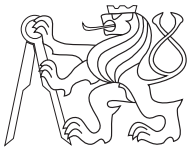
CENTER FOR
MACHINE PERCEPTION



CZECH TECHNICAL
UNIVERSITY

PHD THESIS

Bezeztrátová komprimace obrázků založená na bitové analýze obrazových dat

Jaroslav Fojtík

fojtik@vision.felk.cvut.cz

DOKTORSKÁ DISERTACE

№ K335-99/177

Lze získat na
`ftp://cmp.felk.cvut.cz/pub/cvl/articles/fojtik/compress.Fojtik_PhD99.gz`

Odborný školitel: prof. Ing. Václav Hlaváč, CSc.
Administrativní školitel: Doc. Ing. Bohuslav Kirchmann CSc.

Tento výzkum byl podporován z grantu českého ministerstva školství VS96049 a z grantu grantové agentury České republiky 102/97/0855.

Centrum strojového vnímání, Katedra řídicí techniky elektrotechnické fakulty ČVUT
Karlovo náměstí 13, 121 35 Praha 2
fax (02) 2435 7385, tel. (02) 2435 7465, http://cmp.felk.cvut.cz

Poděkování

Tato práce je vyvrcholením mého doktorandského studia na Českém vysokém učení technickém, fakultě elektrotechnické, Centru strojového vnímání v Praze. Práce byla udělána pod odborným vedením prof. Ing. Václav Hlaváče, CSc. a administrativním vedením doc. Ing. Bohuslava Kirchmanna, CSc. Oběma školitelům děkuji ze vše, co pro mě udělali.

Celá moje práce je inspirována prací prof. Ing. M. I. Schlesingera, DrSc., který na jaře roku 1996 přednášel předmět "Vybrané partie teorie rozpoznávání pro zpracování obrazu". Tímto mu děkuji za inspiraci a námět mojí další práce a konzultace o dalších problémech a jejich řešeních.

Dále bych rád poděkoval Dr. Ing. Jiřímu Matasovi za mnoho podmětných nápadů, Ing. Janu Vydrželovi a Ing. Pavlu Krskovi za pečlivé čtení mého rukopisu.

Většinu prací včetně tohoto textu jsem zpracovával v Centru strojového vnímání, které je díky svému vedoucímu a též všem ostatním spolupracovníkům velmi dobře vybaveno. Část mého poděkování by měla patřit i jim za to, že mi bylo umožněno zde pracovat.

Obsah

Kapitola 1

Úvod

Práce, kterou držíte v ruce, se zabývá z větší části algoritmy komprimace rastrových obrázků.

Je inspirována nápady prof. Schlesingera, který vyvinul funkční metody pro zpracování binárních obrázků založené na analýze malého okolí každého pixelu. Podle provedené analýzy okolí je možno navrhnout binární prediktor a nebo rychlé binární gramatiky pro analýzu dat.

1.1 Proč jsem se zabýval komprimací dat?

Obrazová data jsou ve většině případů poměrně rozsáhlá a svojí prací jsem chtěl přispět k jejich efektivnějšímu ukládání a zpracování.

Na přístupu prof. Schlesingera jsem shledal zajímavé a užitečné vlastnosti. V algoritmu jsou data zpracovávána pro počítače nejpřirozenějším možným způsobem tedy po bitech, což může při vhodné implementaci umožnit rychlé SW popř. HW řešení. Podle prof. Schlesingera by mohl celý jeho přístup po malých úpravách být integrován do hardware a používán např. pro komprimaci videa.

1.2 Struktura práce

Kapitola 1, kterou právě čtete, obsahuje úvod do problematiky celé práce. V následující kapitole 3 jsou rozebrány jiné komprimační algoritmy a jsou diskutovány jejich klady a zápory. Kapitola 4 již pojednává o mé práci. Jsou v ní podrobně popsána jednotlivá rozšíření navrhovaného komprimačního algoritmu. O problematice komprimace pseudobarevných obrázků s paletou je diskutováno v kapitole 5. Rozbor experimentů, které vypovídají o kvalitě navržených algoritmů, je uveden na konci kapitoly 4 o komprimaci a na konci kapitoly 5 o pseudobarevných obrázcích. Podle experimentálních dat je možno ověřit kvalitu kompresního algoritmu. Poslední kapitola 7 obsahuje shrnutí celého textu a náměty pro budoucí práci.

Práce obsahuje i několik příloh. Příloha A je určena pro usnadnění práce programátora, který by se rozhodl implementovat navrhovaný algoritmus. Obsahuje dobře komentované důležité části komprimačního algoritmu. Slovníček použitých termínů je obsažen v poslední příloze B. Obsahuje speciální a málo známé termíny použité v textu, jejichž objasnění může napomoci čtenáři se lépe orientovat v textu.

1.3 Typografické členění dokumentu

Celou práci je možno rozčlenit do následujících celků v desetinném třídění. Nejvýše jsou jednotlivé *kapitoly*. Pod nimi se nacházejí *sekce*. Ještě níže v hierarchii jsou umístěny *oddíly*. Takto budou jednotlivé celky též citovány.

