Prilagoditev transformacije premik s prepletanjem za nize z veliko abecedo

Borut Žalik[1], Damjan Strnad[1], Ivana Kolingerová[2], Andrej Nerat[1], David Podgorelec[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

An adaptation of a string transform Move-with-Interleaving for large alphabets

The aim and usage of string transforms are considered first. After that, the transformation Move-with-Interleaving is explained briefly. The problem of intensive memory access with large alphabets is exposed. The solution, named Move-with-Interleaving-for-Large-Alphabets, is presented. Only a fraction of elements from the alphabet is maintained in memory in this case, while the positions of others are calculated. Finally, the impressive gains are demonstrated on 16-bit digital audio.

1 Uvod

Niz elementov X, $X = \langle x_i \rangle$, $0 \leq i < n$, pri čemer so elementi x_i iz abecede Σ_X z močjo $|\Sigma_X|$, je zaradi svoje naravne prilagojenosti strukturi računalniškega pomnilnika, tako internega kot tistega na perifernih napravah, najpogostejša struktura pri obdelavi podatkov. Posebna skupina algoritmov nad nizi so algoritmi stiskanja podatkov [1]. Delimo jih v izgubne in brezizgubne. Izgubni najprej transformirajo podatke v, najpogosteje, frekvenčni prostor, dobljene frekvenčne koeficiente kvantizirajo, preostanek pa stisnejo z brezizgubnimi postopki. Slednji so bili razviti pred desetletji (na primer Shannon-Fanojevo [2], Huffmanovo [3] in aritmetično kodiranje [4]) in temeljijo na statistični analizi pogostosti simbolov v X; pogostejši simboli dobijo krajše binarne kode, učinkovitost stiskanja pa je omejena z informacijsko entropijo X [5]. A podatkom, za katere obstajajo še kakšne druge značilnosti, lahko s predobdelavo spremenimo njihove statistične lastnosti in s tem izboljšamo učinkovitost stiskanja. Primeri takšnih postopkov so:

- če X vsebuje ponavljajoče se znake, lahko zaporedja enakih znakov nadomestimo z njihovim številom (tehnika RLE [1]),
- če X vsebuje ponavljajoče se vzorce/fraze, jih lahko nadomestimo z žetoni (stiskanje s slovarjem [6]),
- če lahko X transformiramo v drug niz Y, ki ima manjšo informacijsko entropijo. V tem članku se bomo ukvarjali z eno izmed takšnih transformacij.

Članek sestoji iz pet poglavij. V poglavju 2 bomo predstavili transformacijo premik s prepletanjem. Izkaže se, da transformacija pri velikih abecedah (na primer, abecedah, ki imajo več kot 2^8 elementov) deluje počasi. V poglavju 3 bomo zato predlagali spremembo te transformacije, prilagojeno za velike abecede. V poglavju 4 bomo demonstrirali učinkovitost rešitve z merjenjem časa CPE pri transformaciji 16-bitnega digitalnega zvoka. Članek zaključimo s poglavjem 5.

2 Transformacija premik s prepletanjem

Transformacije nizov lahko razdelimo v dve skupini:

- Prva skupina niz $X = \langle x_i \rangle$, $x_i \in \Sigma_X$, pretvori v niz $Y = \langle y_i \rangle$, $y_i \in \Sigma_Y$, kjer velja $\Sigma_X = \Sigma_Y$. Te transformacije tvorijo torej drugačno permutacijo X z lastnostmi, primernejšimi za reševanje specifičnih nalog. Ena izmed takšnih permutacij je tudi Burrows-Wheelerjeva transformacija (BWT) [7]. BWT ima veliko področij uporabe, ena izmed njih je tudi stiskanje podatkov [8], saj pogosto tvori daljša zaporedja enakih simbolov. Sama BWT sicer ne zmanjša informacijske entropije, prav tako ostane $|X| = |Y|$, lahko pa v kombinaciji s transformacijami iz druge skupine omogoči zmanjšanje informacijske entropije [9].
- Druga skupina transformacij pretvori X z elementi iz abecede Σ_X v niz $Y = \langle y_i \rangle$, kjer so $y_i \in \mathbb{N}_0$. Zgornja meja vrednosti elementov y_i je odvisna od vrste transformacij in je lahko bodisi $|\Sigma_X|$ ali celo $|X|$. Najbolj znana predstavnika teh transformacij sta premik naprej (angl. Move-To-Front, MTF) [10] in inverzne frekvence (angl. inversion frequencies, IF) [11]. Primere teh transformacij najdemo v [12].

