \log_2(N) \rfloor$ bitów, pozostałe liczby kodowane są na $\lceil \log_2(N) \rceil$ bitach. Zmodyfikowany kod binarny staje się zwykłym kodem binarnym, gdy N jest potęgą dwójki.

Parametrem kodów Rice'a jest nieujemna całkowita liczba k. Kody Rice'a to szczególny przypadek kodów Golomba, gdy $m = 2^k$. Kody te są łatwiejsze do generacji, gdyż operację dzielenia można zastąpić przez przesunięcie o k bitów w prawo. Zamiast dzielenia modulo i kodowania zmodyfikowanym kodem binarnym można bezpośrednio (binarnie) zakodować k najmniej znaczących bitów x.

Zmodyfikowano parametryczne rodziny kodów Golomba i Rice'a. Rodziny kodów Golomba i Rice'a przeznaczone są do kodowania symboli o wykładniczym rozkładzie prawdopodobieństwa. Kompresując rzeczywiste obrazy, mamy do czynienia ze skończonym alfabetem symboli o rozkładzie jedynie zbliżonym do wykładniczego.

Kody Golomba użyte do kodowania symboli skończonego alfabetu (256 symboli w tym przypadku) nie wykorzystują wszystkich możliwych sekwencji bitów. Prefiksem kodów symboli o najwyższych wartościach (jest ich od 1 do m) jest $\lfloor 255/m \rfloor$ zakodowane unarnie, czyli $\lfloor 255/m \rfloor$ jedynek i jedno zero. Ponieważ nie ma kodów zaczynających się od $\lfloor 255/m \rfloor+1$ jedynek (byłyby, gdybyśmy kodowali symbole z nieskończonego alfabetu), to symbole o najwyższych wartościach można kodować ze zmienionym prefiksem składającym się z $\lfloor 255/m \rfloor$ jedynek. Jeżeli 256 mod m jest różne od zera, to liczba symboli o najwyższych wartościach kodowanych z tym samym prefiksem jest mniejsza od m — wtedy kodując te symbole używamy zmodyfikowanych kodów binarnych dla zakresu 256 mod m zamiast dla zakresu m. Modyfikacja ta powoduje wstawienie bezpośrednio (binarnie) zakodowanych symboli do rodziny (parametr m=128). Dzięki niej kompresor będzie w stanie zaadaptować się do danych nie podlegających kompresji i nie powodować ich ekspansji. Niezmodyfikowane kody Golomba zastosowane do kodowania symboli o równomiernym rozkładzie prawdopodobieństwa powodują ekspansję danych przynajmniej o 0.5 bita na symbol.

Ograniczono rozmiar rodziny i zmieniono kolejność występowania kodów w rodzinie. Kody Golomba pozwalają na uzyskanie nieznacznie lepszych współczynników kompresji niż kody Rice'a. Użycie wszystkich możliwych kodów Golomba zwiększyłoby złożoność pamięciową i obliczeniową kompresora. Zazwyczaj około 20 początkowych kodów z rodziny Golomba jest wykorzystywanych podczas kompresji obrazów. Użycie tylko pewnej liczby początkowych kodów pogorszyłoby uzyskany współczynnik kompresji, gdyby rzeczywiste dane wymagały kodu o dużym parametrze m, który został odrzucony. Dlatego też w zastosowanej rodzinie kodów umieszczono wszystkie kody Rice'a, a po nich pewną liczbę kodów Golomba, które nie są jednocześnie kodami Rice'a. Liczba kodów Golomba w rodzinie wystarcza