Poznámky pod čarou jsou použity k bližšímu upřesnění právě použitého termínu. V matematické sazbě jsou skalární proměnné bez označení $a = b^2$, vektory jsou zvýrazněny šipkou $\vec{r}_i = [q_1 ... q_3]$ a matice jsou sázeny tučně $\mathbf{S}(x, y)$. Zmenšeným písmem jsou označeny příklady a jejich řešení.

Celý text byl na přání školitele napsán v typografickém systému LaTeX a obrázky byly nakresleny programem Xfig. Program Xfig bohužel není určen k psaní akcentovaných písmen. Proto se omlouvám čtenáři za anglické texty v grafech a obrázcích.

Kapitola 2

Cíle disertační práce

V disertační práci jsem si kladl tři cíle. Jejich popis je uveden v následujícím výčtu:

Komprimace šedotónových obrázků. Dal jsem si za úkol zobecnit a upravit prediktor[1] prof. Schlesingera pro komprimaci šedotónových obrázků. K dosažení uvedeného cíle je potřeba původní přístup analyzovat. Dále jsem si dal za úkol celý přístup doplnit tak, aby byl vhodný i pro komprimaci šedotónových obrázků.

Mým cílem bylo i prozkoumání, zda lze zobecnit Schlesingerův prediktor do více rozměrů. V kladném případě by tím vznikl n-rozměrný semiadaptivní (adaptuje se 1x po analýze dat) prediktor, který by měl být vhodný pro komprimaci obrázků.

Komprimace barevných paletových obrázků. Dalším úkolem, který jsem řešil, bylo využití předchozího komprimačního algoritmu pro komprimaci barevných obrázků s paletou. Jakýkoliv pokus o komprimaci paletových obrázků jako šedotónových, který pouze ignoruje paletu, dává neuspokojivé výsledky. Proto jsem si dal za úkol najít způsob, jak modifikovat komprimační algoritmus pro speciální třídu paletových obrázků.

Dvou a více rozměrné metody pro rychlé zpracování binárních dat. Posledním řešeným úkolem je zvážit, zda by nebylo možno zobecnit původní metody prof. Schlesingera pro binární operace se zkomprimovanými 2D daty do více rozměrů. Při použití prof. Schlesingerova přístupu dojde pro binární obrázky k mnohonásobnému urychlení některých početních operací (jedná se o binární funkce NOT, OR, AND, XOR a dále o funkce matematické morfologie: skeleton, dilatace, eroze). Při použití zobecněného přístupu by mělo dojít k urychlení některých početních operací nad n

[1] Prof. Schlesinger vyvinul prediktor, který predikuje data v binárních dvouúrovňových obrázcích na základě analýzy četností výskytů malého okénka (sondy).

dimenzionálními binárními útvary. Pokud by zobecnění bylo možné, tak by bylo vhodné jej experimentálně ověřit.

Kapitola 3

Stav znalostí a přehled o pracech ostatních

3.1 Komprimace dat

Jako první je potřeba vyjasnit termín komprimace, co od ní lze očekávat a jaké může poskytovat výsledky. Tyto informace jsou uváděny z důvodu, aby na komprimační algoritmy nebyly kladeny přílišné nebo nesplnitelné nároky.

Informace, která má být přenášená, nebo uložená na nějakém médiu, se nazývá *zpráva*. Aby bylo možno se zprávou manipulovat, je potřeba ji kódovat a zpracovávat v kódovaném tvaru. V počítačové technice jsou pro kódování využity buňky, které mohou nabývat pouze jedné ze dvou hodnot 0 nebo 1. Buňky se nazývají bity. Zmiňované bity jsou dostatečné pro popis většiny zpráv předávaných v našem okolí. Je možno pomocí nich kódovat a předávat textové zprávy, obrazová data, zvuková data atd. Nejčastěji se namísto bitů pracuje s většími složenými jednotkami jako je slovo, slabika atd, ale to není pro tuto úvahu podstatné.

Pro přenos kódované zprávy je potřeba určité množství elementárních buněk. Počet buněk potřebných k zakódování zprávy se nazývá *délka zprávy*. Každá buňka patřící zprávě dostane nějaký význam pro obsah zprávy. Kódovaná zpráva je ukládána na cílové médium anebo je zasílána přenosovou cestou do vzdáleného bodu. Kapacita cílového média a nebo přenosové cesty není neomezená, a proto vzniká požadavek na komprimaci zpráv. Komprimace je jiné kódování zprávy, které má za následek zmenšení délky zprávy měřené počtem buněk.

> Mějme kódovanou zprávu M o délce l (měřenou např. počtem elementárních buněk). Naším cílem je převést ji na zprávu M^* o délce l^*, kde bychom si přáli, aby $l^* < l$. Jestliže existuje dvojice funkcí f a f^*, které jednoznačně převádějí jedno kódování na druhé: $M^* = f(M)$; $M = f^*(M^*)$, a pro všechny možné zakódované zprávy M jednoznačně platí $M = f^*(f(M))$, hovoříme o *bezeztrátové kompresi*.

