KAUNO TECHNOLOGIJOS UNIVERSITETAS

FUNDAMENTALIŲJŲ MOKSLŲ FAKULTETAS

TAIKOMOSIOS MATEMATIKOS KATEDRA

Justas Žydelis

PAGERINTA NULINIŲ MEDŽIŲ PAIEŠKOS SCHEMA SPIHT ALGORITME

Magistro darbas

Vadovas
prof. dr. J. Valantinas

KAUNAS, 2011

KAUNO TECHNOLOGIJOS UNIVERSITETAS

FUNDAMENTALIŲJŲ MOKSLŲ FAKULTETAS

TAIKOMOSIOS MATEMATIKOS KATEDRA

TVIRTINU
Katedros vedėjas
doc.dr. N.Listopadskis
2011 06 01

PAGERINTA NULINIŲ MEDŽIŲ PAIEŠKOS SCHEMA SPIHT ALGORITME

Matematikos magistro baigiamasis darbas

Vadovas
prof.dr.J.Valantinas
2011 06 01

Recenzentas
doc. R. Marcinkevičius
2011 06 01

Atliko
FMMM 9 gr. stud.
J. Žydelis
2011 06 01

KAUNAS, 2011

KVALIFIKACINĖ KOMISIJA

Pirmininkas: Leonas Saulis, profesorius (VGTU)

Sekretorius: Eimutis Valakevičius, docentas (KTU)

Nariai: Algimantas Jonas Aksomaitis, profesorius (KTU)

Vytautas Janilionis, docentas (KTU)

Vidmantas Povilas Pekarskas, profesorius (KTU)

Rimantas Rudzkis, habil. dr., vyriausiasis analitikas (DnB NORD Bankas)

Zenonas Navickas, profesorius (KTU)

Arūnas Barauskas, dr., vice-prezidentas projektams (UAB „Baltic Amadeus“)

Žydelis J. Improved scheme for the zero-tree search in the SPIHT algorithm: Master‘s work in applied mathematics / supervisor prof. dr. J. Valantinas; Department of Applied mathematics, Faculty of Fundamental Sciences, Kaunas University of Technology. – Kaunas, 2011. – 53 p.

SUMMARY

Embedded zero-tree wavelet (EZW) coding, introduced by J. M. Shapiro in 1993, was first of its kind. Its effectiveness and computational simplicity was competitive with most of the digital image coding algorithms. Following its ideas, Amir Said and William A. Pearlman in 1995, introduced SPIHT (Set Partitioning In Hierarchical Trees) algorithm, which provided even better results than EZW.

In this paper we explain the main working principles behind SPIHT algorithm, reveal its strengths and weaknesses and propose a novel scheme for the accelerated analysis of quad-trees in the discrete wavelet spectrum of a digital image.

TURINYS

LENTELIŲ SĄRAŠAS

PAVEIKSLŲ SĄRAŠAS

ĮVADAS

Skaitmeninių vaizdų suglaudinimo algoritmas EZW (Embedded - Zero-tree – Wavelet, [3]), 1993 metais pasiūlytas J. M. Shapiro, buvo pirmas tokio tipo, kuris savo efektyvumu ir skaičiavimo paprastumu pralenkė tuometinius algoritmus. Įkvėpti šios idėjos Amir Said ir William A. Pearlman 1995 metais pasiūlė būdą, kaip panaudojant EZW algoritmo veikimo principus racionalizuoti vaizdų suglaudinimo procedūrą, ir pateikė SPIHT (Set Partitioning In Hierarchical Trees, [4]) algoritmą.

Šiame darbe paaiškinsime SPIHT algoritmo veikimo principus, apžvelgsime jo gerąsias ir blogąsias puses bei pateiksime alternatyvią nulinių medžių paieškos vaizdo diskrečiajame bangelių spektre schemą, kuri laikinių sąnaudų požiūriu pastebimai pagerina SPIHT algoritmo darbą.

1 DISKREČIOSIOS BANGELIŲ TRANSFORMACIJOS (DBT) IR JŲ TAIKYMAS VAIZDAMS GLAUDINTI

Diskrečioji bangelių transformacija (DBT), be jokios abejonės, yra viena naujausių priemonių, leidžiančių įveikti su Furjė transformacija (DFT) susijusias problemas (trūkumus), tame tarpe, svarbiausią iš jų – lokalizavimo erdvėje problemą. Bangelių analizėje, apdorojant vieną ar kitą skaitmeninį signalą, natūraliai formuojamas kintamo dydžio (mastelio) langas, kuris slenka išilgai laiko (erdvės) ašies, ir kiekvienai lango pozicijai apskaičiuojamas bangelių spektras (transformacija). Procesas kartojamas daug kartų. Galutinis rezultatas – signalo išraiškų (vaizdavimų pagal dažnį ir erdvėje) rinkinys.

Kiekvienas diskrečiosios bangelių transformacijos (DBT) realizavimo žingsnis (iteracija) panaudoja taip vadinamą mastelio funkciją įvesties duomenims (skaitmeniniam signalui) apdoroti. Jeigu pradinis signalas X turi N ($N = 2^n, n \in \mathrm{N}$) reikšmių, tai mastelio funkcija bus panaudota tam, kad būtų apskaičiuotos $N/2$ suvidurkintos reikšmės. Gautosios reikšmės saugomos viršutinėje duomenų vektoriaus (iš N elementų) dalyje.

Kita (bangelių) funkcija, kiekviename DBT realizavimo žingsnyje, taip pat yra panaudojama įvesties duomenims apdoroti. Jei pradinis signalas X turi N reikšmių, tai bangelių funkcija bus pritaikyta tam, kad apskaičiuoti $N/2$ skirtumines (atspindinčias informacijos pasikeitimus signale X) reikšmes. Šios skirtuminės reikšmės yra saugomos apatinėje duomenų vektoriaus (iš N elementų) dalyje.

Kitoje iteracijoje (žingsnyje) abi funkcijos (mastelio ir bangelės) taikomos tiktai suvidurkintų reikšmių vektoriui, gautam prieš tai atliktoje iteracijoje.

Po baigtinio iteracijų (žingsnių) skaičiaus iš skaitmeninio signalo X yra suformuojamas DBT spektras Y.

DBT spektras Y apima vienintelę suvidurkintą reikšmę (gautą n – tojoje iteracijoje) ir sutvarkytą skirtuminių reikšmių rinkinį (gautą ankstesnėse iteracijose).

1.1 BENDROSIOS BANGELIŲ SAVYBĖS IR DBT SKAIČIAVIMO ALGORITMAI

Pradėsime nuo tolydžiosios bangelių transformacijos (TBT) apibrėžimo, būtent:

$$F(s,\tau) = \int f(t) \cdot \overline{\Psi}_{s,\tau}(t)dt, \quad \textbf{(1.1)}$$

kur $\Psi_{s,\tau}(t)$ yra bazinės funkcijos (bangelės). Kintamieji s ir τ (atitinkamai mastelis ir poslinkis) yra nauji bangelių transformacijos argumentai.

Atvirkštinė bangelių transformacija užrašoma taip:

$$f(t) = \iint F(s,\tau)\Psi_{s,\tau}(t)d\tau ds. \quad \textbf{(1.2)}$$

Bangelės generuojamos, panaudojant bazinę („motininę“) bangelę $\Psi(t)$, t.y.

$$\Psi_{s,\tau}(t) = \frac{1}{\sqrt{s}}\Psi\left(\frac{t-\tau}{s}\right). \quad \textbf{(1.3)}$$

Pastarojoje išraiškoje, s yra mastelį keičiantis koeficientas, ir τ – poslinkio koeficientas. Daugiklis $1/\sqrt{s}$ yra normalizavimo koeficientas, įvertinant skirtingas mastelio koeficientų reikšmes.

Kaip matome iš (1.1) išraiškos, vienmatės funkcijos bangelių transformacija yra dvimatė, o dvimatės funkcijos bangelių transformacija, akivaizdu, yra keturmatė.

Aptarsime kai kurias pagrindines bangelių savybes. Bene svarbiausios iš jų - „tinkamumo“ ir „reguliarumo“ sąlygos. Sakoma, jog integruojama kvadratu funkcija$\Psi(t)$, tenkina „tinkamumo“ sąlygą (savybę), jeigu

$$\int \frac{|\Psi(\omega)|^2}{|\omega|}d\omega < +\infty. \quad \textbf{(1.4)}$$

Šioje išraiškoje $\Psi(\omega)$ žymi funkcijos$\Psi(t)$ Furjė transformaciją. Iš (1.4) nelygybės tiesiogiai išplaukia, kad Furjė transformacija $\Psi(\omega)$prilygsta nuliui, kai ω dažnis yra lygus 0, t.y.

$$|\Psi(\omega)|^2|_{\omega=0} = 0. \quad \textbf{(1.5)}$$

Tai reiškia, jog bangelės turi turėti dažnių juostą, apribotą pagal dažnį. Tuo pačiu bangelės (kaip signalo) pastovi dedamoji lygi nuliui, t.y.

$$\int \Psi(t)dt = 0. \qquad \textbf{(1.6)}$$

Kitaip tariant, $\Psi(t)$turi būti bangelė tikrąja to žodžio prasme.

Bangelėms (bangelių funkcijoms) dažnai keliamos papildomos sąlygos, leidžiančios pagerinti bangelių skleidinių konvergavimo greitį. Šios sąlygos siejamos su bangelių „reguliarumo“ savybe. Pastaroji savybė suprantama kaip reikalavimas, jog bangelė būtų pakankamai glodi ir sukoncentruota tiek laiko, tiek dažnių srityse. „Reguliarumas“ yra gana sudėtinga sąvoka. Jos paaiškinimui ir interpretacijai pasinaudosime „nykstančiais“ pradiniais momentais.

Išskleidę bangelių transformaciją ((1.1) išraišką) Teiloro eilute taško t = 0 aplinkoje (paprastumo dėlei, imkime τ = 0), gausime:

$$F(s,0) = \frac{1}{\sqrt{s}}[\sum_{p=0}^{n} f^{(p)}(0) \int \frac{t^p}{p!} \Psi\left(\frac{t}{s}\right) dt + O(n+1)]; \qquad \textbf{(1.7)}$$

čia $f^{(p)}$ žymi p-osios eilės (funkcijos f) išvestinę, o $O(n+1)$- skleidinio liekamąjį narį. Toliau, pažymėję bangelės p-osios eilės pradinį momentąM_p, t.y.

$$M_p = \int t^p \Psi(t)dt, \qquad \textbf{(1.8)}$$

galime (1.7) išraišką perrašyti taip:

$$F(s,0) = \frac{1}{\sqrt{s}}[f(0)M_0 s + \frac{f^{(1)}(0)}{1!} M_1 s^2 +$$
$$+ \frac{f^{(2)}(0)}{2!} M_2 s^3 + \cdots + \frac{f^{(n)}(0)}{n!} M_n s^{n+1} + O(s^{n+2})]. \qquad \textbf{(1.9)}$$

Remiantis „tinkamumo“ sąlygą, galime teigti, kad$M_0 = 0$. Tokiu būdu, pirmasis (1.9) išraiškos narys lygus nuliui. Jeigu ir daugiau (tarkim, iki n-osios eilės) pradinių momentų prilygtų nuliui, skleidinio (bangelių transformacijos) koeficientai $F(s,\tau)$ tolydžiam signalui $f(t)$„gestų“ greičiu s^{n+2} $(0 < s < 1)$. Beje, pradiniai momentai nebūtinai turi būti lygūs nuliui - pakanka, jog jie būtų artimi nuliui.

Apibendrinant tai, kas buvo pasakyta, galima teigti, jog „tinkamumo“ sąlygos išpildymas sąlygoja bangelės svyravimus, o „reguliarumo“ sąlygos išpildymas bei nykstantys momentai – greitesnį bangelių skleidinių konvergavimą.

1.2 DISKREČIOJI LE GALL‘O BANGELIŲ TRANSFORMACIJA (DLGT): SAVYBĖS, SKAIČIAVIMO ALGORITMAI

N-mačio ($N = 2^n, n \in \mathrm{N}$) duomenų vektoriaus $X = (x_0 x_1 \ \dots \ x_{N-1})^T$ diskrečioji Le Gall‘o transformacija (DLGT) apibrėžiama taip:

$$Y_X = (Y_0 Y_1 \ \dots \ Y_{N-1})^T = \mathrm{G}(n) \cdot X; \qquad \textbf{(1.10)}$$

čia G(n) yra DLGT matrica, kurios eilutės (baziniai vektoriai) gaunamos, diskretizuojant baigtinį skaičių pirmųjų Le Gall‘o funkcijų.

Kadangi $\mathrm{G}^T(n) \cdot \mathrm{G}(n) = \mathrm{E}(n)$, tai atvirkštinė Le Gall‘o transformacija (ALGT) nusakoma lygybe

$$X = \mathrm{G}^T(n) \cdot Y_X. \qquad \textbf{(1.11)}$$

Pastebėsime, jog Le Gall‘o „motininės“ bangelės funkcija $\psi(t)$ užrašoma taip:

$$\psi(t) = \begin{cases} -\frac{1}{2}, & 0 \le t < \frac{1}{3}, \\ 1, & \frac{1}{3} \le t < \frac{2}{3}, \\ -\frac{1}{2}, & \frac{2}{3} \le t < 1, \\ 0, & kitu\ atveju, \end{cases} \qquad \textbf{(1.12)}$$

tuo tarpu Le Gall‘o bangelių mastelio funkciją $\varphi(t)$:

$$\varphi(t) = \begin{cases} -\frac{1}{8}, & 0 \le t < \frac{1}{5}, \\ \frac{1}{4}, & \frac{1}{5} \le t < \frac{2}{5}, \\ \frac{3}{4}, & \frac{2}{5} \le t < \frac{3}{5}, \\ \frac{1}{4}, & \frac{3}{5} \le t < \frac{4}{5}, \\ -\frac{1}{8}, & \frac{4}{5} \le t < 1, \\ 0, & kitu\ atveju. \end{cases} \qquad \textbf{(1.13)}$$

Viena svarbiausių DLGT savybių – lokalizavimas erdvėje. Ši savybė gali būti paaiškinama taip – kuo didesnis DLGT koeficiento eilės numeris, tuo mažesnį duomenų vektoriaus (signalo) *X* plotelį jisai „atspindi“.

1.2.1 BAZINĖ DLGT

Kadangi tiesioginis diskrečiosios Le Gall‘o transformacijos skaičiavimas didelių matmenų vektoriams užima daug laiko, tai praktiniams taikymams naudojamas greitasis DLGT spektro skaičiavimo algoritmas (Lifting scheme, [2]).

Duomenų vektorių X perrašome taip:

$$X = (x_0 x_1 \dots x_{N-1})^T = \left(s_0^{(0)} s_1^{(0)} \dots s_{2^n-1}^{(0)}\right)^T = (o_0 e_0 o_1 e_1 \dots o_{2^{n-1}-1} e_{2^{n-1}-1})^T, \quad \textbf{(1.14)}$$

o_j – lyginiai duomenų vektoriaus X elementai, o e_j – nelyginiai; n – transformacijai realizuoti reikalingas iteracijų skaičius.

Po pirmos iteracijos gauname vektorių

$$Y^{(1)} = (s_0^{(1)} s_1^{(1)} \dots s_{2^{n-1}-1}^{(1)} d_0^{(1)} d_1^{(1)} \dots d_{2^{n-1}-1}^{(1)})^T, \quad \textbf{(1.15)}$$

kur

$$d_k^{(i)} = e_k^{(i-1)} - \frac{1}{2}(o_k^{(i-1)} + o_{k+1}^{(i-1)}),\ s_k^{(i)} = o_k^{(i-1)} + \frac{1}{4}(d_{k-1}^{(i)} + d_k^{(i)}), \quad \textbf{(1.16)}$$

su visais$k = 0,1, \dots, 2^{n-i} - 1$; be to $o_{2^{n-i}} \coloneqq o_{2^{n-i}-1}$ ir $d_{-1} \coloneqq d_0$.

Gautas reikšmes s_k naudojame kaip tarpinį duomenų vektorių pratęsiant procedūrą. Po i-tosios ($i \in \{1,2, \dots, n-1\}$) iteracijos gausime vektorių

$$Y^{(i)} = (s_0^{(i)} s_1^{(i)} \dots s_{2^{n-i}-1}^{(i)} d_0^{(i)} d_1^{(i)} \dots d_{2^{n-i}-1}^{(i)})^T. \quad \textbf{(1.17)}$$

Pagaliau po n-tosios iteracijos turėsime vektorių $Y^{(n)}$.

Atlikus *n* iteracijų gaunamas (suformuojamas) DLGT spektras *Y* signalui *X*, būtent:

$$Y = (s_0^{(n)} d_0^{(n)} d_0^{(n-1)} d_1^{(n-1)} d_0^{(n-2)} d_1^{(n-2)} \dots d_0^{(1)} d_1^{(1)} \dots d_{2^{n-1}-1}^{(1)})^T. \quad \textbf{(1.18)}$$

Viršutiniuose skliaustuose įrašyti skaičiai (kintantys nuo 1 iki *n*) žymi, kurios iteracijos metu buvo gauta atitinkama reikšmė.

Pastebime, kad norint apskaičiuoti s_k, reikia visų pirma apskaičiuoti d_k.

Žemiau pateikiamas tiesioginės DLGT apskaičiavimo algoritmas.

