Kratek pregled metod predobdelave zaporedij pred postopki stiskanja podatkov

Borut Žalik[1], Štefan Kohek[1], Simon Kolmanič[1], Ivana Kolingerová[2], David Podgorelec[1], Aljaž Jeromel[1]

[1]*Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko, Maribor, Slovenia*
[2]*University of West Bohemia, Faculty of Applied Sciences, Pilsen, Czech Republic*
E-pošta: borut.zalik@um.si

A short survey on string preprocessing methods before compression

This paper considers three string transformation techniques, which may lead to reduction of the information entropy and better compression. The known lossless compression approaches are mentioned at first. Move-To-Front, Inversion Frequencies, and Burrows-Wheeler Transform are explained after that. The first two transformations may reduce information entropy, while the last rearranges symbols in the way that the same symbols tend to be close together. Effects of these transformations have been demonstrated on gray-scale raster images.

1 Uvod

Imejmo zaporedje $X = \langle x_i \rangle$, $0 \leq i < n$, z abecedo $\Sigma_X = \{x_j\}$, $0 \leq j < m$, kjer je m moč abecede. Naj bo $f : X \to Y$ bijektivna preslikava, ki vsakemu $x_i \in \Sigma_X$ priredi zaporedje binarnih kod/bitov $y_i = \langle b_{i,k} \rangle$, $0 \leq k < r_i$, $\Sigma_Y = \{0, 1\}$, kjer je r_i dolžina kode y_i. Te kode tvorijo izhodno zaporedje $Y = \langle \langle y_i \rangle \rangle$. Če je skupno število bitov zaporedja Y manjše od števila bitov, ki tvorijo X (tudi simboli x_i so predstavljeni z binarnimi kodami), govorimo o stiskanju. Stiskanje podatkov spada med najstarejše izzive v računalništvu, začenši z delom Shannona [1], ki je dokazal, da je smiselno simbolom z večjo verjetnostjo dodeliti krajše binarne kode. Shannon je prav tako izpeljal spodnjo mejo povprečnega števila bitov za $x_i \in \Sigma_X$, kar imenujemo informacijska (oz. Shannonova) entropija H. Pri dodeljevanju kod moramo biti pazljivi, če želimo, da bo izpolnjen pogoj bijektivnosti funkcije f, to je, da bo možno enolično rekonstruirati X iz Y. Ta postopek imenujemo razširjanje. Kodam, ki to zagotavljajo, pravimo predponske/prefiksne kode. Fano [2] in Huffman [3] sta na podlagi statistične analize pogostosti pojavljanja znakov v X predlagala algoritma, ki sestavita takšne kode, s katerimi se zelo približamo informacijski entropiji. Huffman je uspel dokazati, da njegov algoritem sestavi optimalne kode, če je porazdelitev verjetnosti simbolov v X geometrijska. Tudi v splošnem velja, da je tako imenovano Huffmanovo kodiranje uspešnejše od Fanojevega, zato se pogosteje uporablja v praksi [4].

Naslednji, zelo pomemben preboj pri stiskanju podatkov, je bila realizacija aritmetičnega kodiranja v digitalnem računalniku, obremenjenem s končno aritmetiko [5]. Tudi aritmetično kodiranje je statistično in deluje tako, da priredi vsakemu simbolu $x_i \in \Sigma_X$ interval na številski premici, katerega širina je odvisna od verjetnosti simbola – verjetnejši simboli zasedejo širši interval. V postopku kodiranja se interval postopoma oži in sicer manj, če kodiramo simbol z večjo verjetnostjo. Vsako število iz zadnjega intervala ustreza vhodnemu zaporedju. Aritmetično kodiranje se bolje približa Shannonovi informacijski entropiji kot Huffmanovo, če porazdelitev verjetnosti simbolov ne sledi geometrijski porazdelitvi.

Duda [6] je razvil kodirnik ANS (angl. Asymmetric Numeral Systems), ki tudi temelji na statistični analizi. Kodirnik zakodira vhodno zaporedje v celo število, imenovano stanje, ki ga dobimo s celoštevilskim množenjem, deljenjem in seštevanjem. Če zanemarimo ostanke pri deljenju, bomo s kodiranjem vsakega simbola x_i stanje povečali sorazmerno s pogostostjo pojavitve tega simbola. Če sta v vhodnem zaporedju samo dva simbola, kodiranje realiziramo s sistemom enačb, sicer pa si moramo pomagati še s tabeliranjem ostankov. Po stopnji stiskanja je stiskanje ANS primerljivo z aritmetičnim, po hitrosti pa s Huffmanovim kodiranjem [6].

