Napredujoče stiskanje rastrskih slik z algoritmom SPIHT

Luka Kovačič[1], Štefan Kohek[1], Blaž Repnik[1], Borut Žalik[1]

[1]*Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko, Maribor, Slovenia*
E-pošta: l.kovacic@um.si

Progressive raster image compression with SPIHT algorithm

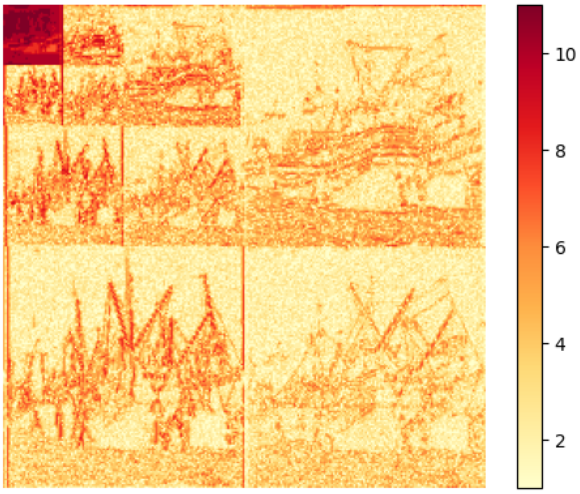
This paper presents SPIHT, an image compression algorithm based on discrete wavelet transform. Basic concepts and data structures necessary for understanding the algorithm are explained at first. An example of its functioning is provided after. At the end, compression efficiency is compared with the JPEG and JPEG 2000 formats on gray-scale raster images.

1 Uvod

Algoritem EZW (angl. Embedded Zerotree Wavelet) [1], ki ga je leta 1993 razvil J. M. Shapiro, predstavlja pomemben mejnik uporabe diskretne valčne transformacije pri stiskanju slik. Je prvi algoritem, ki je izkoristil korelacijo med hierarhično strukturiranimi koeficienti diskretne valčne transformacije. Omogoča napredujoči način stiskanja. Na temeljih, ki jih je postavil EZW, se je razvil algoritem SPIHT (angl. Set Partitioning in Hierarhical Trees) [2], ki je občutno izboljšal učinkovitost stiskanja. Algoritem ni zelo poznan, saj je bil predstavljen na težko razumljiv način. Namen tega članka je preprosto in jasno predstaviti osnovno idejo in delovanje algoritma.

2 Algoritem SPIHT

SPIHT je algoritem za stiskanje slik, ki deluje nad koeficienti diskretne valčne transformacije (angl. Discrete Wavelet Transform, DWT) [3]. Temelji na korelaciji med koeficienti na različnih nivojih DWT, ki spadajo v enak frekvenčni pas (vodoravne, navpične ali diagonalne podrobnosti) in predstavljajo enako področje na sliki (enaka prostorska orientacija). Ob večkratnem izvajanju DWT, glej sliko 1, ugotovimo, da so koeficienti v višjih nivojih načeloma večji po absolutni vrednosti. To pomeni, da za njihovo predstavitev potrebujemo več bitov. To je pričakovano, saj izvajamo DWT nad vedno bolj grobo predstavitvijo izvorne slike, kar pomeni ostrejše prehode in s tem tudi večje razlike med piksli. Če lahko vse koeficiente iste prostorske orientacije od nekega nivoja dekompozicije naprej predstavimo z manj biti kot trenutno mejno vrednost, potem lahko to označimo z enim bitom. Trenutna mejna vrednost je vedno potenca števila 2, kar nam omogoča preverbo, ali so biti koeficientov postavljeni (bit 1) ali ne (bit 0). Če so vsi biti 0 (vsi koeficienti so manjši od trenutne mejne vrednosti), se lahko izognemo pošiljanju bita za vsak koeficient posebej in podatke stisnemo.