Algoritmas1 /Tiesioginė DLGT/

1. $i \coloneqq 1$.
2. Su kiekviena reikšme $k = 0,1, \dots, 2^{n-i} - 1$apskaičiuojami atitinkamai mastelio ir bangelės funkcijų taikymo duomenų vektoriui $S^{(i-1)}$ rezultatai:
   $d_k^{(i)} = e_k^{(i-1)} - \frac{1}{2}(o_k^{(i-1)} + o_{k+1}^{(i-1)})$,
   $s_k^{(i)} = o_k^{(i-1)} + \frac{1}{4}(d_{k-1}^{(i)} + d_k^{(i)})$.
3. Jei $i<n$, tai $i \coloneqq i + 1$ ir pereiname į 2 punktą.

4. Pabaiga.

Ieškant atvirkštinės DLGT (ADLGT), naudojamos formulės

$$o_k^{(i-1)} = s_k^{(i)} - \frac{1}{4}(d_{k-1}^{(i)} + d_k^{(i)}),\ e_k^{(i-1)} = d_k^{(i)} + \frac{1}{2}(o_k^{(i-1)} + o_{k+1}^{(i-1)}) \quad \textbf{(1.19)}$$

bei tos pačios pataisos, kaip ir skaičiuojant DLGT. Beje, norint apskaičiuoti e_k, pirma reikia apskaičiuoti o_k.

ADLGT realizuojantis algoritmas pateikiamas žemiau:

Algoritmas2 /Atvirkštinė DLGT/

Kintamajam i nuosekliai priskiriamos reikšmės $n, n-1, \dots, 2, 1$. Remiantis vektoriais $S^{(i)} = (s_0^{(i)} s_1^{(i)} \dots s_{2^{n-i}-1}^{(i)})^T$ ir $D^{(i)} = (d_0^{(i)} d_1^{(i)} \dots d_{2^{n-i}-1}^{(i)})^T$, atkuriamas vektorius $S^{(i-1)} = (s_0^{(i-1)} s_1^{(i-1)} \dots s_{2^{n-i+1}-1}^{(i-1)})^T = (o_0^{(i-1)} e_0^{(i-1)} o_1^{(i-1)} e_1^{(i-1)} \dots o_{2^{n-i}-1}^{(i-1)} e_{2^{n-i}-1}^{(i-1)})^T$.

1. $i \coloneqq n$.

2. Su kiekviena reikšme $k = 0, 1, \dots, 2^{n-i} - 1$ apskaičiuojamos skaitinės vektoriaus $S^{(i-1)}$ reikšmės

$$o_k^{(i-1)} = s_k^{(i)} - \frac{1}{4}(d_{k-1}^{(i)} + d_k^{(i)}),$$

$$e_k^{(i-1)} = d_k^{(i)} + \frac{1}{2}(o_k^{(i-1)} + o_{k+1}^{(i-1)}).$$

3. Jei $i > 0$, tai $i \coloneqq i - 1$ ir pereiti į 2 punktą.

4. Pabaiga. Atkurtas pradinis duomenų vektorius (vaizdas) $X = (x_0 x_1 \dots x_{2^n-1})^T = S^{(0)} = (o_0^{(0)} e_0^{(0)} o_1^{(0)} e_1^{(0)} \dots o_{N/2-1}^{(0)} e_{N/2-1}^{(0)})^T$.

Tiek tiesioginės, tiek atvirkštinės DLGT apskaičiavimo algoritmus realizuojančios programos (jų kodai) patalpinti 1 priede.

Praktikoje sutinkami atvejai, kai dirbame su sveikaisiais skaičiais, t.y. kai norima suglaudinti skaitmeninį signalą (vaizdą), o po to atkurti jį be informacijos praradimo. Pastebėsime, jog informacijas praradimas sąlygoja kompiuteriniuose skaičiavimuose naudojamas baigtinis tikslumas.

1.2.2 MODIFIKUOTA DLGT

Jeigu žinome, jog duomenų vektorius sudarytas tik iš sveikųjų skaičių, tai galime naudoti DLGT, skirtą darbui su sveikaisiais skaičiais. Jos pagalba gaunamas spektras sudarytas tik iš sveikųjų skaičių. Atsiranda galimybė atkurti visiškai tikslius pradinius duomenų vektorius.

Modifikuotą DLGT nusako išraiškos:

$$d_k^{(i)} = e_k^{(i-1)} - [\frac{1}{2}\left(o_k^{(i-1)} + o_{k+1}^{(i-1)}\right)],\ s_k^{(i)} = o_k^{(i-1)} + [\frac{1}{4}\left(d_{k-1}^{(i)} + d_k^{(i)}\right) + \frac{1}{2}], \quad \textbf{(1.20)}$$

su visais $k = 0,1, \dots, 2^{n-i} - 1$ ir $i \in \{1,2, \dots, n\}$

Pastarosiose išraiškose laužtiniai skliaustai reiškia skaičiaus apvalinimą iki sveikosios jo dalies. Taikomos tos pačios pataisos bei ta pati skaičiavimo tvarka, kaip ir DLGT atveju.

Algoritmas1 /Tiesioginė modifikuota DLGT/

1. $i \coloneqq 1$.
2. Su kiekviena reikšme $k = 0,1, \dots, 2^{n-i} - 1$apskaičiuojami atitinkamai mastelio ir bangelės funkcijų taikymo duomenų vektoriui $S^{(i-1)}$ rezultatai:
   $d_k^{(i)} = e_k^{(i-1)} - [\frac{1}{2}\left(o_k^{(i-1)} + o_{k+1}^{(i-1)}\right)]$,
   $s_k^{(i)} = o_k^{(i-1)} + [\frac{1}{4}\left(d_{k-1}^{(i)} + d_k^{(i)}\right) + \frac{1}{2}]$.
3. Jei $i < n$, tai $i \coloneqq i + 1$ ir pereiname į 2 punktą.
4. Pabaiga.

Ieškant atvirkštinės DLGT (ADLGT), naudojamos formulės

$$o_k^{(i-1)} = s_k^{(i)} - \frac{1}{4}(d_{k-1}^{(i)} + d_k^{(i)}),\ e_k^{(i-1)} = d_k^{(i)} + \frac{1}{2}(o_k^{(i-1)} + o_{k+1}^{(i-1)}) \quad \textbf{(1.21)}$$

bei tos pačios pataisos, kaip ir skaičiuojant DLGT. Beje, norint apskaičiuoti e_k, pirma reikia apskaičiuoti o_k.

Apskaičiuojant ADLGT darbui su sveikaisiais skaičiais naudojamos formulės

$$o_k^{(i-1)} = s_k^{(i)} - [\frac{1}{4}\left(d_{k-1}^{(i)} + d_k^{(i)}\right) + \frac{1}{2}], e_k^{(i-1)} = d_k^{(i)} + [\frac{1}{2}\left(o_k^{(i-1)} + o_{k+1}^{(i-1)}\right)], \quad \textbf{(1.22)}$$

su visais $k = 0,1, \dots, 2^{n-i} - 1$ ir $i \in \{1,2, \dots, n\}$. Kaip ir anksčiau, norint apskaičiuoti e_k, pirma reikia apskaičiuoti o_k. Programa MATLAB parašytas algoritmas pateiktas 1 priede.

Algoritmas2 /Atvirkštinė modifikuota DLGT/

Kintamajam i nuosekliai priskiriamos reikšmės $n, n-1, \dots, 2,1$. Remiantis vektoriais $S^{(i)} = (s_0^{(i)} s_1^{(i)} \dots s_{2^{n-i}-1}^{(i)})^T$ ir $D^{(i)} = (d_0^{(i)} d_1^{(i)} \dots d_{2^{n-i}-1}^{(i)})^T$, atkuriamas vektorius $S^{(i-1)} = (s_0^{(i-1)} s_1^{(i-1)} \dots s_{2^{n-i+1}-1}^{(i-1)})^T = (o_0^{(i-1)} e_0^{(i-1)} o_1^{(i-1)} e_1^{(i-1)} \dots o_{2^{n-i}-1}^{(i-1)} e_{2^{n-i}-1}^{(i-1)})^T$.

1. $i \coloneqq n$.
2. Su kiekviena reikšme $k = 0,1, \dots, 2^{n-i} - 1$ apskaičiuojamos skaitinės vektoriaus $S^{(i-1)}$ reikšmės

$o_k^{(i-1)} = s_k^{(i)} - [\frac{1}{4}\left(d_{k-1}^{(i)} + d_k^{(i)}\right) + \frac{1}{2}]$,

$e_k^{(i-1)} = d_k^{(i)} + \left[\frac{1}{2}\left(o_k^{(i-1)} + o_{k+1}^{(i-1)}\right)\right]$.

3. Jei $i > 0$, tai $i := i - 1$ ir pereiti į 2 punktą.

4. Pabaiga. Atkurtas pradinis duomenų vektorius (vaizdas) $X = (x_0 x_1 \ \dots \ x_{2^n-1})^T = S^{(0)} = (o_0^{(0)} e_0^{(0)} o_1^{(0)} e_1^{(0)} \ \dots \ o_{N/2-1}^{(0)} e_{N/2-1}^{(0)})^T$.

Pastaruoju metu vis didesnį specialistų dėmesį atkreipia progresyvusis skaitmeninių vaizdų kodavimas. Jo esmė tame, kad panaudojant papildomus išsaugotos (apie pradinį vaizdą) informacijos bitus, galima nuosekliai gerinti atkurto signalo (vaizdo) arba jo fragmentų kokybę.

1.3 SKAITMENINIŲ VAIZDŲ SUGLAUDINIMO ALGORITMAS SPIHT: PRIVALUMAI IR TRŪKUMAI

SPIHT algoritmas buvo pristatytas 1995 metais. Jis taikomas efektyviam skaitmeninių vaizdų glaudinimui. Kalbant apie SPIHT algoritmą galima būtų išskirti tokias tris pagrindines dalis, dėl kurių jis taip puikiai tinka skaitmeninių vaizdų glaudinimui:

- spektrinių koeficientų rūšiavimas ir perdavimas pagal dydį;
- papildomų informacijos bitų, skirtų jau perduotoms reikšmėms patikslinti, generavimas;
- DBT panašumo savybės panaudojimas.

Tarkime turime skaitmeninį vaizdą, kurį aprašo pikselių matrica $X(i,j)(i,j \in \{0, N-1\})$. Pritaikome DBT dvimačiam vaizdui *X* ir gauname spektrinių koeficientų matricą *Y*, kuri irgi yra tokio paties dydžio kaip ir *X*. Progresyvaus duomenų perdavimo metu, rekonstruojamos reikšmės $\hat{Y}$ prilyginamos nuliui ir atkuriamos priklausomai nuo koduotos informacijos. Baigus dekodavimą pritaikoma atvirkštinę DBT ir gaunamas skaitmeninio vaizdo įvertis $\hat{X}$.

Pagrindinis progresyvaus duomenų perdavimo tikslas yra svarbios informacijos perdavimas pirmiau nesvarbios.

Atkurto vaizdo kokybę galima įvertinti pasinaudojant VKP,

$$VKP = \frac{1}{N^2}\sum_i \sum_j \left(X(i,j) - \hat{X}(i,j)\right)^2 \quad \textbf{(1.23)}$$

Iš (1.23) akivaizdu, jei $\hat{X}(i,j) = X(i,j)$, tai bendra paklaida sumažėja $\left(\frac{X(i,j)}{N}\right)^2$.Tai reiškia, kad koeficientai turintys dideles reikšmes turėtų būti perduodami pirmiau.

Kadangi SPIHT algoritmas generuoja dvejetainių kodų eilutę, tai pradžioje jis perduoda ne visą reikšmingo spektrinio koeficiento reikšmę, o tik svarbiausia jo bitą. Koeficientas laikomas reikšmingu, jei jo reikšmė absoliutiniu didumu nemažesnė už tam tikrą slenksčio reikšmę, t.y.$|Y(i,j)| \geq 2^r$.Nuosekliai mažinant slenksčio reikšmę (sekančios iteracijos metu iki 2^{r-1}), bus perduodami nauji reikšmingų koeficientų (tie kurie patenka į intervalą $2^{r-1} \leq |Y(i,j)| < 2^r$) bitai ir patikslinami praeitame etape laikyti reikšmingais koeficientai, išvedant $(r-1)$-ąjį reikšmingiausią jų bitą.

Svarbu pastebėti tai, kad SPIHT algoritmo duomenų rikiavimo rezultatai nėra perduodami. Kodavimo algoritmas nepertvarko spektrinių koeficientų vietų, o tiesiog atrenka tuos, kurie laikomi reikšmingais nuosekliai mažinant r reikšmes. Kadangi kodavimo ir dekodavimo algoritmų struktūros yra panašios, tai duomenų vietą erdvėje galima atkurti pasinaudojus kodavimo metu atliktais ir išsaugotais reikšmingumo tikrinimo rezultatais.

Tiesioginis pavienių spektrinių koeficientų tikrinimas yra labai neefektyvus laikinių sąnaudų ir duomenų perdavimo požiūriu. Šiai procedūrai pagerinti naudojamos nereikšmingos aibės.

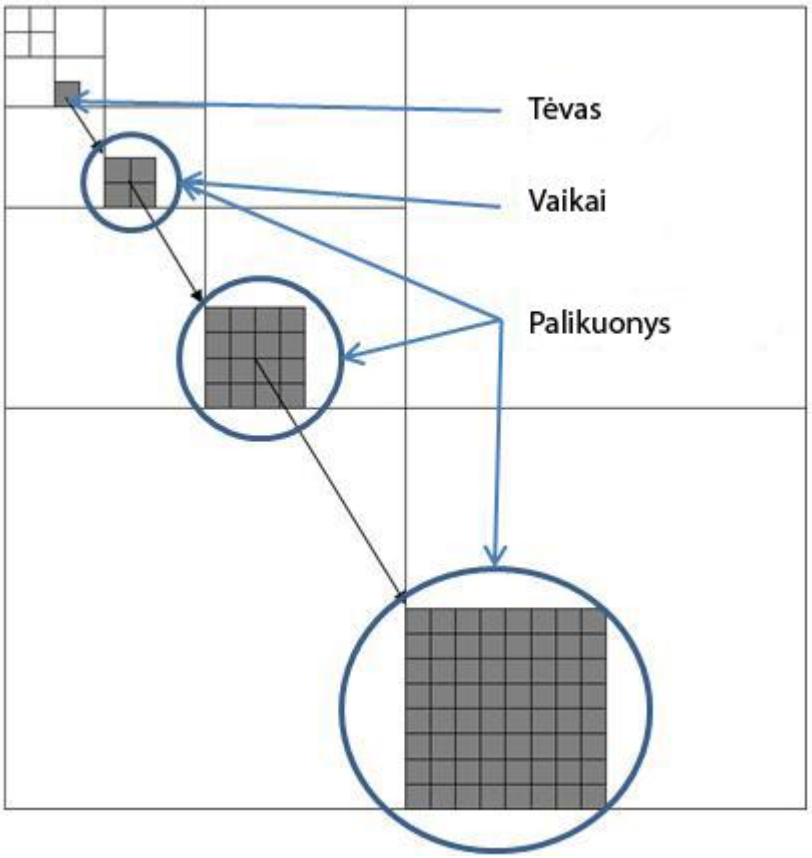
Pikselių matricą daliname į nepersidengiančias aibes T_m ir tikriname jų reikšmingumą tokiu būdu:

$$\max_{(i,j)\in T_m}\{|Y(i,j)|\} \geq 2^r. \qquad \textbf{(1.24)}$$

Spektrinių koeficientų aibę laikome nereikšminga, jei visi ją sudarantys koeficientai mažesni už slenksčio reikšmę. Jei randame, kad aibėje yra reikšmingas elementas, tai pagal tam tikrą taisyklę daliname ją į poaibius ir kiekvienam jų atskirai tikriname reikšmingumą. Tęsiame šį procesą, kol "išdaliname" reikšmingą aibę į nereikšmingas. Dabar iškyla klausimas, kokiu būdu suskirstyti spektrinių koeficientų matricą į aibes, kad kuo didesnę jos dalį būtų galima užkoduoti pasinaudojant nereikšmingomis aibėmis. Šioje vietoje panaudojama DBT panašumo savybė, t.y. aukštesnio lygio DBT spektriniai koeficientai yra panašūs į žemesnio lygio. Kvad-medžių struktūros puikiai tinka tokiam aibių suskirstymui.

Kvad-medis, tai tokia struktūrą DBT spektrinėje srityje, kurios kiekvienas narys, išskyrus paskutinio lygio bangelių koeficientus ir aukščiausio lygio bangelių koeficientą, turi keturis tiesioginius palikuonis. Aukščiausiame kvad-medžio lygyje esantis spektrinis koeficientas vadinami tėvu arba šaknimi, tiesioginiai jo palikuonys – vaikais, o likę kvad-medžio nariai tiesiog palikuonimis. Kvad-medį, kurį sudarantys spektriniai koeficientai yra mažesni už tam tikrą slenksčio reikšmę, vadiname nuliniu medžiu.

Kvad-medžio vaizdo diskrečiajame bangelių spektre pavyzdys pateiktas 1.1 pav.