Zprávu M^* považujme za komprimovanou. Funkci f převádějící základní kódování na kódování komprimované si nazvěme *komprimační funkcí*. Obdobně pojmenujme funkci f^*, která obnovuje původní kódovaný tvar zprávy, *funkcí dekomprimační*.

Poznamenejme, že podmínkou bezeztrátové komprese není jednoznačnost komprimované zprávy M^*. To znamená, že k původní nekomprimované zprávě může existovat celá množina zpráv komprimovaných.

Neplatí-li podmínka $M = f^*(f(M))$ alespoň pro jedinou zprávu M, pak hovoříme o *ztrátové kompresi*. U ztrátové komprese hraje roli *ztrátový činitel* ξ, který lze definovat jako vzdálenost původní a rekonstruované zprávy.

$$\xi = ||M - f^*(f(M))||. \tag{3.1}$$

Metrika[1] ohodnocuje odlišnost původní zprávy od zprávy, která je komprimována ztrátovou kompresí. Existují metriky vhodné pro všechny typy zpráv např. absolutní hodnota rozdílu anebo suma kvadrátu rozdílů. Lepších kompresních poměrů lze dosáhnout zejména u obrazových a zvukových dat analýzou jevů, na které je lidské oko (ucho) nejcitlivější. Metriku definující vzdálenost dvou zpráv je vhodné v takovém případě stanovit s ohledem na obsah zprávy. Pro bezeztrátovou kompresi musí být ztráta vždy nulová bez ohledu na použitou metriku.

Kdyby byla pravděpodobnost všech možných kódování zpráv stejná, pak by se stala jakákoliv bezeztrátová komprese zbytečnou, poněvadž by nepřinášela kýžený výsledek spočívající ve zkrácení délky zprávy a některé zprávy by zákonitě byly prodlouženy. U běžně předávaných textových, obrazových a zvukových dat není předchozí podmínka týkající se vyrovnané pravděpodobnosti všech zpráv splněna. To znamená, že některé zprávy jsou o něco pravděpodobnější, než jiné. Vytváří-li zdroj dat některé zprávy častěji než jiné, pak lze ve zprávách nalézt nadbytečnou informaci (*redundanci*). Pomocí redundance lze definovat komprimaci jako snahu nalézt a odstranit všechny redundance z dat.

3.1.1 Metody pro porovnávání účinnosti komprese

Pro vzájemné porovnávání délky výsledného kódu se používá několik kritérií. Výsledkem každého kritéria je číslo úměrné zmenšení délky zprávy. Známe délku původní zprávy a délku zprávy po komprimaci. Nejjednodušší možností je vypočítat podíl obou čísel a na jeho základě usuzovat o kvalitě komprese. Nazvěme takový podíl kompresním poměrem. V dalším textu není striktně požadován takovýto způsob výpočtu a termín *kompresní poměr* je považován za výsledek jakékoliv

[1] V textu jsou používány dva termíny *Metrika* a *vzdálenost*. Pro potřebu textu si blíže vymezme jejich odlišnost. *Metrika* je funkce definovaná na kartézském součinu dvou prostorů $x \times x$, která splňuje další podmínky, a její funkční hodnotou je skalární veličina. Skalární veličinu si nazvěme v dalším textu *vzdálenost*.

funkce, která ohodnocuje kvalitu komprese. Samozřejmě že při porovnávání dvou komprimačních algoritmů je třeba používat stejnou funkci. Ukažme několik běžně používaných funkcí.

Jedním z často používaných kritérií je *počet bitů na pixel* . Každý rastrový obrázek má v nekomprimovaném tvaru přesně definovaný maximální počet hodnot, kterých může nabývat jeden pixel. Obvykle je uváděna velikost pixelu přímo v bitech. V komprimovaném tvaru dochází ke zmenšení (nebo někdy i zvětšení) objemu dat. Objem dat ve zkomprimovaném tvaru je možno fiktivně rozpočítat na jednotlivé pixely a přiřadit každému pixelu jakousi pseudodélku. Tato pseudodélka pixelu je uváděna jako počet bitů na pixel. Kritérium počet bitů na pixel je celkem hojně používáno. Neumožňuje však porovnávat mezi sebou úspěšnost komprimace pro dva obrázky s nestejnou velikostí pixelu.

Poněvadž je zde zapotřebí srovnávat obrázky s nestejnou velikostí pixelu, bylo vybráno jiné kritérium. Pro porovnání úspěšnosti jednotlivých komprimačních metod bylo využito kritérium *účinnost komprese* CE převzaté z [CAJ96]. Akronym CE byl vytvořen z anglických slov 'Compression Efficiency'. Údaje z jiných článků udávaná v jiných kritériích byla přepočítána na *účinnost komprese* za účelem porovnání výsledků.