do kompresji typowych obrazów. Dla obrazów nietypowych, na przykład obrazów zaszumionych, rozkład prawdopodobieństwa kompresowanych symboli wymaga użycia kodu Golomba o wysokim parametrze m, którego nie ma w rodzinie. W takim przypadku model danych wybiera odpowiedni kod Rice'a, co powoduje nieznaczne pogorszenie współczynnika kompresji w porównaniu do współczynnika możliwego do uzyskania przy zastosowaniu pełnej rodziny kodów Golomba. Kod zawierający bezpośrednią binarną reprezentację symboli (m=128) jest przesunięty na początek rodziny. Dzięki temu kompresor nie będzie powodował ekspansji danych nie podlegających kompresji, nawet jeżeli użyta do kompresji liczba kodów w rodzinie jest bardzo mała (mniejsza niż 8). Na początku kompresji, zanim model zaadaptuje się do rzeczywistej charakterystyki kompresowanych danych, do kodowania zostanie użyty kod o m=128, gdyż będzie to pierwszy ze wszystkich równie dobrych kodów. Zanim model nauczy się, jak kompresować symbole, będzie tak dobierał parametr kodowania, by nie spowodować ekspansji danych.

Długość kodów ograniczono do 32 bitów. Nakładając ograniczenie na maksymalną długość kodu ograniczamy maksymalną ekspansję danych z 256 bitów na symbol do 32 bitów na symbol, a więc ośmiokrotnie. Co więcej, 4-bajtowe kody można przed rozpoczęciem kompresji wygenerować do tablicy, dzięki czemu każdy kod generujemy tylko raz, niezależnie od tego ile razy będzie wykorzystany.

2.4.2. Model danych

Zastosowany kompresor statystyczny pierwszego rzędu wykorzystuje zmodyfikowany model danych znany z algorytmu FELICS [3]. Dla każdego kontekstu parametr kodowania dobierany jest przez model danych za pomocą prostego i szybkiego algorytmu. Dla każdego kontekstu i dla każdej sensownej wartości parametru (jest ich niewielka liczba) model danych przechowuje licznik sumarycznej liczby bitów, jaką dałoby zastosowanie tego parametru. Do kodowania wybierany jest parametr dający najkrótszy kod. Okresowo, gdy dla danego kontekstu wartość minimalnego licznika przekroczy określony próg, wszystkie liczniki dla tego kontekstu są dzielone przez 2. Zabezpieczamy się w ten sposób przed przepełnieniem liczników i powodujemy, że na wybór parametru silniejszy wpływ mają ostatnio kompresowane symbole.

Opisana metoda doboru parametru nie ogranicza się do kodów Golomba czy Rice'a, nadaje się do dowolnej parametrycznej rodziny kodów o ograniczonym rozmiarze. Dla rodziny kodów Rice'a istnieje prostsza metoda wyznaczania parametru kodowania. Została ona zastosowana w algorytmie LOCO-I [16], jak również w opartym na tym algorytmie proponowanym nowym standardzie JPEG-LS [5, 9]. W jej przypadku dla każdego kontekstu wymagane jest przechowywanie tylko dwóch liczników.

2.4.3. Mechanizm adaptacji

Jeżeli obraz podlegający kompresji jest duży, to po skompresowaniu pewnej ilości danych algorytmem adaptacyjnym (rys. 6a) model staje się dokładny i zazwyczaj dość stabilny. W literaturze proponuje się zaprzestanie uaktualniania modelu po przeczytaniu arbitralnie określonej liczby symboli, czyli kompresję pozostałej części obrazu ze stałym modelem danych (rys. 6b). Rozwiązanie to oczywiście przyspiesza działanie kompresora, ale sprawdza się tylko tak długo, jak długo charakterystyka kompresowanych danych nie zmienia się. Zastosowane zostało bardziej eleganckie rozwiązanie. Model jest uaktualniany ze stopniowo coraz mniejszą dokładnością. Cel ten jest osiągany poprzez wykorzystanie do uaktualniania modelu tylko wybranych (a nie wszystkich) symboli ciągu podlegającego kompresji (rys. 6c). Po uwzględnieniu danego symbolu w modelu wybieramy losowo (w pseudolosowy sposób, aby algorytm dekompresji mógł uczynić to samo) liczbę symboli do pominięcia przed następną aktualizacją modelu.