1.1 pav. Kvad-medžio struktūra

Įveskime tokius pažymėjimus:

$O(i,j)$: kvad-medžio struktūroje apibrėžiama kaip visų tiesioginių palikuonių (vaikų), priklausančių tėvo reikšmei (i,j), koordinatės;

$D(i,j)$: visų palikuonių, priklausančių tėvui (i,j), koordinatės;

$L(i,j)$: $D(i,j) - O(i,j)$, visų palikuonių išskyrus vaikus koordinatės;

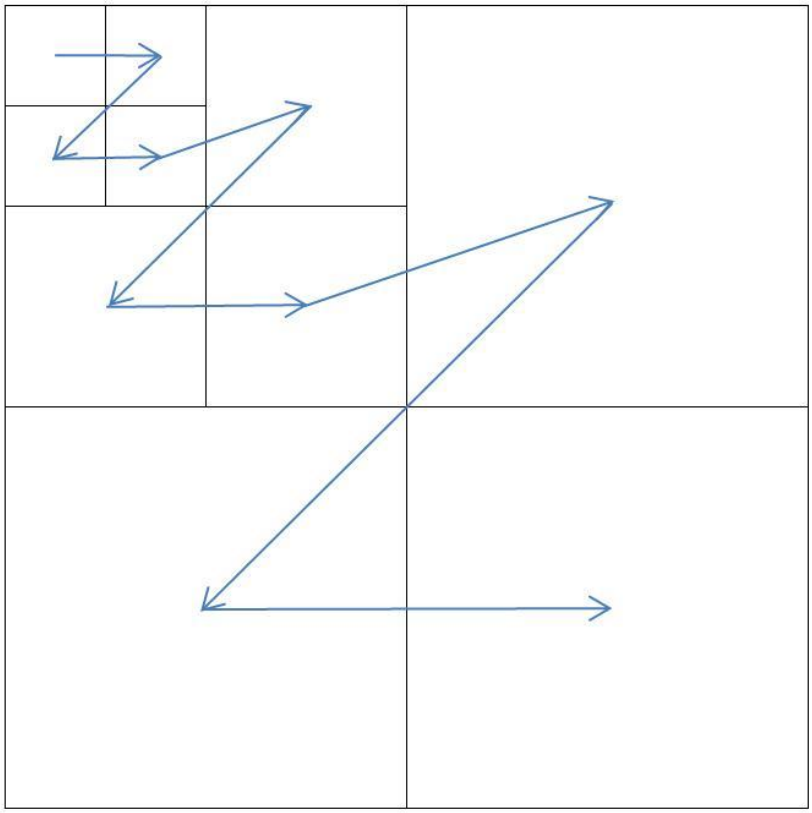
Pavyzdžiui, jei tėvo koordinatės (i,j), tai visas jo vaikų koordinatės galima aprašyti taip:

$O(i,j) = \{(2i, 2j), (2i, 2j+1),\ (2i+1, 2j), (2i+1, 2j+1)\}$;

čia

$i,j \in \left\{0,1,\dots,\frac{N}{2}-1\right\}, (i,j) \neq (0,0)$.

Spektrinių koeficientų skenavimo tvarka yra svarbi, nes tėvų reikšmės turi būti praskenuotos pirmiau palikuonių. Praktikoje dažnai naudojama Morton‘o skenavimo trajektorija, kurios schema pateikta 1.2 pav.



1.2 Morton‘o trajektorija

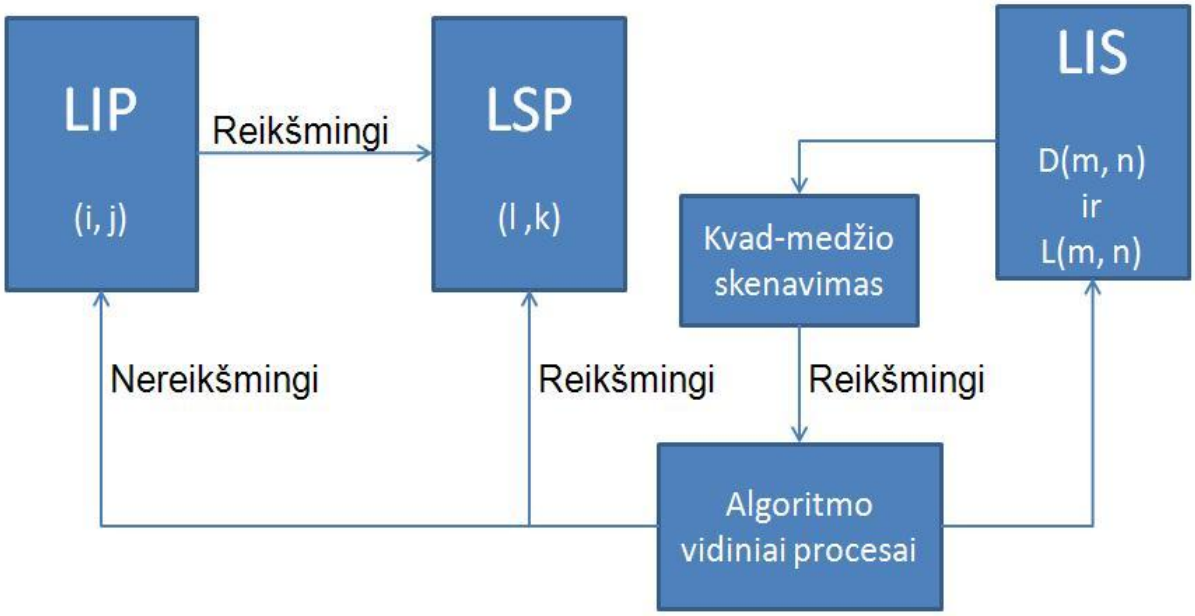
Įveskime pažymėjimus:

LIS (list of insignificant sets)– nereikšmingų aibių sąrašas: jame saugomos bangelių transformacijos spektrinių koeficientų koordinatės, kurios apibrėžia nulinių-medžių struktūras. Šioje aibėje nesaugomi koeficientai, kurie priklauso medžiui, išskyrus tėvus (šaknis). Visi koeficientai turi bent keturis palikuonis.

LIP (list of insignificant pixels) – nereikšmingų pikselių sąrašas: jame saugomos tų spektrinių koeficientų koordinatės, kurie absoliutiniu didumu yra mažesni už slenksčio reikšmę.

LSP (list of significant pixels) – reikšmingų pikselių sąrašas: jame saugomos tų spektrinių koeficientų koordinatės, kurie absoliutiniu didumu yra didesni už slenksčio reikšmę.

Koduodamas ir dekoduodamas vaizdą SPIHT algoritmas generuoja tris aukščiau išvardintus pagalbinius sąrašus, kurie nurodo reikšmingų ir nereikšmingų elementų vaizdo diskrečiajame bangelių spektre išsidėstymą. *LIP*, *LSP* ir *LIS* duomenų srautų schema pateikta 1.3 pav.



1.3 SPIHT algoritmo duomenų srautai

Žemiau pateikiamas bazinis SPIHT skaičiavimo algoritmas.

Reikšmingumo tikrinimui įvedame funkciją

$$S_r(T) = \begin{cases} 1, \max_{(i,j)\in T}\{|Y(i,j)|\} \geq 2^r, \\ 0, \qquad\qquad kitu\ atveju. \end{cases} \tag{1.25}$$

Algoritmas /Bazinis SPIHT algoritmas/

1. Priskiriami pradiniai duomenys.
   Išvedama $r = \lfloor log_2(\max|Y(i,j)|) \rfloor$
   ir priskiriamos pradinės reikšmės sąrašams *LIP*, *LSP*, *LIS*:
   $LIP = ((0;0),(0;1),(1;0),(1,1))$;
   $LSP = (\,)$;
   $LIS = ((0;1),(1;0),(1,1))$.
2. Generuojama bitų eilutė išvedant S_r ir $Y(i,j)$ ženklo reikšmes.
   2.1. Kiekvienam įrašui (i,j) priklausančiam sąrašui *LIP* atliekame:
      2.1.1. Išvedame $S_r((i,j))$;
      2.1.2. Jei $S_r((i,j)) = 1$, perkeliame (i,j) į *LSP* ir išvedame $Y(i,j)$ ženklą;
   2.2. Kiekvienam elementui (i,j) priklausančiam *LIS* atliekame:
      2.2.1. Jei elemento tipas *A*

Išvedame $S_r(D(i,j))$;

Jei $S_r(D(i,j)) = 1$, tai:

Kiekvienam $(k,l) \in O(i,j)$ atliekame:

Išvedame $S_r((k,l))$;

Jei $S_r((k,l)) = 1$, tai perkeliame (k,l) į *LSP* ir išvedame jo ženklą;

Jei $S_r((k,l)) = 0$, tai perkeliame (k,l) į *LIP*;

Jei $L(i,j) \neq \emptyset$, tai perkeliame (i,j) į *LIS* galą, kaip *B* tipo įrašą ir pereiname į 2.2.2 punktą; kitu atveju, pašalinti (i,j) iš *LIS*;

2.2.2. Jei elemento tipas *B*

Išvedame $S_r(L(i,j))$;

Jei $S_r(L(i,j)) = 1$, tai:

Perkeliame visus $(k,l) \in O(i,j)$ į LIS sąrašo galą, kaip *A* tipo įrašus;

Pašaliname (i,j) iš *LIS*.

3. Patikslinamе elementų įeinančių į *LSP*, išskyrus tuos, kuriuos pridėjome paskutinėje iteracijoje, reikšmes, išvesdami *r*-tąjį elemento $Y(i,j)$ bitą.
4. Sumažiname $r := r - 1$. Jei $r > 0$ pereiname į 2 punktą.
5. Pabaiga.

SPIHT algoritmo programinė realizacija Matlab programa pateikta 1 priede.

Praktikoje pastebime, kad SPIHT algoritmo kodavimo metu atliekama nulinių medžių paieška beveik dvigubai prailgina veikimo laiką. Remiantis šiuo pastebėjimu, buvo pasiūlyta pagerinta nulinių medžių paieškos schema.

2 PAGERINTOS NULINIŲ MEDŽIŲ PAIEŠKOS SHEMA SPIHT ALGORITME

Kodavimo metu SPIHT algoritmas ieško nulinių medžių, atlikdamas kvad-medžių skenavimą. Kažkurioje kvad-medžio pozicijoje radus reikšmingą spektrinį koeficientą, pradinis kvad-medis dalinamas į keturis mažesnius ir tuomet tikrinamas naujų kvad-medžių reikšmingumas. Tokiu būdu akivaizdu, kad tie patys spektriniai koeficientai tikrinami daugiau nei vieną kartą. Siekiant išvengti pakartotinio DBT spektrinių koeficientų skenavimo, buvo pasiūlytas naujas metodas, kaip efektyviai

išgauti visą reikalingą informaciją apie kvad-medį sudarančių palikuonių reikšmingumą. Jo įgyvendinimo procedūrą pateikiame žemiau.

Procedūra /Nulinių medžių paieškos schema/

Skaitmeniniam vaizdui $[X(m_1, m_2)]$ $(m_1, m_2 \in \{0, 1, \dots, N-1\}, N = 2^n, n \in \mathrm{N})$ pritaikome DBT ir gauname spektrinių koeficientų matricą $[Y(k_1, k_2)]$ $(k_1, k_2 \in \{0, 1, \dots, N-1\}$.

1. Apskaičiuojamas $r = r_{max} = \lfloor \log_2 \max\{|Y(k_1, k_2)|\} \rfloor$; čia $[Y(k_1, k_2)]$ yra spektrinių koeficientų matrica, gauta vaizdui $[X(m_1, m_2)]$ pritaikius diskrečiąją bangelių transformaciją; $(k_1, k_2,\ m_1, m_2 \in \{0, 1, \dots, N-1\}, N = 2^n, n \in \mathrm{N}$.

2. Formuojama dvejetainių kodų matrica, kurios elementai $< u_r(k_1, k_2), v_r(k_1, k_2) > (k_1, k_2, \in \{0, 1, \dots, N/2 - 1\})$ nurodo kiekvieno bangelių koeficiento $Y(k_1, k_2)$ reikšmingumą slenksčio $T_r = 2^r$ $(r \in \{0, 1, \dots, r_{max}\})$ atžvilgiu, būtent:

$u_r(k_1, k_2) = 1$, jei bent vienas iš koeficientų $|Y(2k_1,\ 2k_2)|$, $|Y(2k_1,\ 2k_2 + 1)|$, $|Y(2k_1 + 1,\ 2k_2)|$, $|Y(2k_1 + 1, 2k_2 + 1)|$ patenka į intervalą $[2^r, 2^r + 1)$, ir $u_r(k_1, k_2) = 0$, kitu atveju.

$$v_r(k_1, k_2) = \begin{cases} u_r(2k_1,\ 2k_2) \vee u_r(2k_1,\ 2k_2 + 1) \vee u_r(2k_1 + 1,\ 2k_2) \vee \\ \qquad u_r(2k_1 + 1,\ 2k_2 + 1), \qquad kai\ \dfrac{N}{8} \le \max\{\mathrm{k}_1, \mathrm{k}_2\} \le N/4 - 1, \\ u_r(2k_1,\ 2k_2) \vee v_r(2k_1,\ 2k_2) \vee u_r(2k_1,\ 2k_2 + 1) \vee \\ \qquad v_r(2k_1,\ 2k_2 + 1) \vee u_r(2k_1 + 1,\ 2k_2) \vee v_r(2k_1 + 1,\ 2k_2) \vee \\ \qquad u_r(2k_1 + 1,\ 2k_2 + 1) \vee v_r(2k_1 + 1,\ 2k_2 + 1), \\ \qquad kai\ \ 1 \le \max\{\mathrm{k}_1, \mathrm{k}_2\} \le N/8 - 1 \end{cases}$$

Gautieji kodai nurodo pozicijos, susijusios su koeficientu (šaknimi) $Y(k_1, k_2)$, palikuonių reikšmingumą, t.y. jei $u_r(k_1, k_2) = 0$ ir $v_r(k_1, k_2) = 0$, tai $Y(k_1, k_2)$ yra nulinio medžio šaknis, esant slenksčiui 2^r, t.y. visi kvad-medį sudarantys koeficientai yra nereikšmingi. Tolimesnis kvad-medžio skenavimas tampa nereikalingu.

Kadangi SPIHT algoritmas „nutraukiamas“ pasiekus norimą vaizdo suglaudinimo lygį, tai bendru atveju, $r = r^* > 0$, ir dvejetainių kodų pozicijos $u_r(k_1, k_2)$ ir $v_r(k_1, k_2)$ $(0 \le r \le r^*)$ išvis nėra analizuojamos. Tokiu būdu, atsiranda galimybė dar pagerinti SPIHT algoritmo efektyvumą – ne iš karto, o nuosekliai (su kiekviena nauja slenksčio T_r reikšme) formuojant anksčiau minėtas dvejetainių kodų pozicijas.

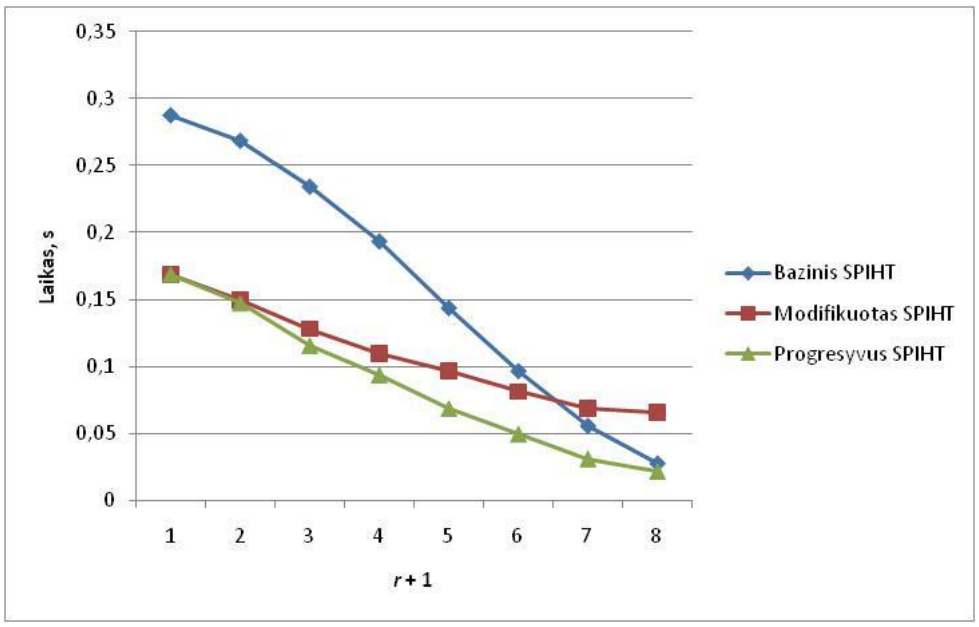
3 EKSPERIMENTŲ REZULTATAI – PALYGINAMOJI DVIEJŲ SPIHT ALGORITMŲ (BAZINIO IR MODIFIKUOTO) EFEKTYVUMO ANALIZĖ

Antro skyriaus teorinę medžiagą pritaikome praktiškai ir atliekame tyrimus.