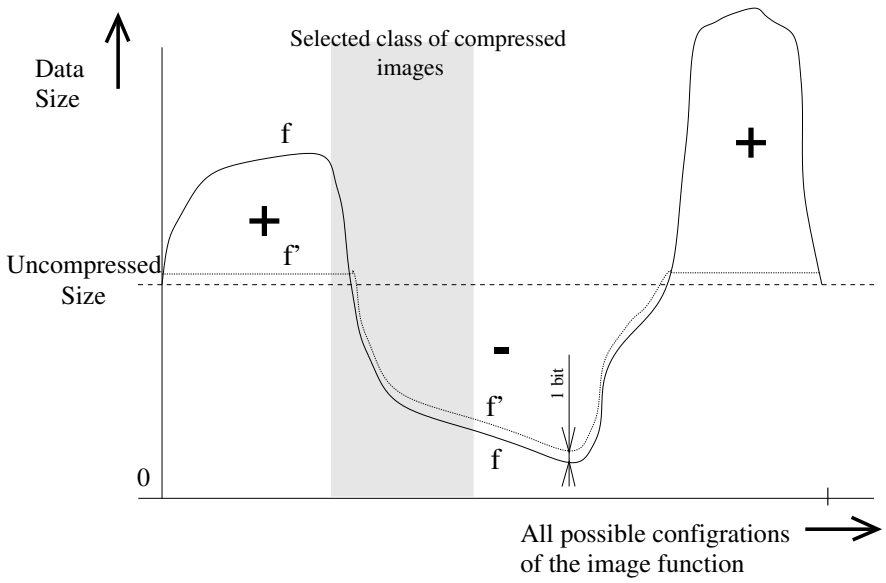
$$CE = \frac{\text{vstupních\_bajtů\_celkem} - \text{bajtů\_na\_výstupu\_celkem}}{\text{vstupních\_bajtů\_celkem}} \cdot 100\% \qquad (3.2)$$

3.1.2 Analýza komprimační funkce f

Graf na obr. 3.1 je pouze ilustrativní. Demonstruje činnost hypotetického komprimačního algoritmu. Na ose x jsou vyneseny všechny možné zprávy, které byly nějak uspořádány. Hlavním účelem je demonstrovat, že jakýkoliv komprimační algoritmus nemůže zkrátit délku kódu pro všechny typy zpráv. Vždy existují takové zprávy, pro něž dochází k prodloužení délky kódu. Ty by měly mít nejmenší pravděpodobnost výskytu. Hypotetická množina vysoce pravděpodobných dat je na obr. 3.1 zobrazena šedým pásem.

Typický průběh komprimační funkce f je zobrazen plnou čarou v grafu pro všechny možné zprávy M. V oblastech označených + dochází k prodloužení zprávy a v oblastech označených znakem − dochází naopak ke zkrácení délky zprávy. Je nemožné snížit množství dat pro všechny zprávy současně. Samozřejmě s výjimkou případů, kdy ke zprávě byla přidána nadbytečná data[2], která jsou následně odstraňována. V takovém případě se však o kompresi nejedná. Pokud v nějakém místě dojde k posunu křivky směrem dolů, v jiném automaticky dojde ke zvednutí křivky směrem nahoru. Prodloužení zprávy (záporná komprese) je u kompresních algoritmů nežádoucí jev a je snaha jej potlačit na únosnou míru.

[2]Data představují blíže nespecifikovanou část nebo části zprávy obsahující nějakou informaci. Data lze při kódování rozložit na elementární buňky.



Obrázek 3.1: Typická závislost kompresního poměru pro všechny možné zprávy.

Dílčí eliminace záporné komprese

Je sice pravda, že záporné komprese se nelze úplně zbavit, ale její vliv lze velmi snadno eliminovat na minimum. Dokonce tento vliv může být tak malý, že jej uživatel kompresního algoritmu ani nezaregistruje.

Popis navrhovaného postupu jsem sice nikde v literatuře nenašel, ale předpokládám, že je znám. Většina komprimačních algoritmů umožňuje ukládat též nekomprimovaná data a některé lepší algoritmy nejprve odhadnou kompresní poměr. V případech, kdy dochází k prodloužení zprávy, jsou data ukládána jako nekomprimovaná.

Mějme libovolnou komprimační funkci f a snažme se k ní najít novou podobnou funkci f', která minimalizuje prodloužení zprávy. Po nové komprimační funkci f' bychom požadovali, aby se chovala téměř stejně jako funkce f v oblastech '$-$' (viz. obr. 3.1) kladné komprese a aby nechávala vstupní zprávu téměř beze změny v oblastech '+' (viz. obr. 3.1) záporné komprese:

$$\begin{array}{lcll} f'(M) & = & \{0, M\} & length(f(M)) \geq length(M) \\ f'(M) & = & \{1, f(M)\} & length(f(M)) < length(M) \end{array} \tag{3.3}$$

Je-li komprimační funkce f' definována podle rovnice (3.3), pak výstupní zpráva z funkce f' může být maximálně o 1 bit delší než původní zpráva. Funkce f' produkuje téměř stejné výsledky jako funkce f oblasti $-$ (viz. obr. 3.1), ale v oblasti + dochází ve většině případů k citelnému vylepšení. Tato myšlenka může být využita i vícenásobně v rámci jednoho kompresního algoritmu např. pro dílčí

části zprávy.