Zastosowano również dynamiczny podział zbiorczych kubełków kontekstów. Stała liczba kubełków kontekstów o stałym rozmiarze stosowana jest m.in. w algorytmie Lossless JPEG [6]. Zastosowane rozwiązanie jest bardziej ogólne. Na początku kompresji model zawiera jeden zbiorowy kubełek obejmujący wszystkie konteksty. Następnie w miarę przybywania danych, a więc i zwiększania się dokładności modelu, większe kubełki dzielą się na mniejsze. To rozwiązanie ma na celu przyspieszenie adaptacji modelu do charakterystyki kompresowanych danych, co powinno być najwyraźniej odczuwalne na małych obrazach. Dodatkowo, jeżeli nałożymy ograniczenia nie pozwalające podzielić się zbiorczym kubełkom na pojedyncze konteksty, zmniejszymy złożoność pamięciową modelu (w porównaniu do modelu algorytmu FELICS).

```
(a) while not EOF
    begin
      pobierz symbol
      zakoduj symbol
      uaktualnij model
    end

(b) while not EOF
    begin
      pobierz symbol
      zakoduj symbol
    end
```

```
(c) while not EOF
    begin
      pobierz symbol
      zakoduj symbol
      if wait=0
        begin
          wyznacz wait
          uaktualnij model
        end
      else
        wait:=wait-1
    end
```

Rys. 6. Kodowanie i modelowanie
Fig. 6. Coding and modelling

2.4.4. Złożoność pamięciowa kompresora statystycznego

Do kompresji wiersza o długości *dim_x* potrzebny jest bufor wyjściowy o długości *dim_x**4+1 oraz obszar pamięci przechowujący model danych.

Rozmiar pamięci potrzebnej na przechowanie modelu danych, przy uwzględnieniu maksymalnej liczby kubełków *k_max* (mniejszej lub równej rozmiarowi alfabetu) wynosi :

256 * 6 + *k_max* * *liczba_kodów* * *rozmiar_licznika* [B],

lub nieco więcej, gdy kompilator będzie umieszczał pola struktur na granicy słowa. Pierwszy składnik sumy — to rozmiar struktury organizującej podział i dostęp do kubełków. Drugi składnik — to tablica liczników. Uzupełniamy kody Rice'a o kilka pierwszych kodów Golomba. Dla alfabetu zawierającego 256 symboli istnieje 8 kodów Rice'a, można spodziewać się, że taka sama liczba dodatkowych kodów Golomba powinna okazać się wystarczająca. W przyjętym rozwiązaniu zdecydowano się na ograniczenie maksymalnej liczby kodów do 32. *Rozmiar_licznika* możemy, dzięki zaproponowanej rodzinie kodów, ograniczyć do 16 bitów. Ponieważ kody nigdy nie będą dłuższe od 32 bitów, to największy z liczników danego kontekstu będzie co najwyżej 32 razy większy od najmniejszego. Jeżeli będziemy dzielili wszystkie liczniki konkretnego kubełka, nim najmniejszy z nich osiągnie wartość 2048 ($2^{16}/32$), to unikniemy przepełnienia liczników. Ponieważ łatwiej, niż sprawdzać następną wartość licznika, jest kontrolować jego aktualną wartość, próg powodujący dzielenie liczników powinien być mniejszy od 2016 (2048 minus maksymalna możliwa długość następnego kodu).

Jeżeli zastosujemy tablicę kodów, to jej rozmiar należy także uwzględnić w oszacowaniu złożoności pamięciowej algorytmu. Każdy kod zapamiętywany jest jako para: (*długość_kodu*, *biarna_reprezentacja_kodu*). Binarna reprezentacja kodu jest przechowywana na 32 bitach. Do przechowania długości kodu wystarczy 5 bitów. Jeżeli tablicę kodów zaimplementujemy jako dwie osobne tablice (kody, długości), to zajmie ona:

256 * 5 * *liczba_kodów* [bajty].

Jeżeli będzie to jedna tablica struktur, to 32-bitowy kompilator prawdopodobnie przydzieli jej: 256 * 8 * *liczba_kodów* [bajty].