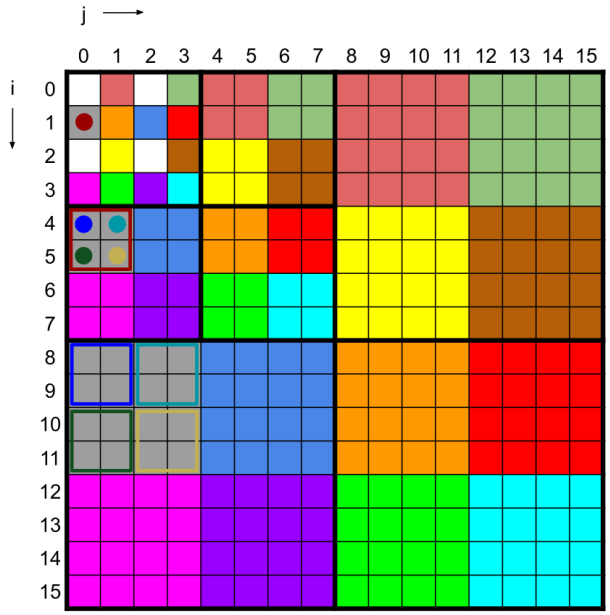


Slika 1: Število bitov, potrebnih za predstavitev koeficienta DWT (biti so označeni na desni strani slike)

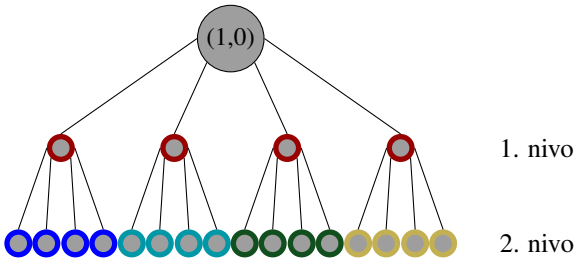
2.1 Drevesa prostorske orientacije

Drevesa prostorske orientacije (angl. Spatial Orientation Trees, SOT) so drevesa, katerih vozlišča oz. elementi spadajo v isto prostorsko orientacijo, obenem pa predpostavljajo enak frekvenčni pas. Recimo, da smo nad sliko velikosti 16×16 izvedli dvonivojsko DWT, torej je aproksimacija (pas LL na najvišjem nivoju DWT) velikosti 4×4. SPIHT razdeli koeficiente iz aproksimacije v neprekrivajoče se bloke velikosti 2×2. Levi zgornji elementi znotraj posameznih blokov (beli kvadratki na sliki 2) nimajo naslednikov. Ostali elementi znotraj blokov predstavljajo korene dreves in imajo neposredne naslednike izjemoma na istem nivoju DWT, ponazorjeni pa so z isto barvo. Vsak element v drevesu ima štiri neposredne naslednike (angl. offsprings). Element na položaju (1,0), prikazan s sivo barvo, ima neposredne naslednike na položajih (4,0), (4,1), (5,0) in (5,1). Slednji imajo svoje neposredne naslednike na naslednjem nivoju DWT (skupno 16 elementov, označenih s sivo barvo), ki predstavljajo posredne naslednike elementa na položaju (1,0). Relacijo med prej omenjenimi sivo obarvanimi koefici-

enti, v obliki štiriškega drevesa, prikazuje slika 3, kjer koren drevesa predstavlja element na položaju (1,0). Na prvem nivoju najdemo neposredne naslednike korena, vsi nadaljnji nivoji pa predstavljajo njegove posredne naslednike. Elementi (vozlišča), ki nimajo naslednikov, so listi drevesa.



Slika 2: Hierarhija med elementi v SOT (povzeto po [4])



Slika 3: Štiriško drevo iz slike 2 (povzeto po [4])

2.2 Podatkovne strukture

Algoritem SPIHT uporablja tri sezname:

- seznam nepomembnih pikslov (angl. List of Insignificant Pixels, LIP), vsebuje vrednosti, ki so manjše od trenutne mejne vrednosti,
- seznam pomembnih pikslov (angl. List of Significant Pixels, LSP), vsebuje vrednosti, ki so večje ali enake napram trenutni mejni vrednosti in
- seznam nepomembnih množic (angl. List of Insignificant Sets, LIS), vsebuje množice koeficientov DWT, določene s strukturo drevesa SOT, ki imajo vse elemente manjše od trenutne mejne vrednosti, torej so nepomembni.