3.1.3 Dimenze dat (prediktoru)

Mám-li k dispozici data, u kterých neexistuje žádná souvislost mezi dvěma prvky, mohu o nich prohlásit, že mají dimenzi 0-DOF[3]. Jedinou možnou metodou komprimace se pak jeví kódování na základě četnosti symbolů. Četnější symboly budou používat kratší kódová slova a méně četné symboly dostanou delší kódová slova. K tomuto účelu se hodí např. Huffmanovo kódování [Ani89] nebo aritmetické kódování. Předchozí tvrzení však platí i opačně. Je-li Huffmanovo kódování aplikováno přímo na data, pak jsou využívány jen informace o četnosti jednotlivých symbolů. S daty se pracuje tak, jako kdyby měla dimenzi pouze 0-DOF a ostatní možnosti zvýšení kompresního poměru jsou tudíž ztraceny.

U běžných textových dat, binárních kódů programů, navzorkovaných zvukových dat atd. existuje určitá souvislost mezi předchozím a následujícím prvkem. O takovýchto datech lze prohlásit, že mají dimenzi 1-DOF. Existující závislosti lze využít např. tak, že jsou vyhledávány podřetězce, které se často opakují. Další možností může být práce pouze s diferencemi původních dat (tedy rozdílu předcházejícího a aktuálního prvku). I nalezená a využitá slabá závislost v datech se může výrazně projevit na zvýšení kompresního poměru.

U dat vzniklých z rastrových obrazů existuje ještě další závislost. Aktuální prvek má nejen 2 sousedy napravo a nalevo, ale ještě další 2 nahoře a dole. O rastrových datech lze tudíž prohlásit, že mají 2D strukturu. Znalost struktury umožňuje další zvýšení kompresního poměru. Rastrové obrazy sice lze komprimovat stejně jako 1-DOF data, ale tím se ochuzujeme o možnost zvýšení kompresního poměru.

V této práci byla dosud za rastrová data považována pouze data obsahující odstíny šedi. Existují však multispektrální barevné obrazy, u kterých existuje závislost nejen horizontální a vertikální, ale i napříč barevnými spektrálními kanály. Barevná data v kódování RGB jsou jen zjednodušeným provedením předchozího případu. O takovýchto datech lze říci, že mají 3-DOF strukturu. Přirozenou 3-DOF strukturu mají prostorová data, jež pocházejí např. z tomografu.

Z pohledu jednotlivých bitů lze na data pohlížet ještě dalším způsobem. Každá buňka struktury má určitou velikost (znak-bajt, slovo-word). Jednotlivé bity jsou však také uspořádány. Toto uspořádání lze též považovat za další dodatečnou dimenzi dat a náležitým způsobem ji využít. Počet uspořádaných bitů pro popis jednoho pixelu se někdy nazývá *bitová hloubka*. Při uvažování bitové hloubky je možno nahlížet na šedotónové obrázky jako na data s 3-DOF strukturou a na tomografická data jako na data s 4-DOF strukturou.

Každá nalezená (a použitá) závislost v datech snižuje množství informace na jednu datovou buňku, a tím umožňuje při kompresi snížit celkovou délku zprávy.

[3] (Z anglických slov Degree Of Freedom) Jedná se o zkratku sloužící k označení počtu rozměrů dat.

Každá dodatečná znalost struktury dat vede totiž k vylepšení datového modelu. Dnes používané modely pro rastrová data plně neodrážejí jejich dvoudimenzionální obsah.

Většina dnes používaných algoritmů (jejich detailní popis je uveden v sekci 3.5) používá pro komprimaci obrazu pouze jednoduchý 1-DOF datový model. Jedná se např. o BMP, PCX - (pouze RLC), GIF - (pouze LZW) viz [Mel96], WPG atd. V knize [Dv94] je uveden detailní popis desítek takovýchto formátů.

Od nových algoritmů navržených speciálně pro 2D strukturu obrazových dat by se dalo očekávat, že dosáhnou výrazně lepších výsledků než dobře známé 1-DOF kompresní algoritmy (ARJ, ZIP, AIN). Bohužel toto tvrzení není vždy pravidlem. Proto je možno v budoucnu očekávat na poli komprimace obrazových dat velké množství další práce. Existují sice algoritmy s velmi dobrými výsledky pro 2D data např. [RA96] založené na segmentaci oblastí a jejich odděleném kódování, ale ty mají neúnosnou výpočetní složitost i pro dnešní výkonné počítače.