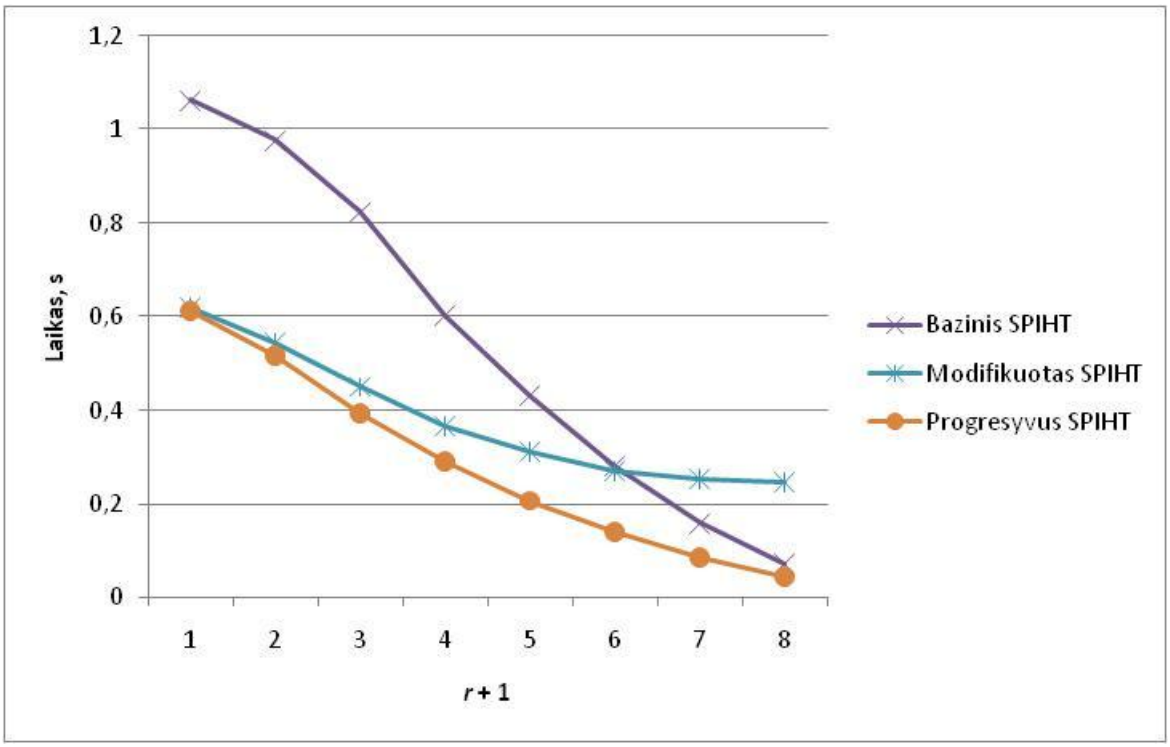
Parenkame įvairių dydžių (128x128, 256x256, 512x521)skaitmeninį vaizdą *Lena.bmp* (3.1 pav.):

3.1 pav. Skaitmeninis vaizdas *Lena.bmp*

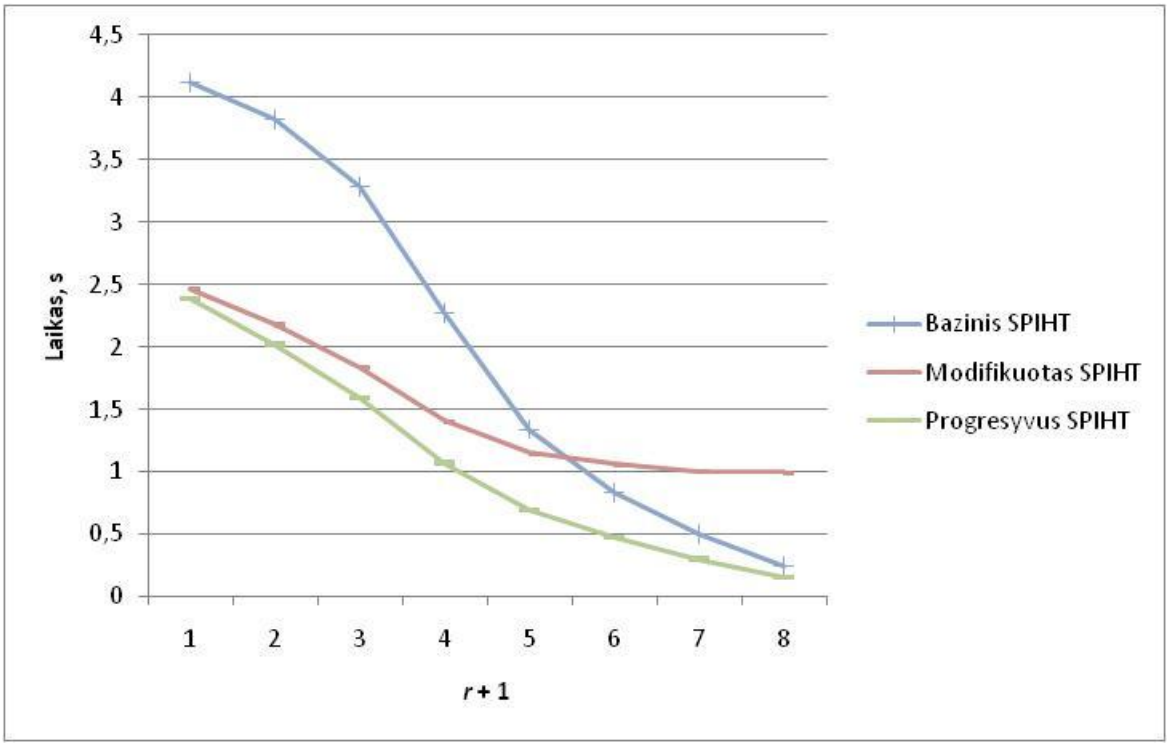
Pritaikome bazinį ir modifikuotą SPIHT algoritmus. Mažindami slenksčio reikšmę, fiksuojame algoritmų veikimo laikus. Gauti rezultatai pavaizduoti 3.2 – 3.4 pav.



3.2 pav. SPIHT algoritmo veikimo laikai vaizdui *Lena.bmp* (128x128)

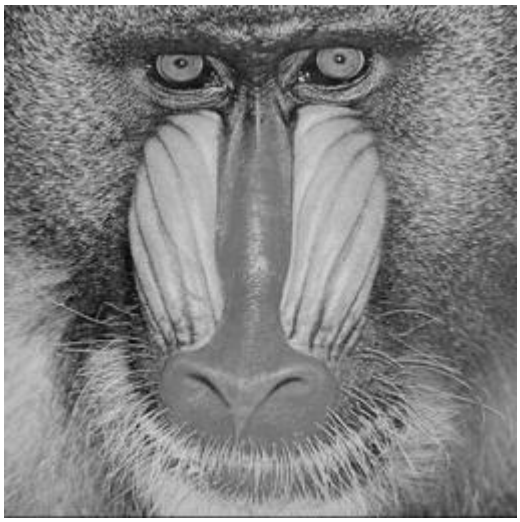


3.3 pav. SPIHT algoritmo veikimo laikai vaizdui *Lena.bmp* (256x256)



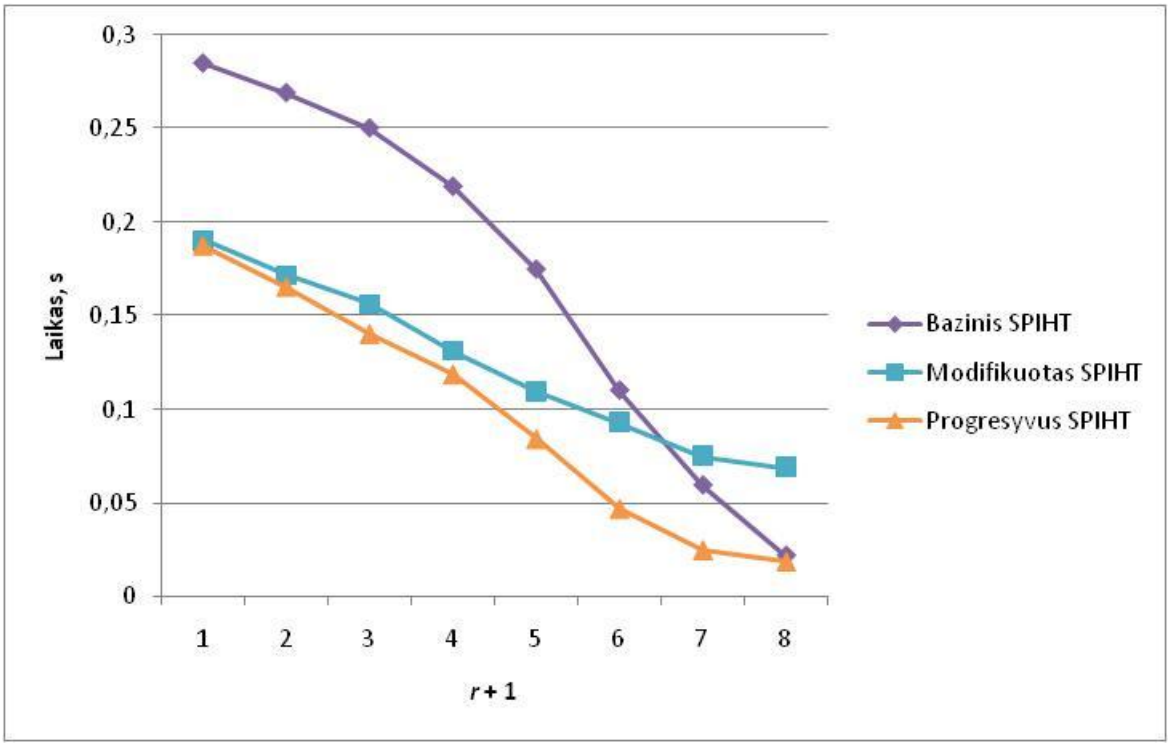
3.4 pav. SPIHT algoritmo veikimo laikai vaizdui *Lena.bmp* (512x512)

Parenkame įvairių dydžių (128x128, 256x256, 512x521) skaitmeninį vaizdą *Baboon.bmp* (3.5pav.) ir jam pritaikome bazinį bei modifikuotus SPIHT algoritmus. Veikimo laikai, priklausantys nuo slenksčio reikšmės, pateikiami 3.6 – 3.8 pav.

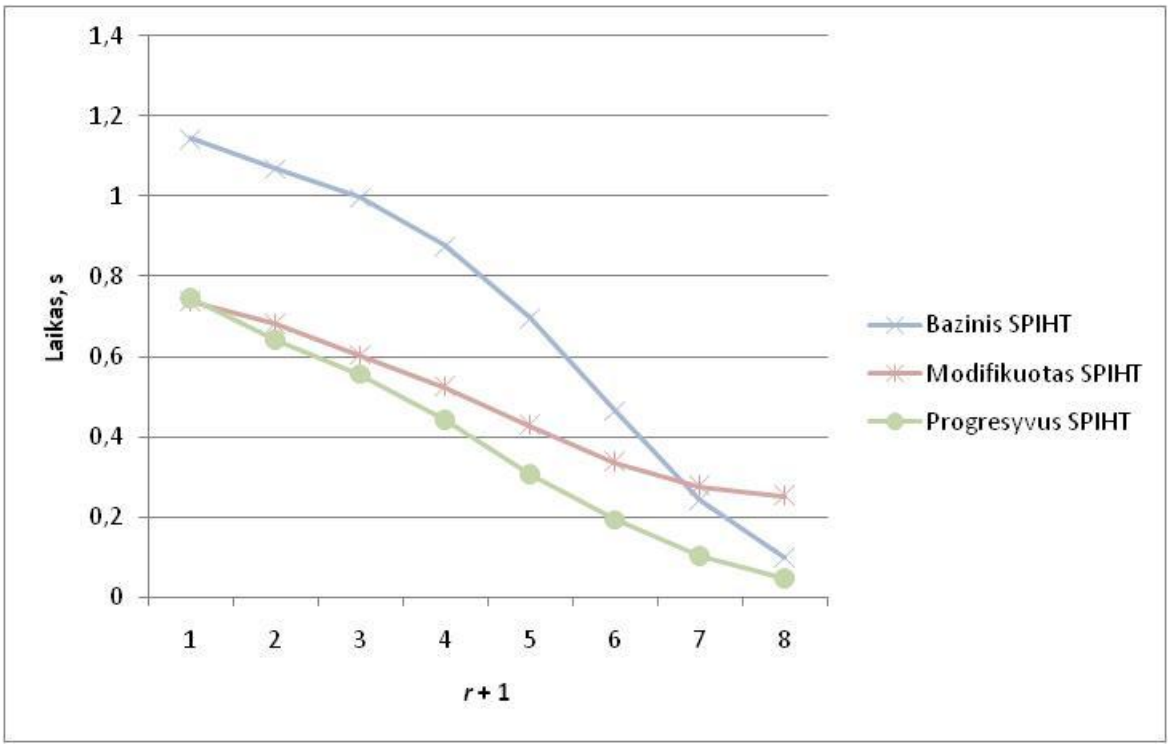


3.5 pav. Skaitmeninis vaizdas *Baboon.bmp*

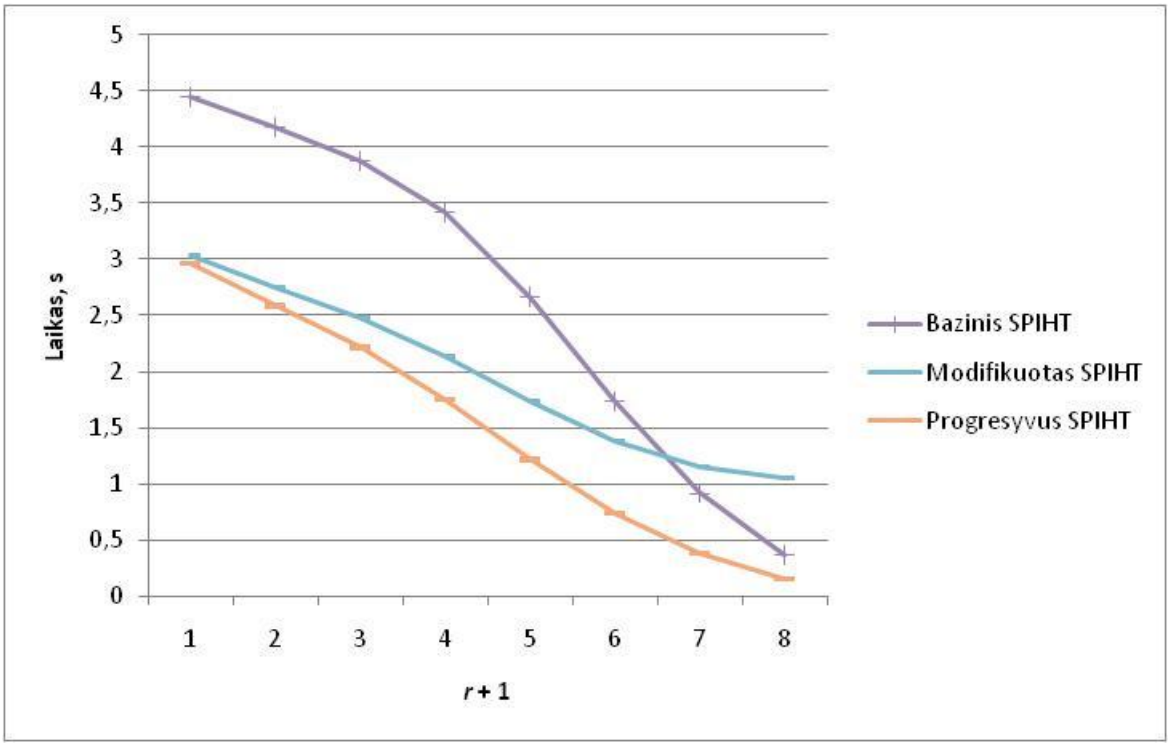
Pritaikome bazinį ir modifikuotą SPIHT algoritmą. Gauti rezultatai pavaizduoti 3.6 – 3.8 pav.



3.6 pav. SPIHT algoritmo veikimo laikai vaizdui *Baboon.bmp* (128x128)



3.7 pav. SPIHT algoritmo veikimo laikai vaizdui *Baboon.bmp* (256x256)



3.8 pav. SPIHT algoritmo veikimo laikai vaizdui *Baboon.bmp* (512x512)

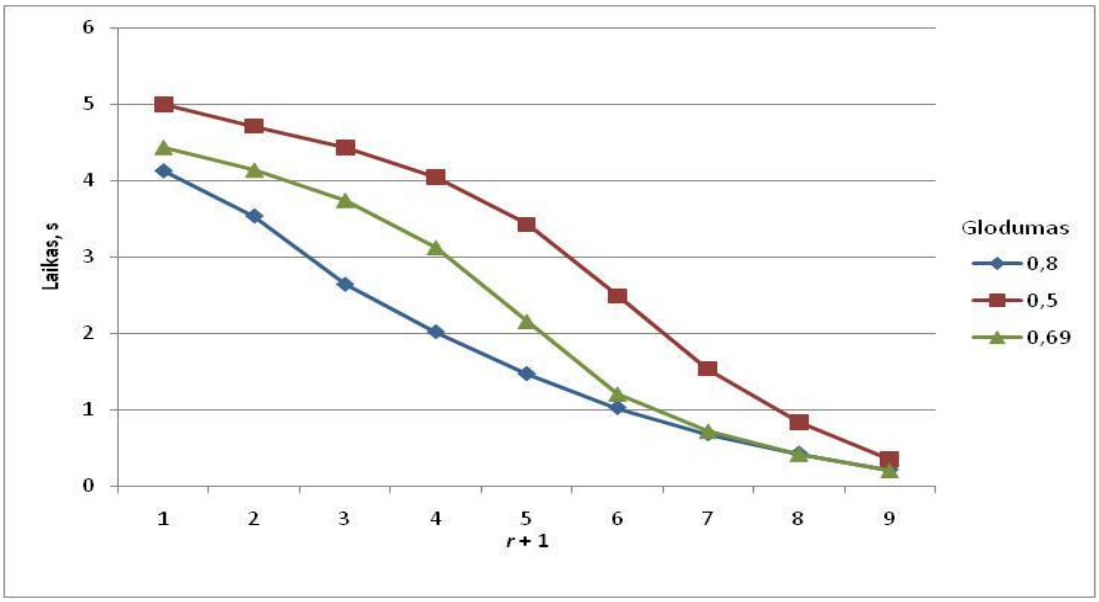
Kaip matome iš 3.2 – 3.4 ir 3.6 – 3.8 pav. apdorodamas skaitmeninius vaizdus mažiausiai laiko sugaišo SPIHT algoritmas su pagerinta nulinių medžių vaizdo diskrečiajame bangelių spektre paieškos schema, kai matrica, nurodanti palikuonių reikšmingumą, generuojama progresyviai.