Odmik od dodelitve predponske kode posameznim simbolom je stiskanje s slovarjem, ki sta ga uvedla Ziv in Lempel [7] z algoritmom LZ77. Predlagala sta tvorjenje dinamičnega slovarja, ki sestoji iz delov že kodiranega sporočila, tako imenovanih fraz ϕ_k. Frazam sta priredila enolične žetone ω_k; bijektivna preslikava je tokrat $f : \phi_k \to \omega_k$. Stisnjeno zaporedje tako sestoji iz zaporedja kod žetonov, $Y = \langle \omega_0, \omega_1, \cdots, \omega_{q-1} \rangle$. Različne možnosti tvorjenja žetonov in fraz ter nadzor slovarja so omogočili razvoj številnih postopkov stiskanja s slovarjem [8, 9, 10, 11, 12].

Interpolativno kodiranje, ki sta ga predstavila Moffat in Stuiver [13], temelji na povsem drugačni ideji, saj ne poteka zaporedoma znak za znakom od začetka do konca zaporedja, ampak po vnaprej določenem vrstnem redu (najpogosteje z rekurzivnim deljenjem X na polovici). Koda simbola, ki ga kodiramo, je zato bolj odvisna od njegovega položaja v X kot od njegove vrednosti. Interpolativno kodiranje dosega primerljive rezultate z aritmetičnim kodiranjem [14].

Vsi omenjeni pristopi so našli svoje mesto v vsakodnevnih aplikacijah in žal tudi dosegli meje svojih zmožnosti. Kako kljub temu izboljšati učinkovitost stiskanja?

Ponujata se dve možnosti:

- uvedba kontekstnega modeliranja [15, 16] ali
- transformacija vhodnega zaporedja v zaporedje, ki bi morebiti bilo bolj stisljivo. Takšne transformacije si bomo ogledali v nadaljevanju tega prispevka.

2 Metode transformacije zaporedij

2.1 Transformacija premik naprej

Transformacijo premik naprej (angl. Move-To-Front – MTF) je uvedel Ryabko [17] in je ena od samoorganizirajočih se podatkovnih struktur. MTF preslika $X = \langle x_i \rangle$, $x_i \in \Sigma_X$, v $Y = \langle y_i \rangle$, kjer je $y_i \in \Sigma_Y = \{0, 1, 2 \cdots |\Sigma_X| - 1\}$, pri čemer ni nujno, da so vsi elementi iz Σ_Y tudi prisotni v Y. MTF torej spremeni domeno, dolžini zaporedij $|X|$ in $|Y|$ pa sta enaki. Za njeno realizacijo potrebujemo seznam L z naključnim dostopom. Transformacija poteka v naslednjih korakih:

- napolni seznam L z vsemi $x_i \in \Sigma_X$;
- za vsak $x_i \in X$ poišči njegov položaj p v L;
- p pošlji v Y;
- premakni znake x_j, $0 \leq j < p$, v L za eno mesto;
- x_i postavi na prvo mesto v L.

Oglejmo si primer. Naj bo $X = \langle$rrrrereregarega$\rangle$ kjer je $\Sigma_X = \{$a, e, g, r$\}$. Postopek transformacije vidimo v tabeli 1. Če bi izračunali informacijsko entropijo zaporedja $Y = \langle 300021111333333 \rangle$, bi dobili $H(Y) = 1{,}75$, zaporedje X pa ima informacijsko entropijo $H(X) = 1{,}80$. Transformacijo MTF je pogosto smiselno uporabiti ponovno [18]. Če Y, $\Sigma_Y = \{0, 1, 2, 3\}$, transformiramo še enkrat, bi dobili $Y_1 = \langle 310033000300000 \rangle$ z občutno manjšo Shannonovo entropijo $H(Y_1) = 1{,}16$. Kot vidimo v tem primeru, se element 2 ne pojavi v Y_1, število simbolov v Y_1 se je zmanjšalo, kar je za stiskanje odlična novica. MTF zmanjša informacijsko entropijo, ko je v podatkih veliko lokalnih korelacij, saj zaporedja enakih znakov transformira v indekse 0, zaporedne enake pare znakov v indekse 1, zaporedja enakih trojic v indekse 2 itn. V primeru takšnih značilnosti v X je zaporedje po MTF bolj stisljivo.