Opozorimo, da smo ohranili terminologijo SPIHT; SPIHT namreč kodira koeficiente DWT in ne vrednosti pikslov. V seznamih hranimo položaje koeficientov znotraj matrike koeficientov. V LIP in LSP predstavljajo individualne koeficiente, v LIS pa množice, ki ponazarjajo drevesa SOT. Imamo dve vrsti množic:

- $D(i,j)$, sestavljajo jo vsi nasledniki, tako neposredni kot posredni, koeficienta na položaju (i,j). V štiriškem drevesu so to vsi elementi drevesa od vključno prvega nivoja dalje in
- $L(i,j)$, sestavljajo jo zgolj posredni nasledniki koeficienta na položaju (i,j). V štiriškem drevesu so to vsi elementi drevesa od vključno drugega nivoja dalje.

Uporabljali bomo še dve množici, ki pa nista del LIS:

- $O(i,j)$, sestavljajo jo vsi neposredni nasledniki koeficienta na položaju (i,j) in
- H, množico sestavljajo vsi koeficienti iz aproksimacije (pas LL) na najvišjem nivoju DWT.

Ugotovimo, da $L(i,j) = D(i,j) - O(i,j)$. $L(i,j)$ lahko predstavimo s štirimi množicami $D(k,l)$, kjer $(k,l) \in O(i,j)$.

2.3 Kodiranje

Kodiramo vsako bitno ravnino posebej, od najbolj pomembne do najmanj pomembne. S številom bitov, potrebnih za predstavitev največjega koeficienta DWT po absolutni vrednosti, določimo začetno mejno vrednost, s katero preverjamo pomembnost individualnih koeficentov ali množic (če imajo bite v trenutno obravnavani bitni ravnini postavljene ali ne). Če so pomembni, na izhod pošljemo bit 1 oz. bit 0, če niso. Mejna vrednost se vsako naslednjo iteracijo razpolovi, saj kodiramo manj pomembne bite. Na začetku kodiranja imamo v LIP vse koeficiente iz množice H, v LIS pa vstavimo kot izhodišča dreves vse koeficiente, ki ne predstavljajo levega zgornjega elementa znotraj posameznih blokov velikosti 2×2, saj ti nimajo naslednikov. LSP je na začetku prazen. Kodiranje posamezne bitne ravnine predstavlja eno iteracijo algoritma, ki poteka v treh fazah:

- Najprej na izhod pošljemo bite koeficientov iz LIP. Pošiljamo individualne bite (en bit za vsak koeficient), podatkov ne stiskamo. Za pomembne koeficiente na izhod pošljemo še en bit za predznak in jih premaknemo v LSP.
- Nato obravnavamo množice iz LIS. Podatke stisnemo, ko je množica nepomembna, saj imajo takrat vsi elementi množice bite enake 0. Na izhod pošljemo zgolj en bit (0) za celotno množico. V nasprotnem primeru množice manjšamo (iz množic tipa D preidemo na množice tipa L), odstranjene koeficente obravnavamo individualno in jih ustrezno uvrstimo v LIP oz. LSP. Če je tudi množica tipa L pomembna, se razdeli na štiri podmnožice tipa D. Postopek manjšanja oz. delitve množice ponavljamo, dokler množica ni nepomembna ali prazna.

- Nazadnje obdelamo LSP. Tudi tukaj, kot v obdelavi LIP, pošiljamo bite individualnih koeficentov, vendar le manj pomembne (brez njihovega najbolj pomembnega bita). Koeficente torej ob kodiranju njihovega najbolj pomembnega bita premaknemo v LSP. Ta faza se v terminologiji SPIHT imenuje korak izboljšav (angl. refinement pass).

Algoritem je možno prilagoditi za paralelno izvajanje [5]. Obstaja tudi različica algoritma SPIHT, imenovana NLS (angl. No List SPIHT) [6], ki deluje brez seznamov.