Žemiau pateikta skaitmeninių vaizdų glodumo parametrų lentelė (3.1 lentelė).

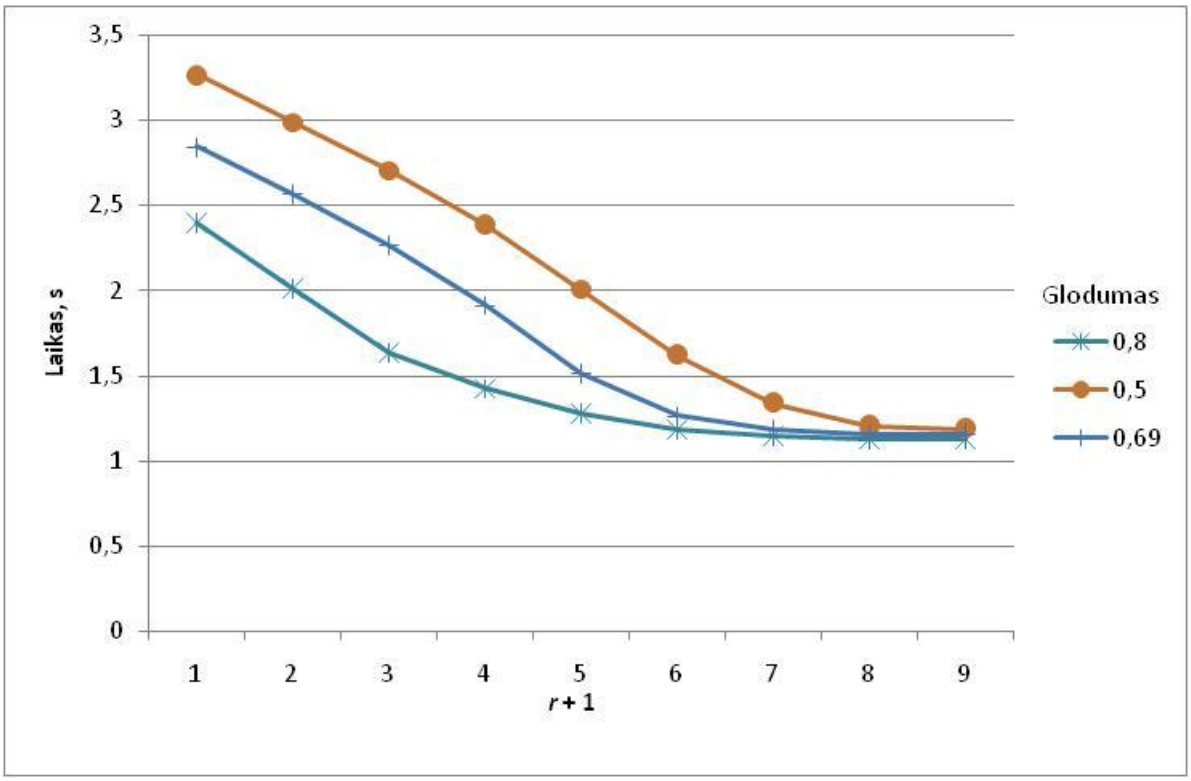
3.1 lentelė. Skaitmeninių vaizdų glodumas

Paveiksliukas	Dydis	Glodumas
Baboon	128	0,57
Baboon	256	0,54
Baboon	512	0,51
Cameraman	128	0,76
Cameraman	256	0,77
Cameraman	512	0,8
Lena	128	0,63
Lena	256	0,66
Lena	512	0,68
Mountain	128	0,65
Mountain	256	0,56
Mountain	512	0,5
Nature	128	0,65
Nature	256	0,67
Nature	512	0,69

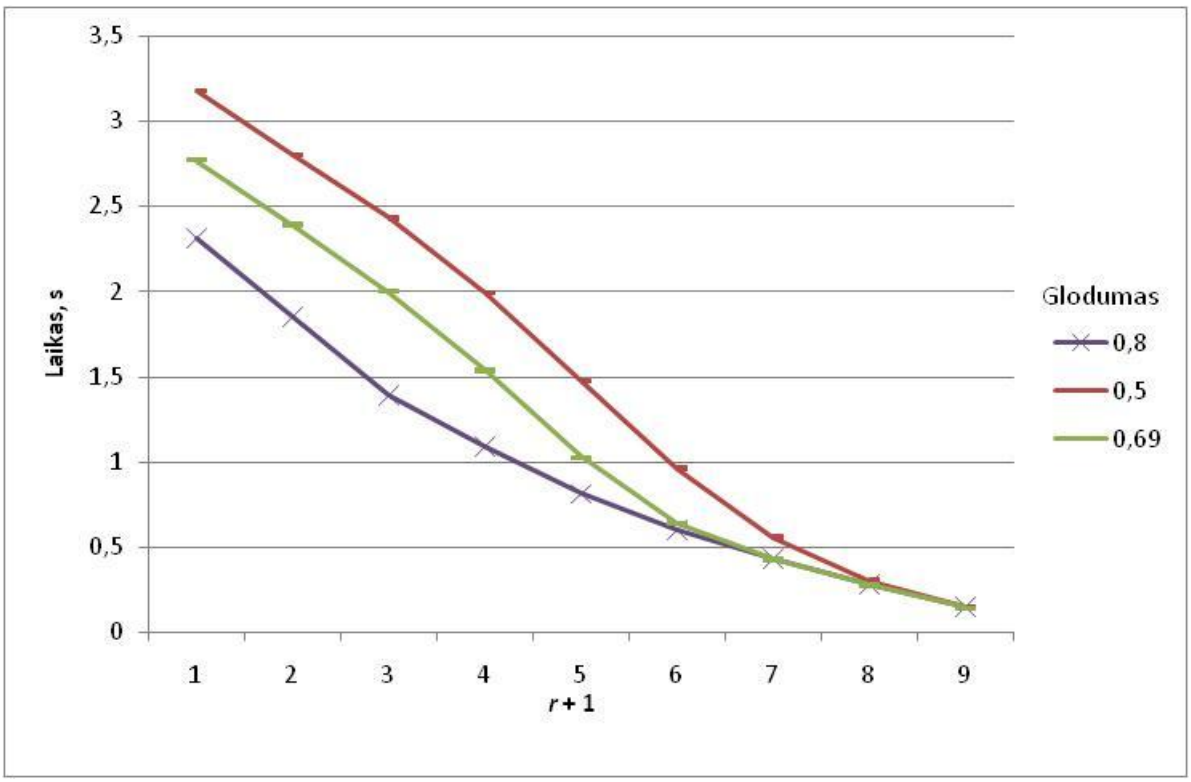
Pasinaudoję 3.1 lentele, pateikiame bazinio ir modifikuotų SPIHT algoritmų veikimo laikų grafikus, kuriuose lyginami skirtingas glodumo reikšmes turintys skaitmeniniai vaizdai.



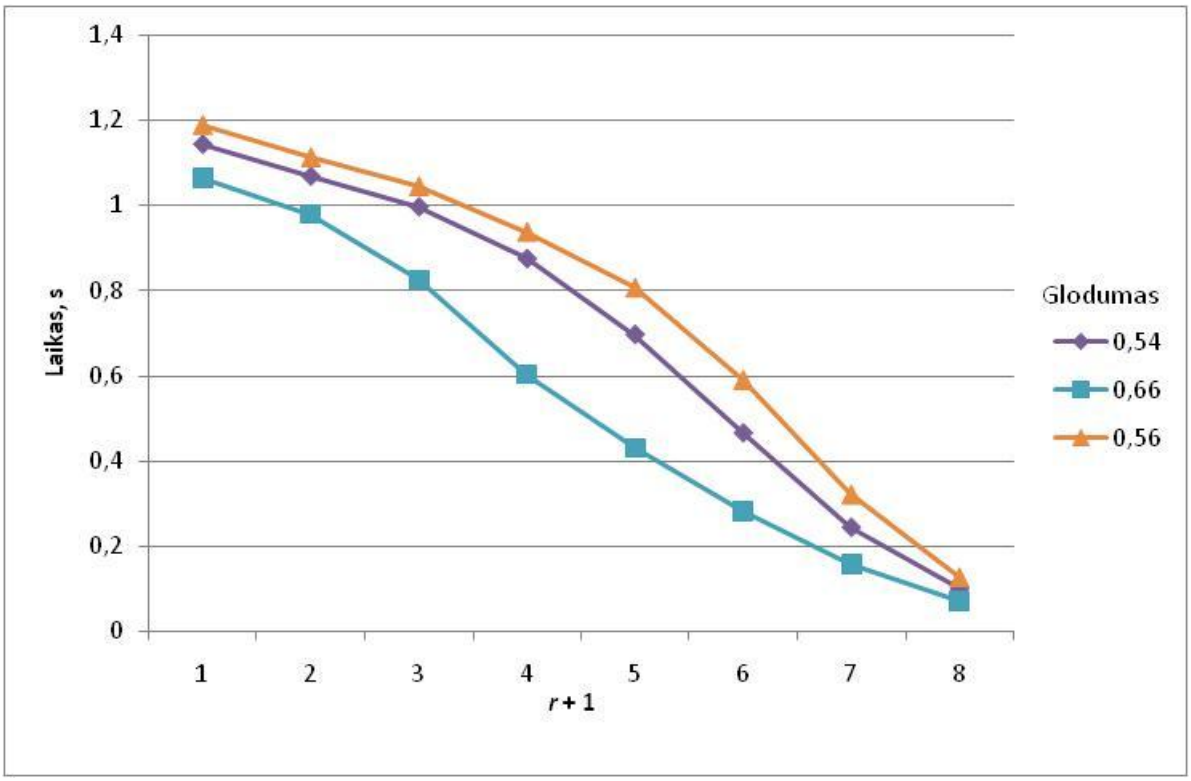
3.9 pav. Bazinio SPIHT algoritmo veikimo laikai vaizdams *Cameraman.bmp*, *Mountain.bmp* ir *Nature.bmp* (512x512)



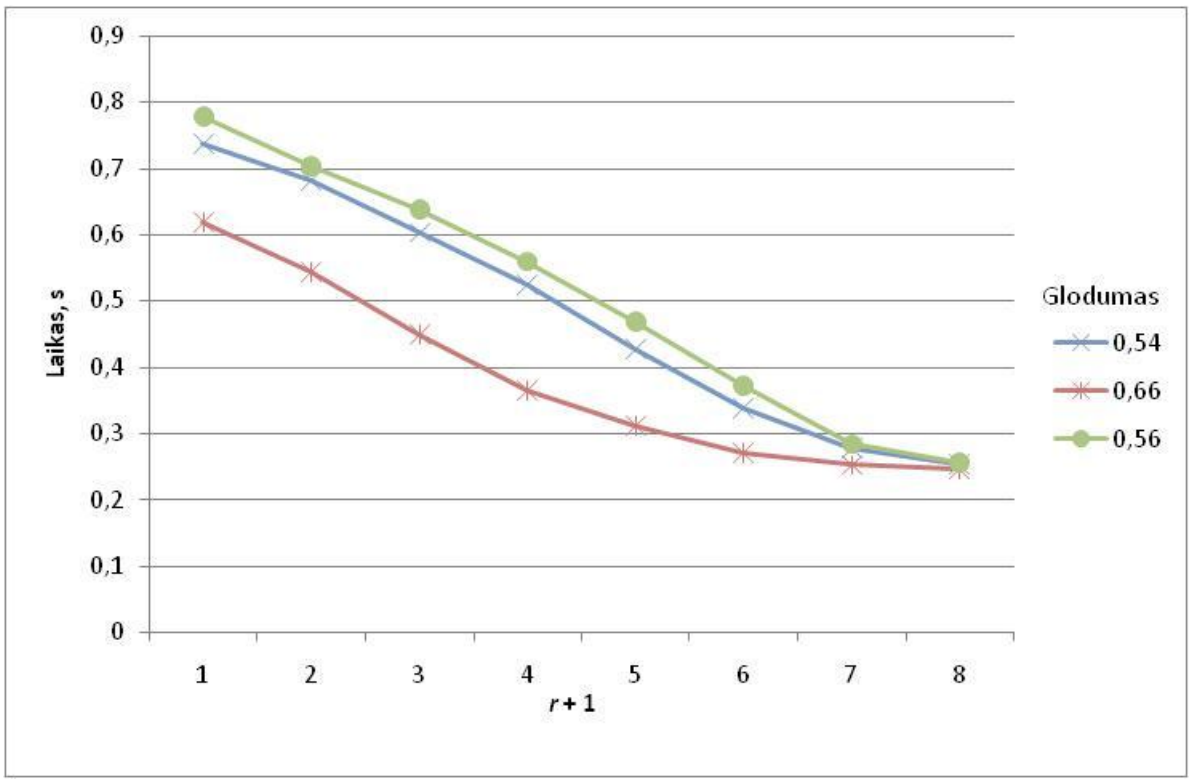
3.10 pav. Modifikuoto SPIHT algoritmo, kai kodai formuojami nuosekliai, veikimo laikai vaizdams *Cameraman.bmp*, *Mountain.bmp* ir *Nature.bmp* (512x512)



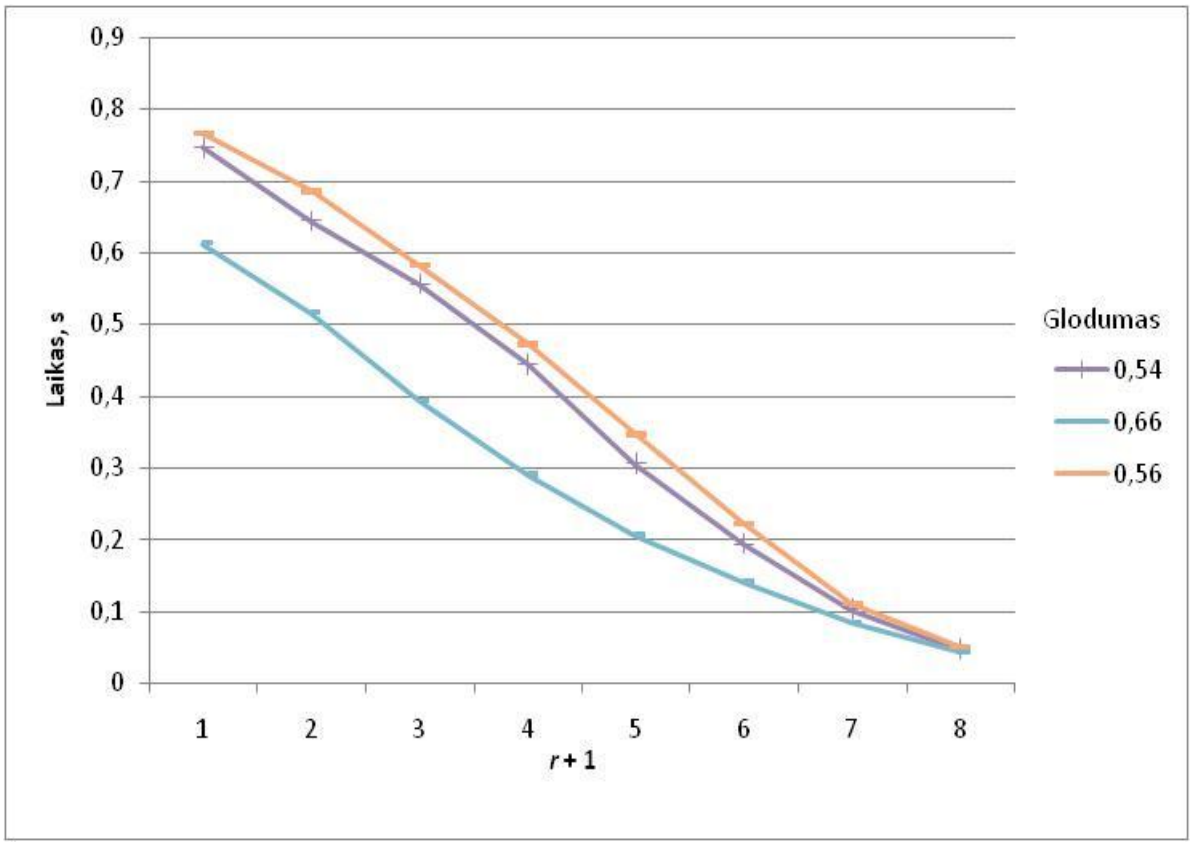
3.11 pav. Modifikuoto SPIHT algoritmo, kai kodai formuojami progresyviai, veikimo laikai, vaizdams *Cameraman.bmp*, *Mountain.bmp* ir *Nature.bmp* (512x512)



3.12 Bazinio SPIHT algoritmo veikimo laikai vaizdams *Baboon.bmp, Lena.bmp* ir *Mountain.bmp* (256x256)



3.13 pav. Modifikuoto SPIHT algoritmo, kai kodai formuojami nuosekliai, veikimo laikai vaizdams *Baboon.bmp, Lena.bmp* ir *Mountain.bmp* (256x256)



3.14 pav. Modifikuoto SPIHT algoritmo, kai kodai formuojami progresyviai, veikimo laikai vaizdams *Baboon.bmp, Lena.bmp* ir *Mountain.bmp* (256x256)

Kaip matome iš 3.1 – 3.6 pav., esant mažiems skirtumams tarp skaitmeninių vaizdų glodumo parametrų reikšmių SPIHT algoritmo laikai beveik sutampa. Didėjant glodumo parametro reikšmei SPIHT algoritmo veikimo laikas trumpėja. Šį faktą galėtume paaiškinti tuo, kad prie didesnių glodumo parametrų reikšmių vaizdą sudarančios aukšto dažnio harmonikos „gęsta“ greičiau. Tokiu būdu, kvad-medžių analizei skiriama mažiau laiko, nes didesnė dalis spektrinių koeficientų yra nereikšmingi prie didelių slenksčio reikšmių.