Transformacija premik s prepletanjem (angl. Move-with-Interleaving, MwI) je bila predlagana pred kratkim [13]. MwI se je izkazala primerljiva glede na zmožnost zmanjševanja informacijske entropije na naboru 32 standardnih testnih sivinskih fotografij, kot sta to zmogli kombinaciji BWT–MTF ali BWT–IF.

Transformacija MwI temelji na transformaciji MTF. MTF za postopek transformacije uporablja seznam L in deluje v naslednjih korakih:

- V fazi inicializacije napolnimo L z elementi iz Σ_X.

- Nato se zaporedoma pomikamo skozi X in za vsak x_i poiščemo njegov položaj p v L.
- p shranimo v Y.
- Elemente v L od položaja 0 do položaja p premaknemo tako, da se njihovi položaji povečajo za ena, x_i pa nato postavimo na prvo mesto v L. Pravimo, da x_i premaknemo naprej.

Transformacija MTF tako transformira zaporedja ponavljajočih se enakih elementov v zaporedje ničel, zaporedja izmenjujočih se elementov v zaporedje enic, zaporedja izmenjujočih se parov v zaporedje dvojk itn. S tem lahko dosežemo, da je informacijska entropija Y manjša kot entropija X. Nizi s takšnimi lastnostmi seveda obstajajo, a so redki. Pri rastrskih slikah, na primer, sosednji piksli pogosto izgledajo enako, a njihove vrednosti se nekoliko razlikujejo. Pri senzorskih meritvah so si vrednosti zaporednih odčitkov pogosto blizu, a zelo redko povsem enake. Zato pri stiskanju podatkov pred MTF uporabimo BWT, ki tvori primernejšo permutacijo za MTF (takšno, kjer so enaki elementi blizu drug drugega). Žal pa BWT zahteva, da poznamo celoten X in posledično ni primerna za pretočno uporabo. Poleg vsega je učinkovita implementacija zahtevna tudi s programerskega stališča. V linearnem času jo sestavimo preko priponskega polja ali priponskega drevesa [8].

Transformacija MwI pa v nasprotju z MTF predpostavlja, da so zaporedni elementi v X pogosto sicer podobni, a ne popolnoma enaki. Deluje v dveh načinih: če je položaj p trenutnega elementa x_i v L manjši od uporabniško definiranega odstopanja δ, potem ostane v načinu MTF, sicer pa preuredi seznam tako, da na začetek seznama premakne $2\delta + 1$ elementov, in sicer v naslednjem vrstnem redu: $\langle x_i, x_i + 1, x_i - 1, x_i + 2, x_i - 2, \cdots x_i + \delta, x_i - \delta\rangle$. Na ta način dosežemo, da so podobni elementi elementu x_i blizu začetka L in zato takoj dobimo manjše vrednosti p. Pri slikah, recimo, ko naletimo na rob področja, lahko predpostavljamo, da bodo naslednji piksli podobni pikslu na robu. Podobno je tudi pri zaporedjih vzorcev iz digitalnega zvoka ali senzorjev. Transformacijo MwI razložimo še s psevdokodom 1. Inicializacijo seznama L opravimo v vrstici 2, tako da so vrednosti iz Σ_X prepletene okoli prve vrednosti iz X. Če Σ_X hrani vrednosti med 0 in 15 in je prva vrednost v X enaka 5, bi po inicializaciji imeli naslednjo stanje $L = \langle 5, 6, 4, 7, 3, 8, 2, 9, 1, 10, 0, 11, 12, 13, 14, 15\rangle$. Zato, da lahko inverzna transformacija MwI sestavi enako začetno stanje, vrednost x_0 shranimo v Y. V glavni zanki za vsak x_i v vrstici 5 poiščemo položaj v L, nato pa ga v naslednji vrstici shranimo v Y. Če je položaj manjši od δ, opravimo transformacijo MTF (vrstica 8), sicer pa tvorimo pomožno polje P (vrstica 10), v katerem je največ $1 + 2\delta$ prepletenih vrednosti okrog x_i, ki jih po strategiji MTF postavimo na začetek L v vrsticah 11 in 12.