2.3.1 Primer delovanja

Slika 4 prikazuje matriko koeficientov, pridobljenih po dvonivojski hierarhični DWT. Za predstavitev največjega absolutnega koeficienta, ki je 31, potrebujemo 5 bitov, zato je mejna vrednost na začetku enaka 16. Začnemo s kodiranjem bitne ravnine na indeksu 4 (najmanj pomembna bitna ravnina je na indeksu 0). Na začetku imamo vse elemente iz bloka velikosti 2×2 v LIP. Vsi elementi, razen tistega na položaju (1,1) so pomembni, saj so po absolutni vrednosti večji od trenutne mejne vrednosti, ki je 16. Zanje na izhod dodatno pošljemo še bit za predznak, in jih premaknemo v LSP. Postopek prikazuje tabela 1.

31	25	-6	2	-2	3	0	0
17	13	4	5	5	3	-1	0
5	10	0	0	1	0	3	-6
-9	7	0	0	-2	0	-1	-1
0	1	12	-4	0	0	0	0
5	-2	-1	2	0	0	0	0
-3	1	4	0	0	0	0	0
0	-2	1	-1	0	0	0	0

Slika 4: Koeficienti dvonivojske DWT

Tabela 1: Obdelava LIP (indeks = 4)

test	pogoj	izhod	opravilo
(0,0)	$31 \geq 16$	1+	(0,0) v LSP
(0,1)	$25 \geq 16$	1+	(0,1) v LSP
(1,0)	$17 \geq 16$	1+	(1,0) v LSP
(1,1)	$13 \geq 16$	0	/

V tabeli 2 vidimo obdelavo LIS, v katerem imamo tri množice. Vsi elementi znotraj posameznih množic so po absolutni vrednosti manjši od trenutne mejne vrednosti, zato za vsako množico na izhod pošljemo bit 0. Tukaj vidimo glavno idejo algoritma, saj smo lahko z enim bitom povedali, da so biti vseh koeficientov znotraj posamezne množice enaki 0.

Tabela 2: Obdelava LIS (indeks = 4)

test	pogoj	izhod	opravilo
D(0,1)	$6 \geq 16$	0	/
D(1,0)	$12 \geq 16$	0	/
D(1,1)	$0 \geq 16$	0	/

Tabela 3 prikazuje obdelavo LSP, kjer za vsak koeficient pošljemo bit v trenutno obravnavani bitni ravnini, če ga še nismo. To se zgodi takrat, ko je koeficient po absolutni vrednosti večji ali enak napram mejni vrednosti iz prejšnje iteracije. Posledično v tej iteraciji izboljšav ni.

Tabela 3: Obdelava LSP (indeks = 4)

test	pogoj	izhod
(0,0)	$31 \geq 32$	/
(0,1)	$25 \geq 32$	/
(1,0)	$17 \geq 32$	/

Sledi nova iteracija, mejna vrednost je sedaj 8. Edini koeficient v LIP, koeficient na položaju (1,1), je postal pomemben. Kot prikazuje tabela 4, na izhod pošljemo bit 1 in njegov predznak ter ga premaknemo v LSP.

Tabela 4: Obdelava LIP (indeks = 3)