Lyginame bazinį SPIHT algoritmą ir SPIHT algoritmą su pagerinta nulinių medžių paieškos schema, kai palikuonių reikšmingumo matrica generuojama progresyviai. Tarkime bazinio SPIHT veikimo laiką pasižymėkime τ_r, o pagerinto – τ_r^*. Apskaičiuojame procentinį laikinių sąnaudų išlošį. Tam pasinaudojame 1.26 formule:

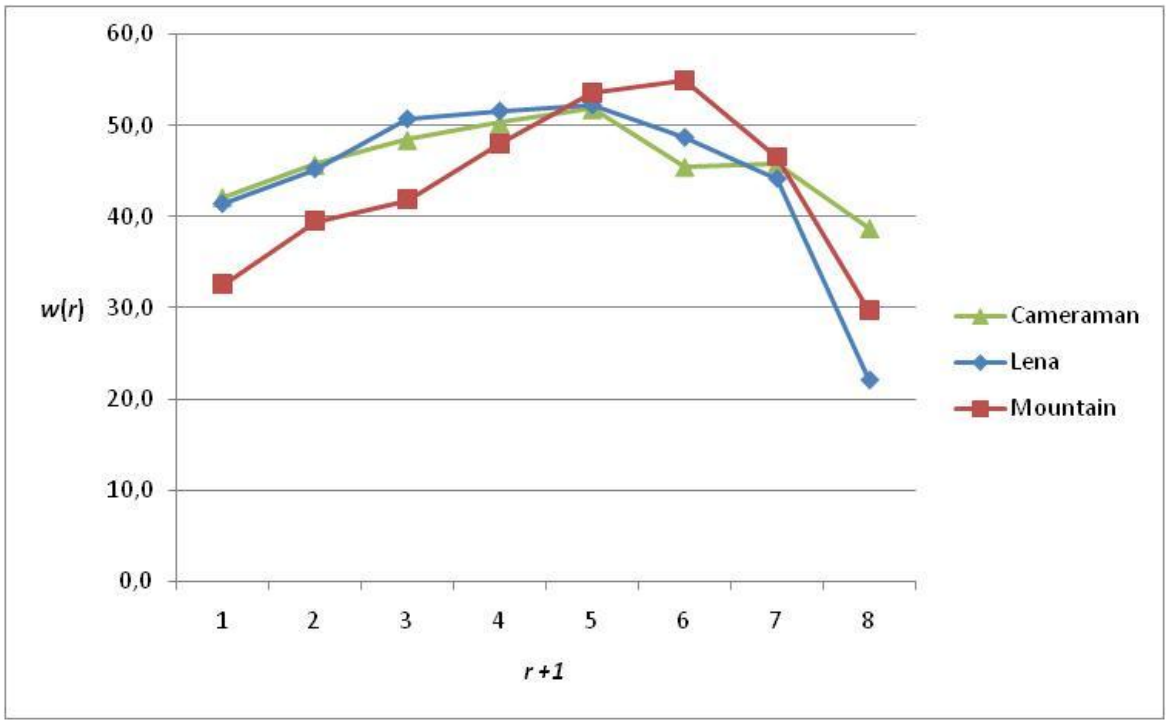
$$w_r = \frac{\tau_r - \tau_r^*}{\tau_r} 100\% \qquad \textbf{(1.26)}$$

Žemiau pateikiame laikinių sąnaudų išlošių lentelę.

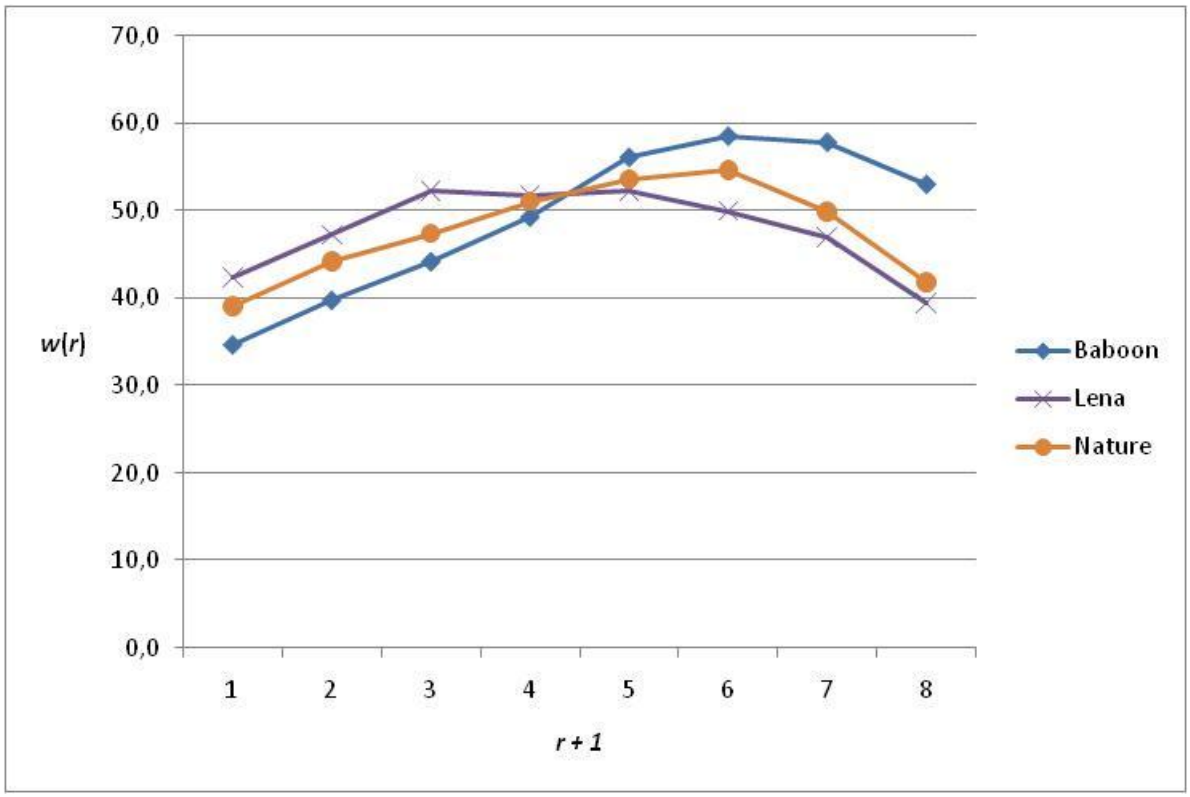
3.2 lentelė. Santykinis laikinių sąnaudų išlošis

Pavadinimas	Dydis	r = 0	r = 1	r = 2	r = 3	r = 4	r = 5	r = 6	r = 7	r = 8
Baboon	128	34,1	38,4	43,8	45,7	51,7	57,3	57,9	13,8	
Baboon	256	34,7	39,8	44,2	49,3	56,1	58,5	57,8	53,0	
Baboon	512	33,5	38,1	42,9	48,9	54,3	57,4	57,9	55,9	
Bridge	128	44,4	49,0	50,1	51,0	49,3	41,2	24,2	9,2	
Bridge	256	46,8	51,7	56,3	57,6	55,6	53,1	42,4		
Bridge	512	47,2	51,5	54,5	55,6	53,6	49,8	43,4	38,2	
Cameraman	128	42,2	45,7	48,4	50,2	51,8	45,5	45,8	38,7	
Cameraman	256	44,9	47,6	48,4	50,2	52,3	52,0	49,4	44,7	47,7
Cameraman	512	44,1	47,8	47,5	46,0	44,8	41,7	37,0	34,4	30,4
Lena	128	41,4	45,2	50,7	51,5	52,2	48,7	44,1	22,0	
Lena	256	42,4	47,2	52,3	51,8	52,2	50,0	47,0	39,4	
Lena	512	42,2	47,2	51,6	52,8	48,6	43,8	40,7	36,3	
Mountain	128	32,5	39,5	41,8	48,0	53,5	54,9	46,5	29,6	
Mountain	256	35,6	38,5	44,3	49,7	57,1	62,4	65,9	61,0	
Mountain	512	36,6	40,7	45,2	50,7	57,1	61,4	63,7	63,5	57,1
Nature	128	36,3	41,1	47,6	45,6	53,3	54,6	50,8	35,4	
Nature	256	39,0	44,2	47,4	51,1	53,6	54,7	49,9	41,8	
Nature	512	37,5	42,2	46,5	50,7	52,2	46,7	39,9	32,8	27,6

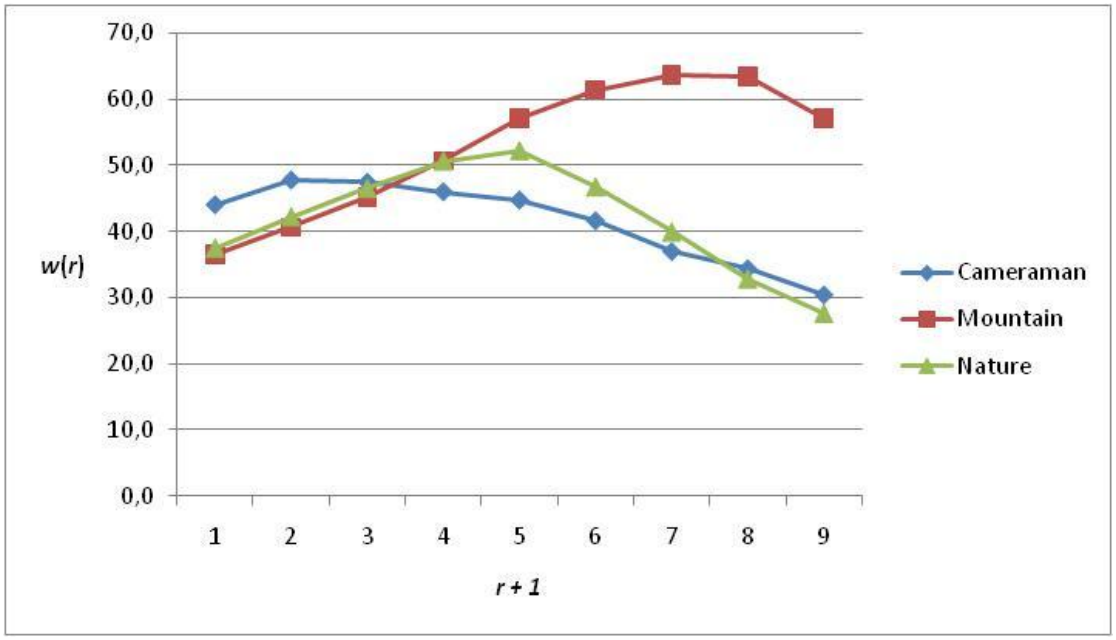
Pasinaudoję 3.2 lentelėje pateiktais duomenimis braižome grafikus, kuriuose mažindami slenkščio reikšmes, atidedame procentinį laikinių sąnaudų išlošį.



3.15 pav. Santykinis veikimo laiko išlošis vaizdams, kurių dydis 128x128.



3.16 pav. Santykinis veikimo laiko išlošis vaizdams, kurių dydis 256x256.



3.17 pav. Santykinis veikimo laiko išlošis vaizdams, kurių dydis 512x512.

Iš 3.15 – 3.17 pav. matome, kad santykinis išlošis siekia 30% ir daugiau. Prie didelių slenksčio reikšmių negalime daryti tikslių išvadų, nes tyrimo rezultatus labai įtakoja kompiuterinio skaičiavimo paklaidos.

IŠVADOS

1. Išsami skaitmeninių vaizdų suglaudinimo algoritmo SPIHT analizė parodė, jog didesnė laikinių sąnaudų dalis (apie 60%) realizuojant algoritmą tenka nulinių medžių diskrečiajame apdorojamo vaizdo DBT spektre analizei. Šis faktas mažina bendrą SPIHT algoritmo efektyvumą.

2. Darbe pasiūlytos dvi pagerintos nulinių medžių paieškos vaizdo DBT spektre schemos (nuosekli ir progresyvi) leidžia reikšmingai pagerinti SPIHT algoritmo efektyvumą (apie 46%). Kitaip tariant, santykinis išlošis nepriklauso nuo apdorojamo vaizdo matmenų.

3. Ištyrus gaunamų laikinių sąnaudų (realizuojant modifikuotą SPIHT algoritmą) išlošį, buvo pastebėta, kad jis priklauso ir nuo apdorojamo skaitmeninio vaizdo glodumo, t.y. nuo to, kaip greitai „gęsta“ vaizdą sudarančios aukšto dažnio harmonikos. Kuo glodesnis vaizdas, tuo didesnis laikinių sąnaudų išlošis.

4. Pasiūlyta pagerinta nulinių medžių paieškos schema galėtu būti pilnai pritaikyta bet kuriam nulinių medžių kodavimo pagrindu „dirbančiam“ vaizdų suglaudinimo algoritmui (tarkim EZW).

LITERATŪRA

1. Jonas Valantinas. Diskrečiosios transformacijos (mokomoji knyga), Technologija, 2008.
2. Patrick J. Van Fleet. Wavelets and lossless compression in the jpeg2000 image compression standard (mokomoji priemonė), University of St. Thomas, 2006.
3. J. M. Shapiro. Embedded image coding using zero-trees of wavelet coefficients. IEEE Trans. Signal Processing, vol. 41: 3445 – 3462, 1993.
4. A. Said and W. A. Pearlman. A new, fast and efficient image codec based on set partitioning in hierarchical trees. IEEE Transactions on Circuits Systems for Video Technology, vol. 6, 1996, 243-250.
5. J. Valantinas and D. Kančelkis. Improving Compression Time in Zero-Tree Based Image Coding Procedures. 2nd International Conference on Image Processing Theory, Tools and Applications (IPTA 2010), Paris, France, 2010, 118 – 121.