3.2 Rozdělení bezeztrátových komprimačních metod do skupin

K dispozici je velké množství komprimačních technik specializovaných na 1-DOF data (řetězce) a používaných též pro komprimaci obrazových dat. Obrazová data mohou být buď rastrová nebo vektorová. Dále se budeme zabývat pouze daty rastrovými. Většina komprimačních technik je dostupná ve formě komerčních nebo volně šířitelných programů např. AIN, ARJ, Stacker, ZIP. Použité techniky nejsou předmětem této práce, a proto se dále budu věnovat hlavně algoritmům specializovaným na dvourozměrná obrazová data.

Věnujme se nyní rozdělení jednotlivých kompresních algoritmů do skupin. Podle [TVP97] jsou bezeztrátové komprimační techniky obvykle rozdělovány do dvou generací.

3.2.1 První generace

Algoritmy *první generace* jsou poměrně jednoduché. Právě složitost datového modelu lze použít pro hrubé rozlišení první a druhé generace. Do první generace patří například kompresní algoritmus, který po dvou po sobě následujících stejných symbolech očekává počet opakování zdvojeného symbolu.

Vzpomeňme zde několik dalších reprezentantů této kategorie: IBM Q80 kodér (bezeztrátový JPEG) [ISO94]; PNG (Portable Network Graphics) [Tea96]; jednotlivé komprimační metody podle Portable Video Research Group ve Stanfordu PVRG-JPEG popisované v článku [CAJ96].

3.2.2 Druhá generace

Do druhé generace jsou většinou zařazovány mnohem složitější přístupy. Nejčastěji jsou založeny na datovém modelu (datový model představuje nějakou apriorní informaci o datech). Bezeztrátové komprimační metody *druhé generace* jsou obyčejně rozdělovány do dvou hlavních skupin. Některé algoritmy je však obtížné zařadit do jediné skupiny, poněvadž kombinují vlastnosti obou skupin.

Hlavní podskupiny algoritmů druhé generace jsou:

A. Algoritmy založené na oblastech.

B. Algoritmy založené na predikci dat.

3.2.3 Algoritmy druhé generace založené na oblastech

Obraz je rozdělen do oblastí, které mají podobnou intenzitu (nebo nějakou jinou charakteristickou veličinu). Jednotlivé oblasti jsou zpracovány nezávisle. Do této skupiny komprimačních metod patří např. Segment [RA96].

Bylo by možno sem zařadit také algoritmy, které vyhledávají v datech určité šablony a následně se snaží celý obraz poskládat z utvořených šablon. Například pro komprimaci textu je možno si vytvořit šablony všech písmen. Je jasné, že pro reálná data šablony nebudou korespondovat s měnící se předlohou vždy na 100%. Protože se každá šablona promítá do výsledné délky komprimovaných dat, je nutno pracovat s omezeným počtem šablon. K pozicím jednotlivých šablon je nutno přidat rozdíly mezi původními daty a daty utvořenými ze šablon. Takto vzniknou rezidua, jež je potřeba též kódovat. Je nutno řešit problém závislosti počtu šablon na počtu reziduí a hledat vhodný kompromis. Výše uvedenému postupu je věnována např. práce [KD97]. Popisovaný algoritmus je optimalizován pro kódování binárních obrázků vzniklých snímáním textových předloh.

Podobný je přístup v případě vektorové kvantizace. U vektorové kvantizace jsou však obrazová data pevně rozdělena na malé části (např. čtverce 8×8 bodů). Obsahy malých kousků obrazu jsou považovány za vektory a cílem je nalézt a vyřadit co největší množství nejméně diskriminativních rozměrů v jednotlivých vektorech.

3.2.4 Algoritmy druhé generace založené na predikci dat

Metody z uvedené skupiny se snaží vytvářet interní model dat[4]. Tento model je součástí prediktoru. Rezidua vzniklá rozdílem predikované a aktuální hodnoty jsou následně kódována. Tak vzniká komprimovaná zpráva. Příkladem takového

[4] *Data* představují blíže nespecifikovanou část nebo části zprávy obsahující nějakou informaci. Data lze při kódování rozložit na elementární buňky.

algoritmu mohou být: Calic [WM97], kroužková transformace z [JCP96] a Logické kódování [CAJ96]. Uvedené příklady jsou detailně popsány v sekci 3.5.

Jiným pokusem o prediktivní model dat se jeví postup dle [Car97]. V této práci jsou v podstatě porovnávány algoritmy CALIC (viz oddíl 3.5 pro podrobnější popis) a FELICS (Fast Efficient Lossless Image Compression System) [AT96, HV93]. Dále je v [Car97] navrženo kódování reziduí aritmetickým kodérem.

Podle činnosti prediktoru lze udělat ještě následující dělení:

- *S pevným modelem dat* - prediktor se nemění v závislosti na datech.
- *Semiadaptivní* - prediktor se pevně nastaví pro celou zprávu po analýze dat. Poté se stane popis prediktoru součástí zakódované zprávy.
- *Adaptivní* - prediktor se plynule mění v závislosti na přicházejících datech v rámci jedné zprávy.