3. Implementacja kompresora

Implementacja ogólnego, wieloetapowego kompresora została zrealizowana w języku C. Ponieważ jedna aplikacja realizuje wszystkie algorytmy, to aby nie rezerwować niepotrzebnie pamięci, wszystkie struktury przydzielane są dynamicznie (bufory, słownik, model danych

i tablica kodów dla kompresora statystycznego). Słownik dla algorytmów słownikowych przydzielany był w rozmiarze wynikającym z maksymalnego rozmiaru słownika, przekazanego jako parametr wywołania programu. Słownik po inicjalizacji zawiera frazę pustą, a w przypadku słownika dla algorytmu LZW dodatkowo 256 nieusuwalnych jednoznakowych fraz. Słownik dla algorytmu LZW zawiera także jedną dwuznakową frazę umieszczoną podczas inicjalizacji ze względów implementacyjnych — dzięki niej w słowniku zawsze znajduje się przynajmniej jedna fraza usuwalna. Model danych kompresora statystycznego i tablica kodów dla tego kompresora przydzielana w zrealizowanej implementacji jest zawsze w rozmiarze zdolnym pomieścić maksymalną dozwolona liczbę kodów (32) i kubełków (256). Inicjalizowana jest tylko rzeczywiście wykorzystywana liczba kodów i kubełków.

Implementacja została przetestowana i uruchomiona w kilku środowiskach sprzętowo-programowych:

- GNU C w systemach Linux, DOS oraz Windows95 na 32-bitowych procesorach Intel x86 i zgodnych.
- Microsoft Visual C++ w systemie Windows95.
- Borland C++ w systemie Windows95.
- Kompilator C systemu Solaris na procesorach SPARC i PentiumPro.
- Kompilator C systemu HP-UX na procesorze PA-RISC 8000.

4. Metody badań

Do badań nad doborem algorytmów użyto ogólnodostępnego zestawu obrazów często stosowanego do porównań algorytmów kompresji (zawiera m.in. obrazy: baboon, boats, camera, lena). Zestaw (tabela 3) zawiera obrazy o rozmiarach od 65536 do 414720 pikseli. Zestaw i odnośniki do kilku publikacji go wykorzystujących znajdują się pod adresem ftp://ftp.funet.fi/pub/misc/test-images.

Badania zostały przeprowadzone na komputerze Compaq Proliant 800, pracującym pod nadzorem systemu operacyjnego Unix SVR4.0 Solaris x.86 2.5.1. Z istotnych parametrów tego komputera należy wymienić rozmiar pamięci operacyjnej: 64MB i procesor: pojedynczy PentiumPro 200MHz z 256kB pamięci cache.

Dla zminimalizowania wpływu obciążenia komputera i wydajności jego podsystemu wejścia/wyjścia na otrzymywane rezultaty pomiary wykonywano w następujący sposób:

- Kompresja wykonywana jest kilkakrotnie (11 razy), czas pierwszego przebiegu jest ignorowany, czas wszystkich pozostałych wykonań jest uśredniany.

- Czas trwania kompresji mierzony był za pomocą systemowego polecenia *time* łącznie dla wszystkich (oprócz pierwszego) przebiegów kompresji, a następnie dzielony przez ich liczbę.
- Jako czas trwania danego eksperymentu przyjmowano sumę zwracanych przez polecenie *time* wartości system time i user time.