MTF in MwI sicer iz teoretičnega stališča delujeta v pričakovanem času $O(|X|)$ [14], a ju je vseeno potrebno pazljivo implementirati. Na prvi pogled zgleda kritična funkcija VrniPoložaj, a jo lahko realiziramo s preslikovalno tabelo in se s tem izognemo iskanju. Večja težava

Psevdokod 1 Transformacija MwI

```
1:  function MwI(X, Σ_X, δ)
2:      L ← InicializirajL(x_0, Σ_X)
3:      Y ← ShraniPoložaj(Y, x_0)
4:      for i ← 1 to |X| do
5:          p ←VrniPoložaj(L, x_i)
6:          Y ← ShraniPoložaj(Y, p)
7:          if p ≤ (2 * δ) then
8:              L ← MTF(p,x_i, L)
9:          else
10:             P ← Prepletaj(x_i, δ)
11:             for j ← |P| − 1 downto 0 do
12:                 L ← MTF(P_j, L)
13:             end for
14:         end if
15:     end for
16:     return Y
17: end function
```

je premikanje vrednosti v L, ki jo opravlja funkcija MTF v vrsticah 8 in 12, saj zahteva poseg v pomnilnik. Analizo različnih implementacij najdemo v [15]. Dokler Σ_X vsebuje malo elementov, je tudi L kratek in v tem primeru transformacija MwI deluje hitro. V računalništvu je najpogostejša abeceda ASCII, ki s svojimi $2^8 = 256$ elementi ne predstavlja večjega zalogaja za MwI [13]. Situacija pa se spremeni, ko uporabimo MwI nad abecedami z 2^{16} ali celo 2^{32} elementi. Abecede takšnih velikosti so stalnica pri digitalnem zvoku, danes pa jih najdemo tudi pri rastrskih slikah in podatkih iz senzorskih meritev. Pri velikih abecedah se delovanje MwI drastično upočasni. V naslednjem poglavju predstavimo spremembo transformacije MwI, ki omogoča njeno uporabo tudi pri velikih abecedah.

3 Transformacije MwILA

Glavna težava transformacije MwI (in seveda tudi MTF) je poseganje v pomnilnik pri premeščanju elementov v seznamu L. Najboljša rešitev bi bila, če bi se teh posegov povsem znebili, a s tem bi izgubili dobre lastnosti transformacije, ko se elementi ponavljajo v vzorcih. Rešitev, ki jo predlagamo, je zato kompromis, in sicer:

- Če je $x_i \in \Delta$, kjer je $\Delta = [\delta_l, \delta_u]$, potem poiščemo položaj p v L ter opravimo transformacijo MTF z reorganizacijo L (torej s posegom v pomnilnik). Pri tem sta δ_l in δ_u kazalca v Σ_X in ju določimo z (1) in (2).

$$\delta_l = \begin{cases} x_i - \delta, & \text{če } x_i - \delta > 0 \\ 0, & \text{sicer} \end{cases} \tag{1}$$

$$\delta_u = \begin{cases} x_i + \delta, & \text{če } x_i + \delta < |\Sigma_X| \\ |\Sigma_X| - 1, & \text{sicer} \end{cases} \tag{2}$$

  Pričakujemo, da je $\delta \ll |\Sigma_X|$, zato bo tudi elementov v seznamu L malo ($|L| = \delta_u - \delta_l + 1$), s tem pa bo majhno tudi število posegov v pomnilnik.

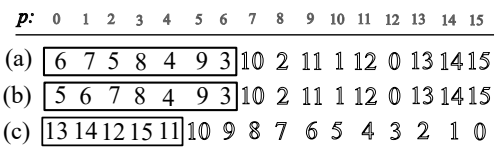
- V nasprotnem primeru položaj p izračunamo, kar nam omogoča prejšnja alineja, ki omejuje premikanje elementov v L samo znotraj Δ, za ostale elemente pa vemo, da so prepleteni okrog x_i.

Predlagano rešitev imenujemo MwILA (angl. MwI for Large Alphabets) in jo razložimo s psevdokodom 2. Algoritem se na prvi pogled ne razlikuje veliko od psevdokoda 1. Najprej v vrstici 2 določimo interval Δ, nato pa v naslednji vrstici napolnimo L z elementi iz Δ, prepletenimi okoli vrednosti x_0. V glavni zanki najprej določimo položaj p elementa x_i (določanju položaja bomo kmalu namenili več pozornosti) ter ga shranimo, nato pa preverimo, ali je x_i znotraj Δ. Če je, realiziramo transformacijo MTF, sicer pa v vrstici 12 določimo nov interval ter sestavimo nov L z vrednostmi, prepletenimi okrog x_i.