Tabela 1: Transformacija MTF

x_i	L				y_i
	0	1	2	3	
init.	a	e	g	r	
r	r	a	e	g	3
r	r	a	e	g	0
r	r	a	e	g	0
r	r	a	e	g	0
e	e	r	a	g	2
r	r	e	a	g	1
e	e	r	a	g	1
r	r	e	a	g	1
e	e	r	a	g	1
g	g	e	r	a	3
a	a	g	e	r	3
r	r	a	g	e	3
e	e	r	a	g	3
g	g	e	r	a	3
a	a	g	e	r	3

Inverzna transformacija MTF je preprosta. Tokrat je vhodno zaporedje zaporedje indeksov $Y = \langle y_i \rangle$, $y_i \in \Sigma_Y, \Sigma_Y = \{0, 1, 2, \cdots, |\Sigma_Y| - 1\}$, poznati moramo še abecedo Σ_X transformiranega zaporedja. Seznam L inicializiramo na enak način, kot smo to storili pri postopku transformacije. Zatem po vrsti jemljemo indekse y_i iz Y, v seznamu L na položaju y_i preberemo znak x_i in ga pošljemo v rekonstruirano zaporedje. Nato na enak način kot pri transformiranju ažuriramo L.

2.2 Transformacija inverzne frekvence

Avtorja transformacije inverzne frekvence (angl. inversion frequencies, IF) sta Arnavut in Magliveras [19, 20]. IF sprejme na vhodu Σ_X ter $X = \langle x_i \rangle, x_i \in \Sigma_X$. Tudi IF pretvori X v $Y = \langle y_i \rangle, y_i \in \Sigma_Y, \Sigma_Y = \{0, 1, 2, \cdots, |X| - 1\}$. Podobno kot MTF, opravi transformacijo iz domene znakov v domeno naravnih števil. A zaloga vrednosti Σ_Y je tokrat omejena z $|X|$ in ne, tako kot pri MTF, z $|\Sigma_X|$. Tudi pri IF ni nujno, da so vsi elementi iz Σ_Y tudi v Y.

IF za vsak x_i najprej poišče njegovo prvo pojavitev v X, za vse naslednje pojavitve pa določi odmik od neposredno predhodne pojavitve. Pri določanju odmika preskoči do sedaj že uporabljene znake $x_j \in \Sigma_X$. Delne rezultate shranimo v pomožna zaporedja A_k, $k = 0, 1, 2, \cdots, |\Sigma_X| - 1$, nato pa jih zlepimo v Y.

Poglejmo si delovanje transformacije IF še s primerom $X = \langle$regaregakvak$\rangle$, kjer je $\Sigma_X = \{$a, e, g, k, r, v$\}$. Delni rezultati za vsak $x_i \in \Sigma_X$ so:

- delni rezultat za 'a': $A_\mathrm{a} = \langle 3, 3, 2 \rangle$;
- delni rezultat za 'e': $A_\mathrm{e} = \langle 1, 2 \rangle$;
- delni rezultat za 'g': $A_\mathrm{g} = \langle 2, 1 \rangle$;
- delni rezultat za 'k': $A_\mathrm{k} = \langle 8, 1 \rangle$;
- delni rezultat za 'r': $A_\mathrm{r} = \langle 0, 0 \rangle$;
- delni rezultat za 'v': $A_\mathrm{v} = \langle 9 \rangle$;

Kratko pojasnilo bo pomagalo pri razumevanju. Prvi 'a' se nahaja na položaju 3 v X, do naslednjega 'a' je treba preskočiti tri druge znake, do zadnjega 'a' pa dva znaka. Prvi 'e' se v X nahaja na položaju 1, do naslednjega 'e' so sicer trije znaki, a ker smo 'a' že uporabili, vpišemo v A_e 2. Lepljenje zaporedij A_a do A_v nam da $Y = \langle 3, 3, 2, 1, 2, 2, 1, 8, 1, 0, 0, 9 \rangle$. Tudi IF lahko zmanjša informacijsko entropijo. V našem primeru je $H(X) = 2{,}52$, medtem ko je $H(Y) = 2{,}46$. Opazimo še nekaj. Vrednosti v pomožnih zaporedjih (razen prve vrednosti, ki pove absolutni položaj znaka) bodo postajale manjše in manjše; pri zadnji črki abecede pa bodo vse zagotovo enake 0. Za inverzno transformacijo IF potrebujemo Σ_X in informacijo o dolžini pomožnih zaporedij, to je zaporedje frekvenc simbolov F. V našem