Prędkość podawana jest w kB/s, współczynnik kompresji w bitach na piksel:

$$\frac{8 * \text{rozmiar danych skompresowanych}}{\text{rozmiar danych wejściowych}}$$

Tabela 3

Zestaw obrazów testowych

1 – obraz
2 – szerokość
3 – wysokość
4 – rozmiar pliku (liczba pikseli)
5 – entropia rzędu zerowego (bitów na piksel)
6 – entropia pierwszego rzędu (bitów na piksel)

1	2	3	4	5	6
airplane	512	512	262144	6.705888	4.257035
baboon	512	512	262144	7.357949	6.147013
boats	720	576	414720	7.088124	4.684382
bridge	256	256	65536	7.668557	5.717757
camera	256	256	65536	7.009716	4.265794
couple	256	256	65536	6.390254	4.168866
goldhill	720	576	414720	7.529989	4.861624
lena	512	512	262144	7.594038	5.163251
peppers	512	512	262144	7.592450	4.933920

5. Dobór algorytmów składowych kompresora

Celem wstępnej selekcji było wyznaczenie co najwyżej kilku konfiguracji kompresora na podstawie powszechnie uznawanych danych testowych i typowych parametrów algorytmów składowych. Przyjęto następujące parametry algorytmów składowych:

- Dekorelacja domyślnym predyktorem algorytmu Lossless JPEG — *Pred7*.
- Kompresja słownikowa ze słownikiem o rozmiarze 4096 fraz (z usuwaniem fraz po wypełnieniu słownika).
- Kompresja statystyczna z modelem danych algorytmu FELICS z wykorzystaniem 8 kodów zmodyfikowanej rodziny Golomba i bez wprowadzania pozostałych z proponowanych modyfikacji — 256 pojedynczych kontekstów, z uaktualnianiem modelu po kompresji każdego symbolu.

Porównano współczynniki kompresji i prędkości kompresji następujących konfiguracji kompresora (tabele 4 i 5):

- *rle* – dekorelacja i kompresja RLE.
- *rlestat* – dekorelacja, kompresja RLE i kompresja statystyczna.
- *stat* – dekorelacja i kompresja statystyczna.
- *lzw* – dekorelacja i kompresja LZW.
- *lz78* – dekorelacja i kompresja LZ78.
- *lz78+stat* – dekorelacja, kompresja LZ78 i kompresja statystyczna, wynik szacowany, przy założeniu że prędkość i współczynnik kompresji symboli będą takie same jak dla kompresji statystycznej całego obrazu. W rzeczywistości wynik byłby gorszy, gdyż kompresor statystyczny kompresuje słabiej i wolniej mniejsze obrazy.
- *lzwfrz* – dekorelacja i kompresja LZW z zamrażaniem słownika po jego wypełnieniu.

Tabela 4

Współczynnik kompresji

obraz	rozmiar	*rle*	*rlestat*	*stat*	*lzw*	*lz78*	*lz78+stat*	*lzwfrz*
airplane	262144	7.949	4.830	4.232	4.751	5.430	4.406	5.030
baboon	262144	8.031	6.691	6.172	7.474	7.583	6.877	8.016
boats	414720	8.018	5.012	4.438	5.023	5.684	4.671	5.827
bridge	65536	8.046	6.513	5.976	7.252	7.457	6.693	7.548
camera	65536	7.944	5.362	4.758	5.440	5.975	4.999	5.796
couple	65536	7.635	4.830	4.287	4.862	5.533	4.498	5.181
goldhill	414720	7.989	5.247	4.760	5.494	6.113	5.122	6.443
lena	262144	8.031	5.444	4.894	5.761	6.331	5.347	6.036
peppers	262144	8.031	5.417	4.870	5.617	6.245	5.267	5.755
średnio		**7.964**	**5.483**	**4.932**	**5.742**	**6.261**	**5.320**	**6.181**

Tabela 5

Prędkość [kB/s]

obraz	rozmiar	*rle*	*rlestat*	*stat*	*lzw*	*lz78*	*lz78+stat*	*lzwfrz*
airplane	262144	3745	904	794	971	1140	820	1638
baboon	262144	4369	794	819	708	904	634	1311
boats	414720	4608	798	829	1037	1220	860	1803
bridge	65536	1638	655	655	468	468	369	596
camera	65536	1638	596	655	504	504	409	596
couple	65536	2185	655	655	504	546	443	596
goldhill	414720	5184	813	813	987	1220	836	1803
lena	262144	3277	771	771	846	1092	754	1542
peppers	262144	4369	771	794	874	1092	764	1542
średnio		**3446**	**751**	**754**	**767**	**910**	**654**	**1270**