test	pogoj	izhod	opravilo
(1,1)	$13 \geq 8$	1+	(1,1) v LSP

Množica $D(0,1)$ ni pomembna, zato na izhod pošljemo bit 0. Množica $D(1,0)$ je pomembna, zato na izhod pošljemo bit 1 in testiramo elemente množice $O(1,0)$. Zanje na izhod pošljemo bit na indeksu 3. Nepomembne vstavimo v LIP, za pomembne pa pošljemo še njihov predznak in jih vstavimo v LSP. Spremenimo še tip množice in jo premaknemo na konec seznama. Množica $D(1,1)$ ni pomembna, zato na izhod damo bit 0. Vsi elementi množice $L(1,0)$ niso manjši od trenutne mejne vrednosti, zato moramo neposredne naslednike vstaviti v LIS kot nova izhodišča dreves. Obenem odstranimo množico $L(1,0)$ iz LIS. Postopek nadaljujemo kot prikazuje tabela 5. V LSP imamo sedaj koeficiente, ki v seznam niso bili dodani v trenutni iteraciji algoritma, zanje pošljemo bit v trenutno obravnavani bitni ravnini, kot prikazuje tabela 6. Kodiranje lahko nadaljujemo vse do najmanj pomembne bitne ravnine. Dekodiranje je podobno kodiranju, le da bite za kodirane koeficiente in množice beremo iz vhoda.

Tabela 5: Obdelava LIS (indeks = 3)

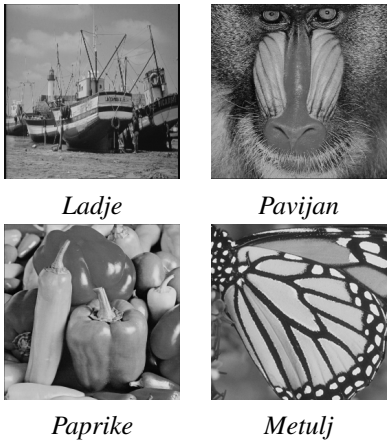
test	pogoj	izhod	opravilo
D(0,1)	$6 \geq 8$	0	/
D(1,0)	$12 \geq 8$	1	test neposrednih naslednikov
(2,0)	$5 \geq 8$	0	(2,0) v LIP
(2,1)	$10 \geq 8$	1+	(2,1) v LSP
(3,0)	$9 \geq 8$	1-	(3,0) v LSP
(3,1)	$7 \geq 8$	0	(3,1) v LIP
			sprememba D(1,0) v L(1,0)
D(1,1)	$0 \geq 8$	0	/
L(1,0)	$12 \geq 8$	1	vstavitev množic D(2,0), D(2,1), D(3,0), D(3,1) in odstranitev množice L(1,0)
D(2,0)	$5 \geq 8$	0	/
D(2,1)	$12 \geq 8$	1	test neposrednih naslednikov
(4,2)	$12 \geq 8$	1+	(4,2) v LSP
(4,3)	$4 \geq 8$	0	(4,3) v LIP
(5,2)	$1 \geq 8$	0	(5,2) v LIP
(5,3)	$2 \geq 8$	0	(5,3) v LIP
			odstranitev množice D(2,1)
D(3,0)	$3 \geq 8$	0	/
D(3,1)	$4 \geq 8$	0	/

Tabela 6: Obdelava LSP (indeks = 3)

test	pogoj	izhod
(0,0)	$31 \geq 16$	1
(0,1)	$25 \geq 16$	1
(1,0)	$17 \geq 16$	0
(1,1)	$13 \geq 16$	/
(2,1)	$10 \geq 16$	/
(3,0)	$9 \geq 16$	/
(4,2)	$12 \geq 16$	/

3 Rezultati

Učinkovitost algoritma SPIHT smo preizkusili na naboru 8-bitnih sivinskih rastrskih slik velikosti 256×256, prikazanih na sliki 5. Rezultate smo pridobili pri petnivojski dekompoziciji z DWT, uporabljen je bil valček bior4.4 [7]. Učinkovitost algoritma smo primerjali s formatoma JPEG in JPEG 2000. Izračunali smo razmerja stiskanja (angl. Compression Ratio, CR) pri čimbolj podobnih vrednostih metrike SSIM [8]. Za JPEG je podan še faktor kakovosti Q. Rezultati so prikazani v tabeli 7. Učinkovitost stiskanja se pričakovano izboljšuje s slabšanjem kakovosti stisnjene slike (manjša vrednost metrike SSIM). SPIHT je v večini primerov dosegel slabše rezultate kot JPEG 2000, a boljše kot JPEG.