Pevný model dat je vhodný zejména pro data z jediného málo se měnícího zdroje. Hlavní výhodou a zároveň nevýhodou je popis dat zabudovaný do algoritmu. Tím lze sice ve speciálních případech o něco zmenšit délku zprávy za cenu nemožnosti adaptace algoritmu na jiná data.

Semiadaptivní algoritmy nejprve analyzují celou zprávu, poté se optimálně nakonfigurují pro konkrétní data příslušející zprávě a nakonec zprávu zakódují. Semiadaptivní algoritmy nabízí vyšší pružnost při zpracování dat. Nevýhodou je nutnost analýzy dat před zpracováním zprávy.

Adaptivní algoritmy se učí při kódování dat. Mohou komprimovat i měnící se data z nestacionárních[5] zdrojů. Avšak nemohou napřed odhadnout kvalitu kódování, jako je tomu u algoritmů semiadaptivních.

Úhel prediktoru

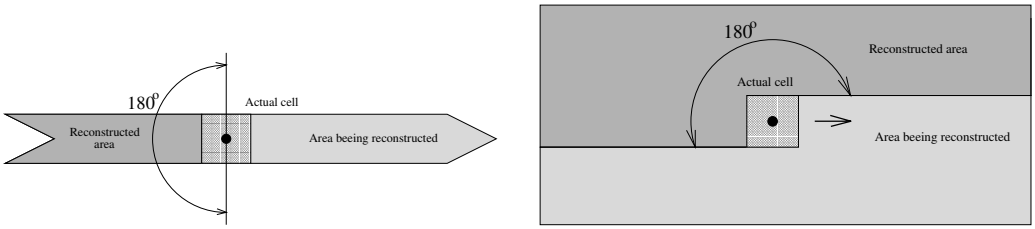
V souvislosti s predikcí aktuální hodnoty je často diskutovaným termínem tzv. *úhel prediktoru* [TVP97]. Zjednodušeně lze říci, že se jedná se o velikost oblasti, z níž jsou data predikována.

Na obr. 3.2(a) je znázorněna situace pro 1 rozměrná data. V tomto případě je situace velmi jednoduchá. Úhel prediktoru může být buď 0^o nebo 180^o. Pokud se prediktor vůbec neohlíží na předchozí data, pak je úhel prediktoru 0^o. Pokud prediktor z předchozích dat určuje data aktuální, pak je úhel prediktoru 180^o. Úhel 360^o by odpovídal nekauzálnímu prediktoru a není prakticky dosažitelný. Hlavní problém nekauzálně predikovaných dat neleží na prediktoru, ale na rekonstruktoru, který musí obnovit původní data.

Podobná situace nastává v případě 2D dat. Tam může maximální úhel prediktoru též dosáhnout 180^o a prediktory s tímto úhlem se běžně používají. Existují

[5] U nestacionárního zdroje dat se četnost jednotlivých zpráv pomalu mění s časem.

sice práce [Xia97], jejichž autoři se pokoušejí o "pseudozvětšení" tohoto úhlu. To lze udělat následujícím způsobem: V obrazových datech dojde k výběru např. každého sudého pixelu a vybrané pixely budou uloženy odděleně. Prediktor nejprve načte oddělená data, a pak jich využije k predikci ostatních dat. Tím sice výrazně vzroste úspěšnost predikce hlavních dat. Ale pomocná data se komprimují se špatným kompresním poměrem a způsobí výrazné snížení celkového algoritmem dosaženého kompresního poměru.



(a) 1 dimenzionální data (b) 2 dimenzionální data

Obrázek 3.2: Grafické znázornění úhlu prediktoru

Existuje ještě jedna poměrně zajímavá metoda jak "obejít" omezení úhlu prediktoru. Metoda je založena na výběru řídících bodů a jejich separátním uložení. Pro následnou predikci jsou používány výhradně řídící body. Jako nejjednodušší možnost lze použít ten nejbližší. Pominu skutečnost, že je pro každý obraz nutno uložit tři datové struktury: binární bitovou mapu s označením řídících bodů, řídící body a rezidua. Právě výběr řídících bodů může představovat poměrně složitý a nejednoznačný úkol. Špatný výběr řídících bodů může degradovat dosažený kompresní poměr. Možností výběrů řídících bodů je takové množství (roste s faktoriálem počtu pixelů), že k jejich výběru lze použít pouze heuristický algoritmus. Tento přístup byl popsán v [TVP97].

3.3 Často používané algoritmy

Tento oddíl se zabývá jednoduchými algoritmy používanými pro komprimaci dat. Většina složitějších metod totiž v sobě obsahuje kombinaci několika jednodušších algoritmů.