primeru bi bilo zaporedje $F = \langle 3, 2, 2, 2, 2, 1\rangle$. Zapisa zaporedja F pa se lahko izognemo z uvedbo stražarja. Naj bo stražar '!'. S stražarjem povemo dekodirniku, da naj začne v rekonstruirano zaporedje vpisovati naslednji znak iz Σ_X. Pri uporabi stražarjev IF potrebuje le Σ_X, $|X|$ in Y, pri čemer pa zadnjega pomožnega polja ne zapišemo. Namreč, za preostala nezasedena mesta po vpisu vseh $|\Sigma_X|-1$ znakov vemo, da jih bo zasedel zadnji znak iz Σ_X. V našem primeru bi bilo zaporedje Y, zapisano z uporabo stražarja, naslednje: $Y = \langle 3, 3, 2, !, 1, 2, !, 2, 1, !, 8, 1, !, 0, 0, !\rangle$. Pri daljših sporočilih tako velja, da je lahko $|Y| < |X|$, če je zadnjih simbolov v abecedi več, kot je uporabljenih stražarjev.

2.3 Transformacija Burrowsa in Wheelerja

Ena izmed možnih idej transformacij zaporedij so tudi njihove permutacije. Permutacija s prav posebnimi lastnostmi je urejeno zaporedje, ki je ugodno za različne nadaljnje obdelave, med drugim tudi za stiskanje podatkov. Burrows in Wheeler sta iznašla postopek, ki sicer X uredi le delno, za povrnitev v izvorno zaporedje pa potrebuje $O(1)$ dodatne informacije [21]. Burrows-Wheelerjeva transformacija (BWT) tako ne spremeni domene, $\Sigma_X = \Sigma_Y$, niti ne zmanjša dolžine izhodnega zaporedja, $|\Sigma_X| = |\Sigma_Y|$, ampak samo premeša $x_i \in X$ tako, da so enaki znaki v transformiranem zaporedju pogosteje drug zraven drugega. BWT ima mnoge pomembne lastnosti (odličen pregled najdemo v [22]).

Naj bo $X = \langle regaregakvak\rangle$. BWT dobimo v treh korakih:

1. Konstrukcija $|X|$ permutacij X s krožnimi premiki v desno (glej tabelo 2, prvi korak).
2. Leksikografsko urejanje zaporedja permutacij (tabela 2, drugi korak).
3. Rezultat Y je še ena permutacija, in sicer tista, ki jo sestavimo iz zadnjih znakov leksikografsko urejenih permutacij (podčrtani znaki v tabeli 2).

Tabela 2: Koraki BWT

vrstica	prvi korak	drugi korak
0	regaregakvak	akregaregak<u>v</u>
1	egaregakvakr	akvakregare<u>g</u>
2	garegakvakre	aregakvakre<u>g</u>
3	aregakvakreg	egakvakrega<u>r</u>
4	regakvakrega	egaregakvak<u>r</u>
5	egakvakregar	gakvakregar<u>e</u>
6	gakvakregare	garegakvakr<u>e</u>
7	akvakregareg	kregaregakv<u>a</u>
8	kvakregarega	kvakregarega
9	vakregaregak	regakvakrega
10	akregaregakv	regaregakva<u>k</u> ←
11	kregaregakva	vakregarega<u>k</u>

Transformacija BWT je torej $Y = \langle$vggrreeaaakk$\rangle$. Za rekonstrukcijo potrebujemo še podatek, v kateri vrstici leksikografsko urejene tabele $|X|$ permutacij najdemo X. Kot vidimo, je to v vrstici 10, podatek ($I_{BWT} = 10$) shranimo za postopek rekonstrukcije.

Rekonstrukcija tokrat, vsaj na prvi pogled, ni trivialna. Y sicer vsebuje vse znake izvornega zaporedja, prav tako poznamo njegov prvi znak, saj tja kaže I_{BWT}. Hitro ugotovimo, da lahko sestavimo tudi urejeno zaporedje prvih znakov z običajnim urejanjem. Tako dobimo prvo vrstico iz drugega stolpca v tabeli 2, označeno s Pv tabeli 3. Postopek rekonstrukcije postane sedaj razumljiv.