Psevdokod 2 Transformacija MwILA

```
1:  function MwI(X, Σ_X, δ)
2:      Δ ← DoločiDelta(x_0, δ,|Σ_X|)
3:      L ← InicializirajL(x_0, Δ)
4:      Y ← ShraniPoložaj(Y, x_0)
5:      x = x_0
6:      for i ← 1 to |X| do
7:          p ←VrniPoložaj(L, Δ, x, x_i, 0, |Σ|)
8:          Y ← ShraniPoložaj(Y, p)
9:          if x_i ∈ Δ then
10:             L ← MTF(p, x_i, L)
11:         else
12:             Δ ← DoločiDelta(x_i, δ,|Σ_X|)
13:             x = x_i
14:             L ← Inicializiraj(x_i, Δ)
15:         end if
16:     end for
17:     return Y
18: end function
```

Ključen del algoritma MwILA se skriva v funkciji VrniPoložaj, ki jo bomo razložili s sliko 1, ko so $\Sigma_X = \{0, 1, 2, \cdots, 15\}$, $\delta = 3$ in $X = \langle 6, 5, 13, 4\rangle$. Določimo $\Delta = [3, 9]$ ter prepletimo elemente okrog $x_0 = 6$. Od tega se jih $|\Delta| = \delta_u - \delta_l + 1 = 7$ nahaja v pomnilniku, to je seznamu L, (na sliki 1 je označen z okvirjem), ostali so določeni samo hipotetično (na sliki so ti elementi zapisani z očrtji). Vrednost prvega elementa shranimo v $Y = \langle 6\rangle$. Stanje po inicializaciji kaže slika 1a. Naslednji element $x_1 = 5$ se nahaja na položaju $p = 2$, tako da je $Y = \langle 6, 2\rangle$. Ker je $x_1 \in \Delta$, uporabimo MTF in dobimo stanje na sliki 1b. Element $x_2 = 13 \notin \Delta$, zato bomo njegov položaj izračunali v vrstici 7 v psevdokodu 2.



Slika 1: MwILA: (a) stanje po inicializaciji, (b) uporaba MTF, (c) stanje po novi inicializaciji.

Postopek izračuna, ki jo opravi funkcija VrniPoložaj, je naslednji:

1. Določimo največji in najmanjši element Σ kot $spMeja$ in $zgMeja$. V našem primeru je $spMeja = 0$ in $zgMeja = 15$.

2. Določimo stražarja st, ki kaže na mesto, kjer se konča prepletanje. Izračunamo $nLevo = x - spMeja$ in $nDesno = zgMeja - x$, kjer je x vrednost elementa, glede na katerega smo opravili prepletanje (v našem primeru je $x = 6$). Dobimo $nLevo = 6$ in $nDesno = 9$. Ker je $nLevo < nDesno$, je $st = 2 \cdot nLevo = 12$, sicer bi ga določili kot $st = 2 \cdot nDesno$. Prepletanje se torej konča na indeksu $st = 12$, kar preverimo na sliki 1a in b.

3. Za izračun potrebujemo še pomožno spremenljivko q, ki je 0, če je $x_i \leq x$, in -1 sicer. p izračunamo s (3). V našem primeru dobimo $p = 2\ \text{abs}(6-13) - 1 = 13$, kar lahko preverimo na sliki 1b. Stanje v $Y = \langle 6, 2, 13\rangle$.

$$p = 2\ \text{abs}(x - x_i) + q \tag{3}$$

4. Če je $p > st$, smo pristali v področju, kjer ni prepletanja, zato moramo izračunati nov položaj s (4).

$$p = \left\lfloor \frac{p - st + q}{2} \right\rfloor + st - q \tag{4}$$

V našem primeru imamo pred zadnjo iteracijo stanje na sliki 1c. Zadnji element $x_3 = 4$, ki je, kot vidimo na sliki, izven področja prepletanja. Izračunajmo: $nLevo = 13$ in $nDesno = 2$. $nDesno > nLevo$, zato $st = 4$, kar je tudi položaj, kjer se prepletanje konča

Psevdokod 3 Funkcija VrniPoložaj

```
1:  function VRNIPOLOŽAJ(L, Δ, x, x_i, spMeja, zgMeja)
2:      if x_i ∈ Δ then
3:          p ← 0
4:          while x_i ≠ L[p] do p ← p + 1
5:          end while
6:          return p
7:      end if
8:      nLevo ← x − spMeja
9:      nDesno ← zgMeja − x
10:     st ← 2 ∗ nLevo
11:     if nDesno < nLevo then
12:         st = 2 ∗ nDesno
13:     end if
14:     q ← 0
15:     if x_i > x then q = −1
16:     end if
17:     p ← 2 ∗ abs(x_i − x) + q
18:     if p > st then p ← ⌊(p−st+q)/2⌋ + st − q
19:     end if
20:     return p
21: end function
```

(glej sliko 1c). q dobi vrednost 0, saj je $4 < 13$. S (3) dobimo $p = 2\,\mathrm{abs}(4 - 13) + 0 = 18$, kar je večje od stražarja. Zato uporabimo še (4) ter izračunamo $p = \lfloor 0.5\,(18 - 4 + 0) \rfloor + 4 - 0 = 11$, kjer se $x_i = 4$ tudi nahaja. Transformacija MwILA je $Y = \langle 6, 2, 13, 11 \rangle$. V prvi stavek **if** v funkciji VrniPoložaj vstopimo, ko je $x_i \in \Delta$, ter s sprehodom skozi L poiščemo položaj p elementa x_i, sicer pa p določimo z opisanim izračunom.