Na podstawie uzyskanych wyników możemy stwierdzić, że algorytm RLE jest bardzo szybki, ale nie kompresuje danych testowych. Na 9 obrazów 5 uległo ekspansji, średni współczynnik kompresji był zbliżony do 8 (7.964). Wynik ten byłby jeszcze gorszy, gdyby nie mechanizm wyprowadzania nieskompresowanych danych, jeżeli kompresja kończy się niepowodzeniem, ograniczający ekspansję do 2 bajtów na wiersz obrazu. Łączenie nieskutecznego algorytmu RLE z kompresją statystyczną nie poprawia prędkości kompresji i pogarsza znacząco współczynnik kompresji (z 4.932 do 5.483).

Do dalszych badań wybrano kompresor statystyczny. Uzyskał on najwyższy średni współczynnik kompresji. Prędkość działania tego kompresora można przyspieszyć przez zmniejszenie częstości uaktualniania modelu.

Prędkość działania kompresora statystycznego jest zbliżona do prędkości działania algorytmów uznawanych za szybkie: FELICS i Unix compress. Uzyskany średni współczynnik kompresji jest zbliżony do uzyskiwanych przez algorytmy Lossless JPEG i FELICS. Porównanie średnich współczynników kompresji uzyskanych dla zestawu obrazów testowych przez wybrane konfiguracje kompresora oraz przez kilka znanych algorytmów kompresji zamieszczono w tabeli 6.

Tabela 6

Średni współczynnik kompresji zestawu obrazów testowych

algorytm	współczynnik
JPEG-LS	4.576
FELICS	4.903
konfiguracja *stat*	4.932
Lossless JPEG (Huffman)	5.086
PNG (najsilniejsza kompresja)	5.215
konfiguracja *lzw*	5.742
dekorelacja *Pred7* i Unix compress	5.802
Unix compress	7.055

Szybsze od kompresji statystycznej są warianty kompresora słownikowego. Analizując wyniki uzyskane przez te kompresory, należy pamiętać, że inicjalizacja słownika jest czasochłonna. Dla większych obrazów należy spodziewać się wzrostu szybkości przetwarzania. Pomimo lepszej prędkości średniej, dla najmniejszych obrazów (o rozmiarze 65536 bajtów) kompresor słownikowy jest wolniejszy od statystycznego.

Spośród konfiguracji wykorzystujących kodowanie słownikowe do dalszych badań wybrano algorytm LZW. Po kompresji tym algorytmem nie da się wprost dokonać dalszej kompresji słownikowej. Algorytm ten osiągnął średni współczynnik kompresji gorszy od algo-

rytmu statystycznego o 0.81 bita na piksel (średni współczynnik kompresji wyniósł 5.742). Średnia prędkość przetwarzania jest nieznacznie większa od prędkości algorytmu statystycznego, ale rośnie znacznie szybciej wraz ze wzrostem rozmiaru przetwarzanego obrazu. Prędkość działania tego kompresora można zwiększyć przez zmniejszenie częstości uaktualniania słownika. Wyniki wariantu z zamrażaniem słownika po jego wypełnieniu (*lzwfrz*) pokazują, że możemy się spodziewać znacznego przyspieszenia. Przyspieszenie można także uzyskać przez zmniejszenie wielkości słownika.