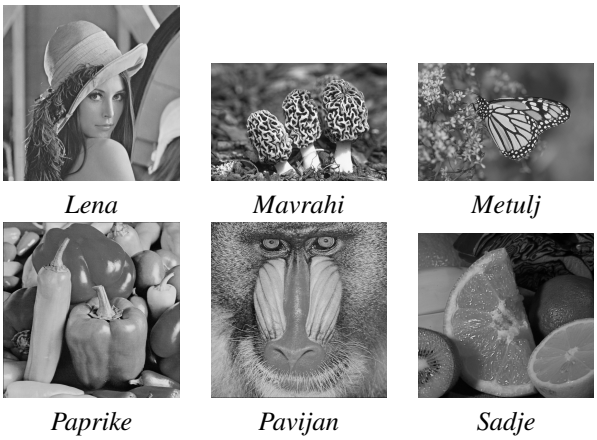
Tabela 3: Prikaz inverzne BWT

i	0	1	2	3	4	5	6	7	8	9	10 ↓	11
P	a	a	a	e	e	g	g	k	k	r	r	v
BWT	v	g	g	r	r	e	e	a	a	a	k	k

Na prvo črko rekonstuiranega zaporedja kaže kazalec I_{BWT}. Dobimo $X = \langle$r$\rangle$. Gre za drugi 'r' v P, zato poiščemo drugi 'r' v BWT. Najdemo ga na položaju 4. $P_4 =$ e, kar je drugi rekonstruiran znak, torej $X = \langle$re$\rangle$. Drugi znak 'e' v BWT se nahaja na položaju 6; $P_6 =$ g; $X = \langle$reg$\rangle$. Na ta način nadaljujemo do konca. Tipično sledi za BWT MTF [23], ki zaporedja enakih znakov transformira v zaporedja ničel. Druge možnosti transformacij za BWT pa najdemo v [24]. Slabost BWT je časovna zahtevnost $O(|X|^2 \log |X|)$. Danes vemo, da lahko BWT dobimo v linearnem času iz priponskega polja [22].

3 Rezultati

Učinek transformacij MTF in IF smo preizkusili na naboru standardnih 8-bitnih sivisnkih rastrskih slik, ki jih kaže slika 1. Slike so v različnih ločljivostih (*Lena*, *Paprike* in *Pavijan* v 512×512, *Mavrahi* v 481×321, *Metulj* v 768×512 in *Sadje* v 512×480. Vrednosti pikslov smo pretvorili v vhodno zaporedje X z skanirnim prebiranjem. Število elementov v X je tako določeno s številom pikslov slike. Tabela 4 kaže vpliv transformacij



Slika 1: Testne slike.

MTF in IF na informacijsko entropijo H. Kot vidimo, sta obe transformaciji zmanjšali informacijsko entropijo, njuni rezultati pa so zelo primerljivi.

Tabela 5 kaže uspešnost transformacij MTF in IF po tem, ko smo najprej opravili transformacijo BWT. Vidimo, da je vpliv BWT občuten. Samo v primeru slike

Tabela 4: Vpliv MTF in IF na informacijsko entropijo

Slika	$H(X)$	$H(\text{MTF}(X))$	$H(\text{IF}(X))$
Lena	7,348	**6,629**	6,720
Mavrahi	7,585	7,230	**7,095**
Metulj	7,182	6,082	**5,935**
Paprike	7,594	**6,840**	6,967
Pavijan	7,358	**7,308**	7.415
Sadje	7,366	6,294	**7.218**
Povprečje	7,406	6,731	6.891

Pavijan je uporaba BWT, ki ji sledi MTF, bila uspešnejša od kombinacije BWT in IF.

Tabela 5: Vpliv MTF in IF na informacijsko entropijo po BWT

Slika	$H(X)$	$H(\text{MTF}(X))$	$H(\text{BWTIF}(X))$
Lena	7,348	5,240	**5,118**
Mavrahi	7,585	5,961	**5,788**
Metulj	7,182	4,770	**4,632**
Paprike	7,594	5,459	**5,373**
Pavijan	7,358	**6,656**	6,680
Sadje	7,366	4,805	**4,657**
Povprečje	7,406	5,481	5,375

4 Zaključek

V članku predstavimo tri tehnike transformacije zaporedij. Na primeru šestih sivinskih rastrskih slik prikažemo njihov učinek na zmanjšanje informacijske entropije. Pokažemo, da je uporaba transformacije BWT pred MTF in IF smiselna. Iskanje novih transformacij, ki bi vključevale tudi kontekstno znanje o podatkih, predstavlja zanimiv izziv za bodoče raziskave.