4 Rezultati

V tem poglavju preverimo učinkovitost MwILA pri transformaciji digitalnega zvoka s 16-bitnimi vzorci. Tabela 1 kaže porabljen čas CPE transformacije MwI s parametrom $\delta = 50$ in MwILA s parametri $\delta = \{50, 250, 500\}$.

Vidimo, da je MwI izjemno počasna v nasprotju z MwILA, ki brez težav tudi pri večjih δ omogoča transformacijo v sprotnem času (testni zvočni posnetki A imajo dolžino dobrih 150 sekund). MwI sprotne transformacije nikakor ne omogoča. Pri tem je smiselno omeniti, da se rezultata transformacij MwI in MwILA sicer razlikujeta, a na zmanjšanje informacijske entropije med MwI in MwiLA ni bistvene razlike (zaznali smo jo na tretji decimalki). Pri zelo malih δ, na primer 2 ali 5, se porabljen čas CPE še dodatno zmanjša, a s tem izgubimo učinke transformacije MTF, ki, kot smo razložili, deluje znotraj območja Δ. Spomnimo, da MTF zazna ponavljajoče se vzorce v podatkih in jih pretvori v enake indekse.

Tabela 1: Demonstracija učinkovitosti transformacije MwILA, kjer so časi transformacij MwI in MwILA podani v sekundah

A	št.vzorcev	MwI(50)	MwILA		
			50	250	500
1	6507987	1.244,684	0,534	2,185	3,889
2	7089519	1.080,654	0,578	2,275	3,971
3	7624599	1.272,627	0,642	2,502	4,379

Za testiranje smo uporabili osebni 64-bitni računalnik s procesorjem AMD Ryzen 5 5500 s taktom 3,6 GHz in 32 GB hitrega pomnilnika. Testni program smo napisali v C++ v razvojnem okolju Visual studiu 2022 verzije 4.8.09032 in ga izvajali na operacijskem sistemu Windows 11. Vse podporne funkcije obeh implementacij so bile enake.

5 Zaključek

V članku smo predstavili transformacije nizov, ki nam, poleg drugih možnosti, omogočajo zmanjšati informacijsko entropijo nizov, s čimer entropijski kodirniki dosežejo boljša razmerja stiskanja. Predstavimo relativno novo transformacijo MwI, ki pa postane pri večjih abecedah, na primer pri zvoku, 16- ali 32-bitnih rastrskih slikah ali pri senzorskih podatkih, praktično neuporabna. Predlagamo spremembo transformacije, imenujemo jo transformacija MwILA, ki učinkovito zmanjša poseganje v pomnilnik in s tem brez težav omogoča podporo sprotni transformaciji zvočnih posnetkov.

Transformacija MwI (in MwILA) ima še kar nekaj možnosti nadaljnjih raziskav. Na primer, pri transformaciji besedila bi bilo smiselno vzpostaviti kontekstne odvisnosti zaporednih črk ter z njimi polniti seznam L ter tako dobiti transformiran vektor z majhnimi celoštevilčnimi vrednostmi.

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 državnih proračunov.

Literatura

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

[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] 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.

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

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

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

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

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

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

[11] 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.

[12] B. Žalik, Š. Kohek, S. Kolmanič, I. Kolingerová, D. Podgorelec, A. Jeromel. Kratek pregled metod predobdelave zaporedij pred postopki stiskanja podatkov. V: A. Žemva, A, Trost (ur.) Zbornik dvaintridesete mednarodne Elektrotehniške in računalniške konference, ERK 2023, 2023, 133–136.

[13] B. Žalik, D. Strnad, D. Podgorelec, I. Kolingerová, L. Lukač, N. LUKAČ, S. Kolmanič, K. Rizman Žalik, Š. Kohek. A new transformation technique for reducing information entropy : a case study on greyscale raster images. Entropy. 25(12), 2023, 1591.

[14] S. Irani. Two results on the list update problem, Information Processing Letters 38(6), 1991, 301–306.

[15] B. Žalik, M. Žalik, Mitja, B. Lipuš. On Move-To-Front Implementation. V: Proceedings of 27^{th} IEEE International Conference on Circuits, Systems, Communications and Computers, CSCC 2023, 43–47.