1 PRIEDAS. PROGRAMŲ TEKSTAI

Programa Matalb parašyta pagrindinės programos algoritmas:

```
function Pagrindine()

duom = double(imread('bridge256.bmp'));
spektras = MDLGT(duom);
i = floor(log2(max(max(abs(spektras)))));
A = zeros(1,i);
B = A;
C = A;
for t = 0:i
   AA = zeros(1,5);
   BB = AA;
   CC = AA;
   for tt = 1:5
      tic
      SPIHT_enc(spektras, 512*512*8,t);
      AA(tt) = toc;
      tic
      SPIHT_enc2(spektras, 512*512*8,t);
      BB(tt) = toc;
      tic
      SPIHT_enc3(spektras, 512*512*8,t);
      CC(tt) = toc;
   end
   A(t+1) = mean(AA);
   B(t+1) = mean(BB);
   C(t+1) = mean(CC);
end
% AA = SPIHT_dec(length(duom),floor(log2(max(max(abs(spektras))))),bitai);
% rezultatas = MADLGT(AA);
% imwrite(rezultatas,gray(256),'rez.bmp','bmp');
hold on
axis([0 i 0 max([A B C])+0.1])
plot(0:i,A,'LineWidth',2)
plot(0:i,B,'LineWidth',2,'Color','g')
plot(0:i,C,'LineWidth',2,'Color','r')
legend('bazinis SPIHT', 'modifikuotas SPIHT', 'progresyvus SPIHT')
xlabel('r')
ylabel('Laikas, s')
```

Programa Matlab parašytas DLGT algoritmas:

```
function spektras = DLGT(duom)

X = duom;
Y = zeros(size(X,1),size(X,2));
n = log2(size(X,1));
o = zeros(size(X,1)/2);
e = zeros(size(X,1)/2);
d = zeros(size(X,1));
for m = 1:2^n
   s = X(:,m);
   for i = 1:n
      for j = 1:2^(n-i)
         o(j) = s(2*j-1);
         e(j) = s(2*j);
      end
      for k = 1:2^(n-i)
         if k == 2^(n-i)
            d(k) = e(k) - o(k);
         else
            d(k) = e(k) - (o(k)+o(k+1))/2;
         end
      end
      for k = 1:2^(n-i)
         if k == 1
            s(k) = o(k) + d(k)/2;
         else
            s(k) = o(k) + (d(k-1)+d(k))/4;
         end
      end
      for j = 1:2^(n-i)
         Y(2^(n-i)+j,m) = d(j);
      end
      Y(1,m) = s(1);
   end
end
for m = 1:2^n
   s = Y(m,:);
   for i = 1:n
      for j = 1:2^(n-i)
         o(j) = s(2*j-1);
         e(j) = s(2*j);
      end
      for k = 1:2^(n-i)
         if k == 2^(n-i)
            d(k) = e(k) - o(k);
         else
            d(k) = e(k) - (o(k)+o(k+1))/2;
         end
      end
```

```
        for k = 1:2^(n-i)
            if k == 1
                s(k) = o(k) + d(k)/2;
            else
                s(k) = o(k) + (d(k-1)+d(k))/4;
            end
        end
        for j = 1:2^(n-i)
            Y(m,2^(n-i)+j) = d(j);
        end
        Y(m,1) = s(1);
    end
end
spektras = Y;
```

Programa Matlab parašytas atvirkštinės DLGT algoritmas:

```
function X = ADLGT(spektras)

Y = 100*spektras;
X = zeros(size(Y,1),size(Y,2));
n = log2(size(Y,1));
o = zeros(size(Y,1)/2);
e = zeros(size(Y,1)/2);
d = zeros(size(Y,1));
for m = 1:2^n
    s(1) = Y(m,1);
    for i = n:-1:1
        for j = 1:2^(n-i)
            d(j) = Y(m,2^(n-i)+j);
        end
        for k = 1:2^(n-i)
            if k == 1
                o(k) = s(k) - d(k)/2;
            else
                o(k) = s(k) - (d(k)+d(k-1))/4;
            end
        end
        for k = 1:2^(n-i)
            if k == 2^(n-i)
                e(k) = d(k) + o(k);
            else
                e(k) = d(k) + (o(k)+o(k+1))/2;
            end
        end
        for j = 1:2^(n-i)
            s(2*j-1) = o(j);
            s(2*j) = e(j);
        end
```

```
    end
    Y(m,:) = s;
end
for m = 1:2^n
    s(1) = Y(1,m);
    for i = n:-1:1
        for j = 1:2^(n-i)
            d(j) = Y(2^(n-i)+j,m);
        end
        for k = 1:2^(n-i)
            if k == 1
                o(k) = s(k) - d(k)/2;
            else
                o(k) = s(k) - (d(k)+d(k-1))/4;
            end
        end
        for k = 1:2^(n-i)
            if k == 2^(n-i)
                e(k) = d(k) + o(k);
            else
                e(k) = d(k) + (o(k)+o(k+1))/2;
            end
        end
        for j = 1:2^(n-i)
            s(2*j-1) = o(j);
            s(2*j) = e(j);
        end
    end
    X(:,m) = s;
end
for m = 1:2^n
    for j = 1:2^n
        if X(m,j) > 256
            X(m,j) = 256;
        else
            if X(m,j) < 1
                X(m,j) = 1;
            else
                X(m,j) = round(X(m,j));
            end
        end
    end
end
```

Programa Matlab parašytas modifikuotos DLGT algoritmas:

```
function spektras = MDLGT(duom)

X = duom;
Y = zeros(size(X,1),size(X,2));
n = log2(size(X,1));
o = zeros(size(X,1)/2);
e = zeros(size(X,1)/2);
d = zeros(size(X,1));
for m = 1:2^n
   s = X(:,m);
   for i = 1:n
      for j = 1:2^(n-i)
         o(j) = s(2*j-1);
         e(j) = s(2*j);
      end
      for k = 1:2^(n-i)
         if k == 2^(n-i)
            d(k) = e(k) - floor(o(k));
         else
            d(k) = e(k) - floor((o(k)+o(k+1))/2);
         end
      end
      for k = 1:2^(n-i)
         if k == 1
            s(k) = o(k) + floor(d(k)/2 + 1/2);
         else
            s(k) = o(k) + floor((d(k-1)+d(k))/4 + 1/2);
         end
      end
      for j = 1:2^(n-i)
         Y(2^(n-i)+j,m) = d(j);
      end
      Y(1,m) = s(1);
   end
end
for m = 1:2^n
   s = Y(m,:);
   for i = 1:n
      for j = 1:2^(n-i)
         o(j) = s(2*j-1);
         e(j) = s(2*j);
      end
      for k = 1:2^(n-i)
         if k == 2^(n-i)
            d(k) = e(k) - floor(o(k));
         else
            d(k) = e(k) - floor((o(k)+o(k+1))/2);
         end
      end
```

```
        for k = 1:2^(n-i)
            if k == 1
                s(k) = o(k) + floor(d(k)/2 + 1/2);
            else
                s(k) = o(k) + floor((d(k-1)+d(k))/4 + 1/2);
            end
        end
        for j = 1:2^(n-i)
            Y(m,2^(n-i)+j) = d(j);
        end
        Y(m,1) = s(1);
    end
end
spektras = Y;
```

Programa Matlab parašytas modifikuotos atvirkštinės DLGT algoritmas:

```
function X = MADLGT(spektras)

Y = spektras;
X = zeros(size(Y,1),size(Y,2));
n = log2(size(Y,1));
o = zeros(size(Y,1)/2);
e = zeros(size(Y,1)/2);
d = zeros(size(Y,1));
for m = 1:2^n
    s(1) = Y(m,1);
    for i = n:-1:1
        for j = 1:2^(n-i)
            d(j) = Y(m,2^(n-i)+j);
        end
        for k = 1:2^(n-i)
            if k == 1
                o(k) = s(k) - floor(d(k)/2 + 1/2);
            else
                o(k) = s(k) - floor((d(k)+d(k-1))/4 + 1/2);
            end
        end
        for k = 1:2^(n-i)
            if k == 2^(n-i)
                e(k) = d(k) + floor(o(k));
            else
                e(k) = d(k) + floor((o(k)+o(k+1))/2);
            end
        end
        for j = 1:2^(n-i)
            s(2*j-1) = o(j);
            s(2*j) = e(j);
        end
```

```
    end
    Y(m,:) = s;
end
for m = 1:2^n
    s(1) = Y(1,m);
    for i = n:-1:1
        for j = 1:2^(n-i)
            d(j) = Y(2^(n-i)+j,m);
        end
        for k = 1:2^(n-i)
            if k == 1
                o(k) = s(k) - floor(d(k)/2 + 1/2);
            else
                o(k) = s(k) - floor((d(k)+d(k-1))/4 + 1/2);
            end
        end
        for k = 1:2^(n-i)
            if k == 2^(n-i)
                e(k) = d(k) + floor(o(k));
            else
                e(k) = d(k) + floor((o(k)+o(k+1))/2);
            end
        end
        for j = 1:2^(n-i)
            s(2*j-1) = o(j);
            s(2*j) = e(j);
        end
    end
    X(:,m) = s;
end
for m = 1:2^n
    for j = 1:2^n
        if X(m,j) > 256
            X(m,j) = 256;
        else
            if X(m,j) < 1
                X(m,j) = 1;
            else
                X(m,j) = round(X(m,j));
            end
        end
    end
end
```

Programa Matlab parašytas Bazinio SPIHT kodavimo algoritmas:

```
function bitai = SPIHT_enc(duom, maxbitai,tttt)

T = floor(log2(max(max(abs(duom)))));
LIP = zeros(size(duom,1)*size(duom,2),2);
LIP(1:4,:) = [1 1; 1 2; 2 1; 2 2];
LSP = zeros(size(duom,1)*size(duom,2),2);
LIS = zeros(size(duom,1)*size(duom,2),3);
LIS(1:3,:) = [1 2 1; 2 1 1; 2 2 1];
bitai = zeros(1,maxbitai);
k = 1;
nr_LSP = 1;
nr_LIP = 4;
nr_LIS = 3;
while k <= maxbitai
   for n = 1:nr_LIP
      if LIP(n,1) ~= 0
         if abs(duom(LIP(n,1),LIP(n,2))) >= 2^T
            bitai(k) = 1;
            k = k + 1;
            LSP(nr_LSP,:) = [LIP(n,1) LIP(n,2)];
            nr_LSP = nr_LSP + 1;
            if sign(duom(LIP(n,1),LIP(n,2))) > 0
               bitai(k) = 1;
            else
               bitai(k) = 0;
            end
            k = k + 1;
            LIP(n,1) = 0;
         else
            bitai(k) = 0;
            k = k + 1;
         end
      end
   end
   n = 1;
   while n <= nr_LIS
      if LIS(n,3) == 1
         i = LIS(n,1); j = LIS(n,2);
         S = Tikrinimas(i,j,T,duom,1);
         if S == 1
            bitai(k) = 1;
            k = k + 1;
            i = LIS(n,1); j = LIS(n,2);
            temp = [2*i-1 2*j-1; 2*i-1 2*j; 2*i 2*j-1; 2*i 2*j];
            for mm = 1:4
               if abs(duom(temp(mm,1),temp(mm,2))) >= 2^T
                  bitai(k) = 1;
                  k = k + 1;
                  LSP(nr_LSP,:) = [temp(mm,1) temp(mm,2)];
```

```
                nr_LSP = nr_LSP + 1;
                if sign(duom(temp(mm,1),temp(mm,2)))>0
                   bitai(k) = 1;
                else
                   bitai(k) = 0;
                end
                k = k + 1;
             else
                bitai(k) = 0;
                k = k + 1;
                nr_LIP = nr_LIP + 1;
                LIP(nr_LIP,:) = [temp(mm,1) temp(mm,2)];
             end
          end
          if i*4 <= size(duom,1) && j*4 <= size(duom,2)
             nr_LIS = nr_LIS + 1;
             LIS(nr_LIS,:) = [LIS(n,1:2) 0];
          end
          LIS(n,3) = 2;
       else
          bitai(k) = 0;
          k = k + 1;
       end
    else
       if LIS(n,3) == 0
          i = LIS(n,1); j = LIS(n,2);
          S = Tikrinimas(i,j,T,duom,0);
          if S == 1
             bitai(k) = 1;
             k = k + 1;
             nr_LIS = nr_LIS + 4;
             LIS(nr_LIS-3:nr_LIS,:) = [2*LIS(n,1)-1 2*LIS(n,2)-1 1;
                2*LIS(n,1)-1 2*LIS(n,2) 1;
                2*LIS(n,1) 2*LIS(n,2)-1 1;
                2*LIS(n,1) 2*LIS(n,2) 1];
             LIS(n,3) = 2;
          else
             bitai(k) = 0;
             k = k + 1;
          end
       end
    end
    n = n + 1;
end

for i = 1:nr_LSP-1
    if abs(duom(LSP(i,1),LSP(i,2))) >= 2^(T+1)
       bitai(k) = bitget(abs(round(duom(LSP(i,1),LSP(i,2)))),T+1);
       k = k + 1;
    end
```

```
    end
    T = T-1;
    if T < 0+tttt
        bitai = bitai(1,1:k);
        break
    end
end

function S = Tikrinimas(i,j,T,duom,tipas)

S = 0;
laipsnis = log2(size(duom,1)) - floor(log2(max(i,j)));
dydis = (4^(laipsnis+1) - 1)/3;
temp = zeros(dydis, 2);
if tipas == 1
    temp(1,:) = [i j];
    nr_temp = 1;
else
    temp(1:4,:) = [2*i-1 2*j-1; 2*i-1 2*j; 2*i 2*j-1; 2*i 2*j];
    nr_temp = 4;
end
n = 1;
while n<=nr_temp
    i = temp(n,1);
    j = temp(n,2);
    if abs(duom(2*i-1,2*j-1)) >= 2^T || ...
            abs(duom(2*i-1,2*j)) >= 2^T || ...
            abs(duom(2*i,2*j-1)) >= 2^T || ...
            abs(duom(2*i,2*j)) >= 2^T
        S = 1;
        break
    end
    if i <= size(duom,1)/4 && j <= size(duom,2)/4
        nr_temp = nr_temp + 4;
        temp(nr_temp - 3:nr_temp, :) = [ 2*i-1 2*j-1; 2*i-1 2*j; 2*i 2*j-1; 2*i 2*j];
    end
    n = n + 1;
end
```

Programa Matlab parašytas modifikuoto SPIHT algoritmo, kai priklausomumo matrica generuojama nuosekliai, kodas:

```
function bitai = SPIHT_enc2(duom, maxbitai,tttt)

T = floor(log2(max(max(abs(duom)))));
v = Tikrinimas2(abs(duom),T);
LIP = zeros(size(duom,1)*size(duom,2),2);
LIP(1:4,:) = [1 1; 1 2; 2 1; 2 2];
LSP = zeros(size(duom,1)*size(duom,2),2);
LIS = zeros(size(duom,1)*size(duom,2),3);
LIS(1:3,:) = [1 2 1; 2 1 1; 2 2 1];
bitai = zeros(1,maxbitai);
k = 1;
nr_LSP = 1;
nr_LIP = 4;
nr_LIS = 3;
while k <= maxbitai
   for n = 1:nr_LIP
      if LIP(n,1) ~= 0
         if abs(duom(LIP(n,1),LIP(n,2))) >= 2^T
            bitai(k) = 1;
            k = k + 1;
            LSP(nr_LSP,:) = [LIP(n,1) LIP(n,2)];
            nr_LSP = nr_LSP + 1;
            if sign(duom(LIP(n,1),LIP(n,2))) > 0
               bitai(k) = 1;
            else
               bitai(k) = 0;
            end
            k = k + 1;
            LIP(n,1) = 0;
         else
            bitai(k) = 0;
            k = k + 1;
         end
      end
   end
   n = 1;
   while n <= nr_LIS
      if LIS(n,3) == 1
         if v(LIS(n,1),LIS(n,2),T+1) == 1
            bitai(k) = 1;
            k = k + 1;
            i = LIS(n,1); j = LIS(n,2);
            temp = [2*i-1 2*j-1; 2*i-1 2*j; 2*i 2*j-1; 2*i 2*j];
            for mm = 1:4
               if abs(duom(temp(mm,1),temp(mm,2))) >= 2^T
                  bitai(k) = 1;
                  k = k + 1;
```

```
                LSP(nr_LSP,:) = [temp(mm,1) temp(mm,2)];
                nr_LSP = nr_LSP + 1;
                if sign(duom(temp(mm,1),temp(mm,2)))>0
                   bitai(k) = 1;
                else
                   bitai(k) = 0;
                end
                k = k + 1;
             else
                bitai(k) = 0;
                k = k + 1;
                nr_LIP = nr_LIP + 1;
                LIP(nr_LIP,:) = [temp(mm,1) temp(mm,2)];
             end
          end
          if i*4 <= size(duom,1) && j*4 <= size(duom,2)
             nr_LIS = nr_LIS + 1;
             LIS(nr_LIS,:) = [LIS(n,1:2) 0];
          end
          LIS(n,3) = 2;
       else
          bitai(k) = 0;
          k = k + 1;
       end
    else
       if LIS(n,3) == 0
          if v(LIS(n,1)*2-1,LIS(n,2)*2-1,T+1) == 1 ||...
                v(LIS(n,1)*2-1,LIS(n,2)*2,T+1) == 1 ||...
                v(LIS(n,1)*2,LIS(n,2)*2-1,T+1) == 1 ||...
                v(LIS(n,1)*2,LIS(n,2)*2,T+1) == 1
             bitai(k) = 1;
             k = k + 1;
             nr_LIS = nr_LIS + 4;
             LIS(nr_LIS-3:nr_LIS,:) = [2*LIS(n,1)-1 2*LIS(n,2)-1 1;
                2*LIS(n,1)-1 2*LIS(n,2) 1;
                2*LIS(n,1) 2*LIS(n,2)-1 1;
                2*LIS(n,1) 2*LIS(n,2) 1];
             LIS(n,3) = 2;
          else
             bitai(k) = 0;
             k = k + 1;
          end
       end
    end
    n = n + 1;
end
for i = 1:nr_LSP-1
    if abs(duom(LSP(i,1),LSP(i,2))) >= 2^(T+1)
       bitai(k) = bitget(abs(round(duom(LSP(i,1),LSP(i,2)))),T+1);
       k = k + 1;
```

```
            end
        end
        T = T-1;
        if T < 0+tttt
            bitai = bitai(1,1:k);
            break
        end
    end

    function v = Tikrinimas2(Y,T)

    u = zeros(size(Y,1)/2,size(Y,2)/2,T+1);
    v = zeros(size(Y,1)/2,size(Y,2)/2,T+1);
    N = size(Y,1);
    for k1 = N/2:-1:1
        for k2 = N/2:-1:1
            for r = 1:T+1
                if Y(k1*2-1,k2*2-1) >= 2^(r-1) && Y(k1*2-1,k2*2-1) < 2^r...
                        || Y(k1*2-1,k2*2) >= 2^(r-1) && Y(k1*2-1,k2*2) < 2^r...
                        || Y(k1*2,k2*2-1) >= 2^(r-1) && Y(k1*2,k2*2-1) < 2^r...
                        || Y(k1*2,k2*2) >= 2^(r-1) && Y(k1*2,k2*2) < 2^r
                    u(k1,k2,r) = 1;
                else
                    u(k1,k2,r) = 0;
                end
                if max(k1,k2) > N/4
                    v(k1,k2,r) = u(k1,k2,r);
                end
                if max(k1,k2) > N/8 && max(k1,k2) <= N/4
                    v(k1,k2,r) = bitor(u(k1,k2,r),bitor(bitor(u(k1*2-1,k2*2-1,r),u(k1*2-
1,k2*2,r)),bitor(u(k1*2,k2*2-1,r),u(k1*2,k2*2,r))));
                end
                if max(k1,k2) <= N/8
                    v(k1,k2,r) = bitor(u(k1,k2,r),bitor(bitor(bitor(u(k1*2-1,k2*2-1,r),u(k1*2-
1,k2*2,r)),bitor(u(k1*2,k2*2-1,r),u(k1*2,k2*2,r))),bitor(bitor(v(k1*2-1,k2*2-1,r),v(k1*2-
1,k2*2,r)),bitor(v(k1*2,k2*2-1,r),v(k1*2,k2*2,r)))));
                end
            end
        end
    end
```