Odrzucono obie konfiguracje wykorzystujące algorytm LZ78. Stosowanie tego algorytmu jako wstępnego etapu przed kompresją statystyczną, w porównaniu do kompresji statystycznej bez wstępnego przetwarzania LZ78, powoduje, że zarówno współczynnik kompresji, jak i prędkość przetwarzania pogarszają się. Algorytm LZ78 zastosowany do kompresji residuum jest wyraźnie szybszy od algorytmu LZW. Przyczyną tego jest po części fakt, że część symboli obrazu jest przez algorytm LZ78 przepisywana na wyjście bez kompresji (jeden symbol na znalezioną frazę). Ta cecha jednocześnie wpływa negatywnie na uzyskiwane przez LZ78 współczynniki kompresji. Średni współczynnik kompresji uzyskany przez algorytm LZ78 (6.261) jest gorszy od średniego wyniku kompresora statystycznego o 1.329 bita na piksel. Pogorszenie średniego współczynnika kompresji jest o ponad połowę większe niż w przypadku algorytmu LZW. Dodatkowym argumentem przemawiającym za odrzuceniem algorytmu LZ78 jest to, iż uzyskał on niższą średnią prędkość i gorszy średni współczynnik kompresji od wersji algorytmu LZW z zamrażaniem słownika po wypełnieniu.

6. Uwagi końcowe i wnioski

Do dalszych badań wybrano dwa algorytmy: kompresja LZW residuum i kompresja statystyczna residuum. Algorytmy te różnią się znacznie. Kompresor słownikowy (w zrealizowanej odmianie) ma większe wymagania pamięciowe od użytego kompresora statycznego. W przypadku danych kompresujących się bardzo dobrze, kompresor słownikowy jest w stanie uzyskać współczynnik kompresji zbliżony do zera, podczas gdy prefiksowy koder statystyczny nie osiągnie kompresji lepszej niż 1 bit na symbol. Istotną cechą kodowania LZW jest asymetryczność tej metody. Dekompresja jest szybsza, ponieważ unikamy wyszukiwania w słowniku. Struktura słownika dekompresora może być znacznie prostsza i mniejsza od struktury słownika wymaganej przez kompresor.

Następne fazy badań stanowiły: dobór parametrów dla wyselekcjonowanych algorytmów, wyznaczenie efektów zastosowania proponowanych zmian oraz porównanie z innymi, już istniejącymi algorytmami. Uzyskane wyniki opisane są w pracach [14, 15].

Podsumowując, należy stwierdzić, iż koncepcja wprowadzenia dodatkowego etapu kompresji poprzedzającego kompresję statystyczną nie sprawdziła się. Sprawdziły się natomiast pozostałe modyfikacje wprowadzone do wybranych algorytmów kompresji [14, 15]. Algorytmy te w swoich podstawowych odmianach uzyskiwały prędkości zbliżone do znanych szybkich adaptacyjnych algorytmów kompresji (FELICS, Unix compress) i przeciętne współczynniki kompresji. Poprzez zastosowanie wspomnianych modyfikacji uzyskano wzrosty prędkości działania dwu– lub kilkakrotne bez, lub przy nieznacznym pogorszeniu współczynników kompresji. Proponowane modyfikacje pozwoliły również na poprawę wyników algorytmów dla obrazów zaszumionych (kompresujących się słabo) lub nie podlegających kompresji.

LITERATURA

1. Allred, L.G.; Kelly, G.E.: A lossless image compression technique for infrared thermal images. Proceedings of the SPIE - The International Society for Optical Engineering, 1992 USA, Vol: 1702 s. 230-7.
2. Drozdek, A.; Simon, D.L.: Struktury danych w języku C. WNT, Warszawa 1996.
3. Howard, P.G.; Vitter, J.S.: Fast and efficient lossless image compression. Proceedings DCC '93. Data Compression Conference, IEEE Comput. Soc. Press Los Alamitos, CA, USA 1992, s. 351-60.
4. Howard, P.G.; Vitter, J.S.: Fast progressive lossless image compression. Proceedings of the SPIE - The International Society for Optical Engineering, 1994 USA, Vol: 2186 s. 98-109.
5. ISO/IEC JTC 1/SC 29/WG 1 FCD 14495: Lossless and near-lossless compression of continuous-tone still images (JPEG-LS). ISO Working Document ISO/IEC JTC1/SC29/WG1 N522, 1997.
6. Langdon, G.; Gulati, A.; Seiler, E.: On the JPEG model for lossless image compression. Proceedings DCC '92. Data Compression Conference, IEEE Comput. Soc. Press Los Alamitos, CA, USA 1992, s. 172-80.
7. Melnychuck, P.W.; Rabbani, M.: Survey of lossless image coding techniques. Proceedings of the SPIE - The International Society for Optical Engineering, 1989 USA, Vol: 1075 s. 92-100.
8. Memon, N.D.; Sayood, K.: Lossless image compression: A comparative study. Proceedings of the SPIE - The International Society for Optical Engineering, 1995 USA, Vol: 2418 s. 8-20.