Slika 5: Testne slike

4 Zaključek

V članku predstavimo SPIHT, algoritem za napredujoče stiskanje rastrskih slik, ki temelji na diskretni valčni transformaciji. Algoritem primerjamo s formatoma JPEG in JPEG 2000. Slednji temelji na SPIHT z izboljšanim konceptom EBCOT (angl. Embedded Block Coding with Optimal Truncation), ki daje boljše razmerje stiskanja. JPEG 2000 je novejši tudi napram formatu JPEG, zato je pričakovano dosegel najboljše rezultate. Sledi SPIHT, JPEG pa se je odrezal najslabše.

Zahvala
Projekt (Paradigma stiskanja podatkov z odstranjevanjem obnovljivih informacij, št. J2-4458) je financirala Javna agencija za znanstvenoraziskovalno in inovacijsko dejavnost Republike Slovenije iz državnega proračuna.

Tabela 7: Rezultati izgubnega stiskanja

Ladje						
SPIHT	CR	3,2	4,8	8,3	16,7	42,7
	SSIM [%]	98,8	97,0	92,6	84,3	71,7
JPEG	CR	2,6	4,2	7,3	14,8	29,5
	SSIM [%]	98,8	96,8	92,7	84,0	70,0
	Q	94	84	56	18	6
JPEG 2000	CR	3,3	6,1	9,6	18,0	42,3
	SSIM [%]	99,0	96,6	93,4	86,2	73,5
Pavijan						
SPIHT	CR	1,9	2,7	4,6	10,0	31,6
	SSIM [%]	99,3	97,0	90,5	76,6	53,7
JPEG	CR	1,5	2,4	4,1	9,7	26,7
	SSIM [%]	99,5	97,0	90,7	76,4	54,3
	Q	96	88	68	22	6
JPEG 2000	CR	2,2	3,2	5,8	11,7	54,8
	SSIM [%]	99,2	96,7	88,9	78,3	46,0
Paprike						
SPIHT	CR	3,8	6,7	11,6	21,2	45,7
	SSIM [%]	98,5	96,4	93,4	87,7	78,8
JPEG	CR	2,8	6,0	10,3	17,0	25,0
	SSIM [%]	98,6	96,3	93,3	88,1	80,6
	Q	94	76	40	16	8
JPEG 2000	CR	4,4	7,0	16,1	29,8	43,3
	SSIM [%]	98,4	97,0	92,6	86,4	80,7
Metulj						
SPIHT	CR	2,6	3,8	6,1	11,0	24,2
	SSIM [%]	98,9	97,5	94,6	89,2	80,1
JPEG	CR	2,1	3,1	5,1	10,1	16,8
	SSIM [%]	98,9	97,4	94,6	88,9	81,1
	Q	94	86	64	20	8
JPEG 2000	CR	3,0	4,6	6,7	13,8	29,6
	SSIM [%]	98,8	97,2	95,2	88,1	78,6

Literatura

[1] J. M. Shapiro, Embedded image coding using zerotrees of wavelet coefficients, IEEE Transactions on Signal Processing 41(12), 1993, 3445-3462.

[2] A. Said, W. A. Pearlman, A new, fast, and efficient image codec based on set partitioning In hierarchical trees, IEEE Transactions on Circuits and Systems for Video Technology 6(3), 1996, 243-250.

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

[4] L. Kovačič, Implementacija SPIHT: Magistrsko delo. Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko, 2024.

[5] Y. Jin, H. J. Lee, A block-based pass-parallel SPIHT Algorithm, IEEE Transactions on Circuits and Systems for Video Technology, 22(7), 2012, 1064-1075.

[6] F. W. Wheeler, W. A. Pearlman, SPIHT image compression without lists, IEEE International Conference on Acoustics, Speech, and Signal Processing, 2000, 2047-2050.

[7] Bior4.4, https://www.mathworks.com/help/wavelet/ref/wfilters.html (dostop 01.07.2024).

[8] Z. Wang, A. Bovik, H. Sheikh, E. Simoncelli, Image quality asesment: from error visibility to structural similarity, IEEE Transactions on Image Processing, 13(4), 2004, 600-612.