Programa Matlab parašytas modifikuoto SPIHT algoritmo, kai priklausomumo matrica generuojama progresyviai, kodas:

```
function bitai = SPIHT_enc3(duom, maxbitai,tttt)

T = floor(log2(max(max(abs(duom)))));
LIP = zeros(size(duom,1)*size(duom,2),2);
LIP(1:4,:) = [1 1; 1 2; 2 1; 2 2];
LSP = zeros(size(duom,1)*size(duom,2),2);
LIS = zeros(size(duom,1)*size(duom,2),3);
LIS(1:3,:) = [1 2 1; 2 1 1; 2 2 1];
bitai = zeros(1,maxbitai);
k = 1;
nr_LSP = 1;
nr_LIP = 4;
nr_LIS = 3;
while k <= maxbitai
   v = Tikrinimas3(abs(duom),T);
   for n = 1:nr_LIP
      if LIP(n,1) ~= 0
         if abs(duom(LIP(n,1),LIP(n,2))) >= 2^T
            bitai(k) = 1;
            k = k + 1;
            LSP(nr_LSP,:) = [LIP(n,1) LIP(n,2)];
            nr_LSP = nr_LSP + 1;
            if sign(duom(LIP(n,1),LIP(n,2))) > 0
               bitai(k) = 1;
            else
               bitai(k) = 0;
            end
            k = k + 1;
            LIP(n,1) = 0;
         else
            bitai(k) = 0;
            k = k + 1;
         end
      end
   end
   n = 1;
   while n <= nr_LIS
      if LIS(n,3) == 1
         if v(LIS(n,1),LIS(n,2)) == 1
            bitai(k) = 1;
            k = k + 1;
            i = LIS(n,1); j = LIS(n,2);
            temp = [2*i-1 2*j-1; 2*i-1 2*j; 2*i 2*j-1; 2*i 2*j];
            for mm = 1:4
               if abs(duom(temp(mm,1),temp(mm,2))) >= 2^T
                  bitai(k) = 1;
                  k = k + 1;
```

```
                LSP(nr_LSP,:) = [temp(mm,1) temp(mm,2)];
                nr_LSP = nr_LSP + 1;
                if sign(duom(temp(mm,1),temp(mm,2)))>0
                   bitai(k) = 1;
                else
                   bitai(k) = 0;
                end
                k = k + 1;
             else
                bitai(k) = 0;
                k = k + 1;
                nr_LIP = nr_LIP + 1;
                LIP(nr_LIP,:) = [temp(mm,1) temp(mm,2)];
             end
          end
          if i*4 <= size(duom,1) && j*4 <= size(duom,2)
             nr_LIS = nr_LIS + 1;
             LIS(nr_LIS,:) = [LIS(n,1:2) 0];
          end
          LIS(n,3) = 2;
       else
          bitai(k) = 0;
          k = k + 1;
       end
    else
       if LIS(n,3) == 0
          if v(LIS(n,1)*2-1,LIS(n,2)*2-1) == 1 ||...
                v(LIS(n,1)*2-1,LIS(n,2)*2) == 1 ||...
                v(LIS(n,1)*2,LIS(n,2)*2-1) == 1 ||...
                v(LIS(n,1)*2,LIS(n,2)*2) == 1
             bitai(k) = 1;
             k = k + 1;
             nr_LIS = nr_LIS + 4;
             LIS(nr_LIS-3:nr_LIS,:) = [2*LIS(n,1)-1 2*LIS(n,2)-1 1;
                2*LIS(n,1)-1 2*LIS(n,2) 1;
                2*LIS(n,1) 2*LIS(n,2)-1 1;
                2*LIS(n,1) 2*LIS(n,2) 1];
             LIS(n,3) = 2;
          else
             bitai(k) = 0;
             k = k + 1;
          end
       end
    end
    n = n + 1;
end
for i = 1:nr_LSP-1
    if abs(duom(LSP(i,1),LSP(i,2))) >= 2^(T+1)
       bitai(k) = bitget(abs(round(duom(LSP(i,1),LSP(i,2)))),T+1);
       k = k + 1;
```

```
            end
        end
        T = T-1;
        if T < 0+tttt
            bitai = bitai(1,1:k);
            break
        end
    end

    function v = Tikrinimas3(Y,T)

    u = zeros(size(Y,1)/2,size(Y,2)/2);
    v = zeros(size(Y,1)/2,size(Y,2)/2);
    N = size(Y,1);
    for k1 = N/2:-1:1
        for k2 = N/2:-1:1
            if Y(k1*2-1,k2*2-1) >= 2^T && Y(k1*2-1,k2*2-1) < 2^(T+1)...
                || Y(k1*2-1,k2*2) >= 2^T && Y(k1*2-1,k2*2) < 2^(T+1)...
                || Y(k1*2,k2*2-1) >= 2^T && Y(k1*2,k2*2-1) < 2^(T+1)...
                || Y(k1*2,k2*2) >= 2^T && Y(k1*2,k2*2) < 2^(T+1)
                u(k1,k2) = 1;
            else
                u(k1,k2) = 0;
            end
            if max(k1,k2) > N/4
                v(k1,k2) = u(k1,k2);
            end
            if max(k1,k2) > N/8 && max(k1,k2) <= N/4
                v(k1,k2) = bitor(u(k1,k2),bitor(bitor(u(k1*2-1,k2*2-1),u(k1*2-
1,k2*2)),bitor(u(k1*2,k2*2-1),u(k1*2,k2*2))));
            end
            if max(k1,k2) <= N/8
                v(k1,k2) = bitor(u(k1,k2),bitor(bitor(bitor(u(k1*2-1,k2*2-1),u(k1*2-
1,k2*2)),bitor(u(k1*2,k2*2-1),u(k1*2,k2*2))),bitor(bitor(v(k1*2-1,k2*2-1),v(k1*2-
1,k2*2)),bitor(v(k1*2,k2*2-1),v(k1*2,k2*2)))));
            end
        end
    end
```

Programa Matlab parašytas SPIHT dekodavimo algoritmas:

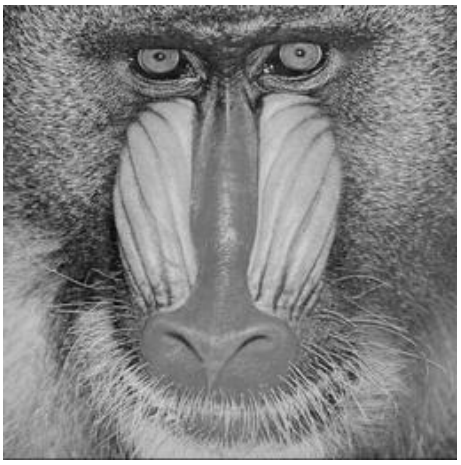
```
function X = ADLGT(spektras)

Y = 100*spektras;
X = zeros(size(Y,1),size(Y,2));
n = log2(size(Y,1));
o = zeros(size(Y,1)/2);
e = zeros(size(Y,1)/2);
d = zeros(size(Y,1));
for m = 1:2^n
   s(1) = Y(m,1);
   for i = n:-1:1
      for j = 1:2^(n-i)
         d(j) = Y(m,2^(n-i)+j);
      end
      for k = 1:2^(n-i)
         if k == 1
            o(k) = s(k) - d(k)/2;
         else
            o(k) = s(k) - (d(k)+d(k-1))/4;
         end
      end
      for k = 1:2^(n-i)
         if k == 2^(n-i)
            e(k) = d(k) + o(k);
         else
            e(k) = d(k) + (o(k)+o(k+1))/2;
         end
      end
      for j = 1:2^(n-i)
         s(2*j-1) = o(j);
         s(2*j) = e(j);
      end
   end
   Y(m,:) = s;
end
for m = 1:2^n
   s(1) = Y(1,m);
   for i = n:-1:1
      for j = 1:2^(n-i)
         d(j) = Y(2^(n-i)+j,m);
      end
      for k = 1:2^(n-i)
         if k == 1
            o(k) = s(k) - d(k)/2;
         else
            o(k) = s(k) - (d(k)+d(k-1))/4;
         end
      end
```

```
        for k = 1:2^(n-i)
            if k == 2^(n-i)
                e(k) = d(k) + o(k);
            else
                e(k) = d(k) + (o(k)+o(k+1))/2;
            end
        end
        for j = 1:2^(n-i)
            s(2*j-1) = o(j);
            s(2*j) = e(j);
        end
    end
    X(:,m) = s;
end
for m = 1:2^n
    for j = 1:2^n
        if X(m,j) > 256
            X(m,j) = 256;
        else
            if X(m,j) < 1
                X(m,j) = 1;
            else
                X(m,j) = round(X(m,j));
            end
        end
    end
end
```

2 PRIEDAS. DARBE NAUDOTI SKAITMENINIAI VAIZDAI

(a)



(b)

(c)

(d)

(e)

(f)

2.1 pav. Eksperimentams atlikti naudoti skaitmeniniai vaizdai: (a) *Nature.bmp,* (b) *Baboon.bmp,* (c) *Bridge.bmp,* (d) *Cameraman.bmp,* (e) *Lena.bmp,* (f) *Mountain.bmp*

3 PRIEDAS. SPIHT ALGORITMŲ VEIKIMO LAIKŲ LENTELĖS

3.1 lentelė. SPIHT algoritmų veikimo laikai

Pavadinimas	Dydis	Metodas	Glodumas	r = 0	r = 1	r = 2	r = 3	r = 4	r = 5	r = 6	r = 7	r = 8
Baboon	128	1	0,57	0,2846	0,2688	0,25	0,2188	0,1746	0,11	0,0594	0,0218	
Baboon	128	2	0,57	0,1904	0,1718	0,1564	0,1312	0,1096	0,093	0,075	0,0688	
Baboon	128	3	0,57	0,1876	0,1656	0,1404	0,1188	0,0844	0,047	0,025	0,0188	
Baboon	256	1	0,54	1,1438	1,0688	0,9968	0,875	0,697	0,4658	0,244	0,1	
Baboon	256	2	0,54	0,7374	0,6812	0,603	0,5246	0,428	0,3376	0,278	0,253	
Baboon	256	3	0,54	0,7468	0,6436	0,5564	0,444	0,3062	0,1934	0,103	0,047	
Baboon	512	1	0,51	4,453	4,178	3,8782	3,4282	2,6718	1,7436	0,9188	0,3688	
Baboon	512	2	0,51	3,0344	2,7562	2,478	2,1374	1,7346	1,3846	1,1504	1,0562	
Baboon	512	3	0,51	2,9594	2,5844	2,2156	1,7532	1,2218	0,7436	0,387	0,1626	
Bridge	128	1	0,64	0,2718	0,2449	0,1963	0,151	0,1057	0,0612	0,0364	0,0218	
Bridge	128	2	0,64	0,151	0,1326	0,1073	0,0921	0,0782	0,0695	0,0657	0,0636	
Bridge	128	3	0,64	0,1511	0,125	0,098	0,074	0,0536	0,036	0,0276	0,0198	
Bridge	256	1	0,65	1,0158	0,919	0,7286	0,5158	0,3376	0,1934	0,0812		
Bridge	256	2	0,65	0,547	0,4686	0,375	0,297	0,25	0,2222	0,2094		
Bridge	256	3	0,65	0,5406	0,4436	0,3184	0,2186	0,1498	0,0908	0,0468		
Bridge	512	1	0,66	4,2348	3,8094	3,1094	2,2722	1,4626	0,878	0,4968	0,2374	
Bridge	512	2	0,66	2,3092	1,997	1,6532	1,3342	1,1282	1,025	0,9846	0,9846	
Bridge	512	3	0,66	2,2344	1,8468	1,4156	1,0092	0,678	0,4406	0,2812	0,1466	
Cameraman	128	1	0,76	0,2854	0,2562	0,2172	0,185	0,1469	0,0937	0,0598	0,0297	
Cameraman	128	2	0,76	0,1656	0,1427	0,1245	0,1099	0,0953	0,0895	0,0735	0,0678	
Cameraman	128	3	0,76	0,1651	0,1391	0,112	0,0921	0,0708	0,0511	0,0324	0,0182	
Cameraman	256	1	0,77	1,1218	0,972	0,812	0,6842	0,5438	0,3906	0,2532	0,1464	0,078
Cameraman	256	2	0,77	0,6314	0,5374	0,469	0,4156	0,3656	0,3186	0,2874	0,2788	0,2782
Cameraman	256	3	0,77	0,6186	0,5094	0,4188	0,3408	0,2594	0,1876	0,1282	0,081	0,0408
Cameraman	512	1	0,8	4,1312	3,5404	2,6434	2,0218	1,472	1,028	0,6844	0,4282	0,2156
Cameraman	512	2	0,8	2,3934	2,0094	1,6376	1,4248	1,281	1,1878	1,1468	1,1314	1,1282
Cameraman	512	3	0,8	2,3096	1,847	1,3876	1,0908	0,8126	0,5998	0,4312	0,281	0,15
Lena	128	1	0,63	0,2876	0,2686	0,2344	0,1936	0,144	0,097	0,0562	0,0282	
Lena	128	2	0,63	0,169	0,1498	0,1282	0,1094	0,0966	0,0812	0,0688	0,0654	
Lena	128	3	0,63	0,1686	0,1472	0,1156	0,0938	0,0688	0,0498	0,0314	0,022	
Lena	256	1	0,66	1,063	0,9778	0,8248	0,6028	0,4312	0,2814	0,1592	0,072	
Lena	256	2	0,66	0,6184	0,5436	0,45	0,3658	0,3124	0,2716	0,2532	0,247	
Lena	256	3	0,66	0,6124	0,516	0,3938	0,2908	0,2062	0,1408	0,0844	0,0436	
Lena	512	1	0,68	4,1184	3,8188	3,281	2,266	1,3374	0,8342	0,5	0,2406	
Lena	512	2	0,68	2,4592	2,1746	1,834	1,4028	1,1468	1,0562	1,0032	0,9968	
Lena	512	3	0,68	2,3814	2,0158	1,5878	1,0686	0,6874	0,469	0,2966	0,1532	
Mountain	128	1	0,65	0,2843	0,2677	0,248	0,2233	0,1838	0,1156	0,0546	0,026	
Mountain	128	2	0,65	0,1911	0,1792	0,1567	0,1391	0,1146	0,0901	0,0698	0,0666	
Mountain	128	3	0,65	0,1918	0,162	0,1443	0,1162	0,0854	0,0521	0,0292	0,0183	
Mountain	256	1	0,56	1,1878	1,1124	1,044	0,9376	0,8066	0,5906	0,3218	0,1282	
Mountain	256	2	0,56	0,7782	0,703	0,6374	0,5592	0,469	0,3718	0,2842	0,2562	

Pavadinimas	Dydis	Metodas	Glodumas	r = 0	r = 1	r = 2	r = 3	r = 4	r = 5	r = 6	r = 7	r = 8
Mountain	256	3	0,56	0,7652	0,6844	0,5812	0,4718	0,3462	0,222	0,1096	0,05	
Mountain	512	1	0,5	4,9998	4,7156	4,4342	4,0464	3,428	2,494	1,5406	0,8376	0,3566
Mountain	512	2	0,5	3,2656	2,9872	2,7032	2,3846	2,0032	1,622	1,3406	1,2094	1,1874
Mountain	512	3	0,5	3,1718	2,797	2,428	1,9938	1,4716	0,962	0,5594	0,306	0,153
Nature	128	1	0,65	0,2901	0,2709	0,2464	0,2057	0,175	0,123	0,0663	0,0308	
Nature	128	2	0,65	0,1864	0,1695	0,1552	0,1359	0,1099	0,09	0,0766	0,0681	
Nature	128	3	0,65	0,1849	0,1596	0,1291	0,112	0,0818	0,0558	0,0326	0,0199	
Nature	256	1	0,67	1,0968	1,0188	0,897	0,7404	0,5594	0,359	0,1872	0,079	
Nature	256	2	0,67	0,675	0,6028	0,5248	0,4378	0,3656	0,3	0,2626	0,25	
Nature	256	3	0,67	0,6688	0,5688	0,472	0,3624	0,2594	0,1628	0,0938	0,046	
Nature	512	1	0,69	4,4274	4,131	3,7346	3,1188	2,1532	1,203	0,7182	0,4186	0,203
Nature	512	2	0,69	2,8438	2,566	2,2654	1,9122	1,5122	1,2626	1,1816	1,1564	1,153
Nature	512	3	0,69	2,766	2,387	1,9968	1,5376	1,0282	0,6406	0,4314	0,2812	0,147