9. Memon, N.D.; Wu, X.: Recent developments in context-based predictive techniques for lossless image compression. The Computer Journal, Great Britain 1997, Vol: 40 Iss: 2/3 s 127-36.
10. Migas, A.: Kompresja danych. Charakterystyka metody LZW. Zeszyty Naukowe Politechniki Śląskiej, Gliwice 1995, seria Informatyka z. 29, s. 95-112.
11. Reingold, E.M.; Nievergelt, J.; Deo, N.: Algorytmy kombinatoryczne. PWN, Warszawa 1985.
12. Skarbek, W.: Metody reprezentacji obrazów cyfrowych. Akademicka Oficyna Wydawnicza PLJ, Warszawa 1993.
13. Skarbek, W.: Multimedia algorytmy i standardy kompresji. Akademicka Oficyna Wydawnicza PLJ, Warszawa 1998.
14. Starosolski, R.: Fast, robust and adaptive lossless image compression. Machine Graphics and Vision, Warszawa 1999, Vol. 8, No. 1 s. 95-116.
15. Starosolski, R.: Fast and adaptive lossless grayscale image compression using the LZW algorithm. Zaakceptowany do publikacji w Archiwum Informatyki Teoretycznej i Stosowanej nr 2/99, Gliwice.
16. Weinberger, M.J.; Seroussi, G.; Sapiro, G.: LOCO-I: A low complexity, context-based, lossless image compression algorithm. Proceedings DCC '96. Data Compression Conference, IEEE Comput. Soc. Press Los Alamitos, CA, USA 1996.
17. Witten, I.H.; Moffat, A.; Bell, T.C.: Managing Gigabytes. Van Nostrand Reinhold, USA 1994.
18. Yovanof, G.S.; Sullivan, J.R.: Lossless predictive coding of colour graphics. Proceedings of the SPIE - The International Society for Optical Engineering, 1992 USA, Vol: 1657, s. 68-82.

Recenzent: Prof. dr hab. Władysław Skarbek

Wpłynęło do Redakcji 19 listopada 1998 r.

Abstract

The main objective of the research reported was to improve the speed of the image compression algorithm at the cost of at most a little compression ratio deterioration — in the practical image processing applications little changes of compression ratio are negligible.

We start with the traditional decorrelation – statistical compression scheme. To improve the compression speed we add an additional stage preceding the statistical compression: a fast RLE or dictionary compressor whose task is to quickly reduce the amount of data to be processed by the, considered to be slow, statistical compressor.

We also alter the statistical compression process: instead of updating the data model each time a symbol is coded we do it after coding of some (pseudo-randomly selected) symbols only (fig. 6).

The algorithms we examine are: the statistical coder based on the one first used in the FELICS algorithm, RLE, LZ78 and LZW.

Real-life applications require algorithms to have reasonable and strictly bounded memory requirements. The algorithm's performance on the worst case data is also important. The data expansion and the speed decrease in such a case should be insignificant and also bounded. We introduce many improvements to the existing data compression algorithms to meet these requirements.

In this paper after the short introduction to the lossless image compression, we describe the algorithms used and report the results of the selection of the algorithms.

Generally we find that preceding the statistical compression by the examined faster algorithms does not improve the overall compression speed. We select two schemes that may be further improved by fully utilizing the introduced modifications.

Research on the selected algorithms, effects of introduced modifications and comparison to other techniques are described in separate papers [15, 16].