Zahvala
Projekt (Paradigma stiskanja podatkov z odstranjevanjem obnovljivih informacij, št. J2-4458) sta financirali Javna agencija za znanstvenoraziskovalno in inovacijsko dejavnost Republike Slovenije ter Czech Science Foundation (project No. 23-04622L) iz državnega proračuna.

Literatura

[1] C. E. Shannon, A mathematical theory of communication, Bell System Technical Journal (27), 1948, 379–423.

[2] R. M. Fano, The transmission of information, Tehniško poročilo št. 65, Research Laboratory of Electronics at MIT, 1949.

[3] D. Huffman, A method for the construction of minimum–redundancy codes, Proceedings of the IRE 40(9), 1952, 1098–1101.

[4] B. Furht, A survey of multimedia compression techniques and standards. Part I: JPEG standard, Real Time Imaging 1(1), 1995, 49–67.

[5] E. Bodden, M. Clasen, J. Kneis, Arithmetic coding revealed: A guided tour from theory to praxis, Tehniško poročilo št. 2007-5, McGill University, Sable Research Group, 2007.

[6] J. Duda, Asymmetric numeral systems: entropy coding combining speed of Huffman coding with compression rate of arithmetic coding, arXiv:1311.2540, 2013.

[7] J. Ziv, A. Lempel, A universal algorithm for sequential data compression, IEEE Transactions on Information Theory 23(3), 1977, 337–343.

[8] D. Salomon, G. Motta, Handbook of data compression, Springer (5. izdaja), 2010.

[9] K. Sayood, Introduction to data compression, Morgan Kaufmann (4. izdaja), 2012.

[10] M. Nelson, J.-L. Gailly, The data compression book, M&T Books (2. izdaja), 1996.

[11] I. M. Pu, Fundamental data compression, Butterworth–Heinemann, 2006.

[12] B. Žalik, Aplikacije računalniških algoritmov, Univerzitetna založba, Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko, 2023.

[13] A. Moffat, A., L. Stuiver, Binary interpolative coding for effective index compression, Information Retrieval 3(1), 2000, 25–47.

[14] B. Žalik, D. Strnad, Š. Kohek, I. Kolingerová, A. Nerat, N. Lukač, B. Lipuš, M. Žalik, D. Podgorelec, FLoCIC: A Few Lines of Code for Raster Image Compression, Entropy 25(3), 2023, 533.

[15] D. Marpe, H. Schwarz, T. Wiegand, Context-based, adaptive binary arithmetic coding in the H.264/AVC video compression standard, IEEE transactions on circuits and systems for video technology, 13(7), 2003, 620–636.

[16] B. Žalik, D. Mongus, N. Lukač, K. Rizman Žalik, Can Burrows-Wheeler Transform be replaced in chain code compression?, Information sciences 525, 2020, 109–118.

[17] B. Y. Ryabko, Data compression by means of a 'book stack', Problems in Information Transmission 16(4), 1980, 265-269.

[18] B. Žalik, N. Lukač, Chain code lossless compression using Move-To-Front transform and adaptive Run-Length Encoding, Signal Processing Image Communication 29(1), 2014, 96–106.

[19] Z. Arnavut, S. S. Magliveras, Block sorting and compression, V: J. A. Storer, M. Cohn (ur.) Proceedings of the IEEE Data Compression Conference, 1997, 181–190.

[20] Z. Arnavut, Move-To-Front and inversion coding, V: J. A. Storer, M. Cohn (ur.) Proceedings of the IEEE Data Compression Conference, 2000, 193–202.

[21] M. Burrows, D. J. Wheeler, A block-sorting lossless data compression algorithm, Tehniško poročilo št. 124, HP System Research Center, 1994.

[22] A. Adjeroh, A. Mukherjee, T. Bell, The Burrows-Wheeler transform: Data compression, suffix arrays, and pattern matching, Springer Science + Business Media, 2008.

[23] B. Žalik, D. Mongus, K. Rizman Žalik, N. Lukač, Chain code compression using string transformation techniques, Digital signal processing 53, 2016, 1-10.

[24] J. Abel, Post BWT stages of the Burrows-Wheeler compression algorithm, Software: Practice and Experience 40(9), 2010, 751–777.