3.3.1 Huffmanovo kódování

Nejčastěji lze zprávu rozložit na posloupnost symbolů. Množina všech použitelných symbolů v rámci jedné zprávy se nazývá *abeceda*. Nejjednodušším kódováním symbolů se jeví přiřazení pořadového čísla každému symbolu. Dále již postačí manipulovat pouze s čísly do velikosti rovné počtu symbolů.

V počítačové technice jsou čísla kódována ve dvojkové soustavě. Proto pro základní kódování symbolů je většinou nalezena nejbližší vyšší mocnina 2 než je počet všech symbolů a každému symbolu je přiřazeno jedno slovo s pevnou bitovou délkou. Ještě častější je zarovnání bitové délky slova na násobek čísla 8.

Obvykle nemají všechny symboly v abecedě stejnou četnost výskytu ve zprávě. Standardně je každému symbolu vyhrazeno kódové slovo se stejnou délkou. Typicky se jedná o písmeno (bajt - 8 bitů), slovo (word - 16 bitů), dvojslovo (dword - 32 bitů) atd. Používaná reprezentace umožňuje rychlý přístup k datům. Pro dlouhodobé uložení na cílové médium se z hlediska délky kódované zprávy jeví jako výhodnější použití kódu s proměnnou délkou. Kód s pevnou délkou představuje mezní variantu kódu s proměnnou délkou, která vzniká za podmínky, že všechny symboly mají stejnou četnost.

Lze-li kód s proměnnou délkou číst jednoznačně pouze ve směru toku dat, jedná se o kód prefixový. U prefixového kódu představuje každé kódové slovo prefix pro následující zbytek zprávy. U afixového kódu lze kódovaná data číst v opačném směru. Huffmanovo kódování [Adá89] představuje variantu prefixového kódu minimalizující délku kódované zprávy. Minimalizace je prováděna na základě četností jednotlivých symbolů. Původně byl výpočet založen na pravděpodobnostech výskytu symbolu ve zprávě. Výpočet lze stejně dobře udělat s absolutními hodnotami počtu výskytů jednotlivých symbolů. Výhodou druhé varianty je možnost počítání s celými čísly.

symbol	A	B	C	D	E
počet výskytů	100	150	50	10	100

Tabulka 3.1: Hypotetická abeceda pro ukázku Huffmanova kódování

Nejlepší je demonstrovat výpočet Huffmanova kódu na příkladu.

Mějme abecedu 'ABCDE' s počty výskytů jednotlivých symbolů uvedených v tabulce tab. 3.1. Vytvořme pro uvedenou abecedu Huffmanův kód. Celý postup je zobrazen na obr. 3.3. Tvorba Huffmanova kódu začíná setříděním tabulky symbolů podle četností výskytu jednotlivých symbolů.

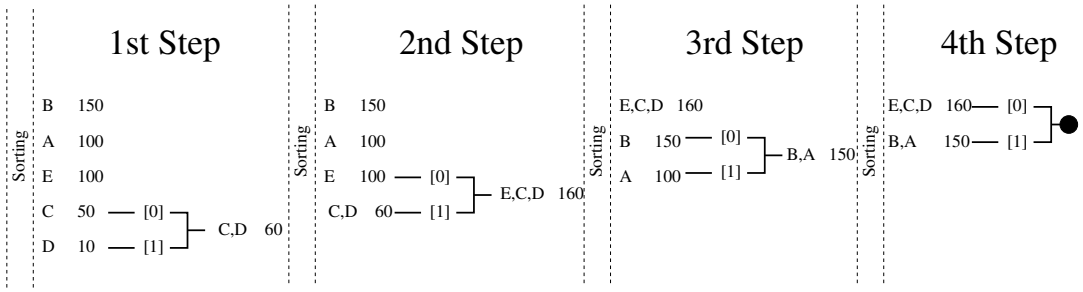
1. Poslední dva symboly s nejnižší četností jsou sloučeny dohromady a dále je s nimi počítáno jako s jedním symbolem. Četnost výskytu složeného symbolu je dána součtem četností dílčích symbolů. Pro rozlišení obou symbolů je jednomu (v našem případě C) symbolu přiřazen afix 0 a druhému (v našem případě D) afix 1.

2. Znovu sloučíme dva symboly s nejnižší četností. Jedná se o symbol E a složený symbol C,D.

3. Slučujeme symboly A a B.

4. Slučujeme symboly B,A a E,C,D.

Slučování symbolů probíhá do té doby, dokud nedojde ke sloučení poslední dvojice symbolů. Pro každý symbol lze zpětným postupem dohledat z tabulky jemu odpovídající kódové slovo.



Obrázek 3.3: Ukázka způsobu tvorby Huffmanova kódu

Analýzou výpočetní náročnosti dekódování Huffmanova kódu se zabývá článek [CL97]. Článek mě však nepřesvědčil o výhodnosti navrhované metody, protože se v praxi používají efektivnější metody, se kterými jsem se seznámil.

3.3.2 Aritmetické kódování

U Huffmanova kódu je celý kódový prostor rozdělen na slova, která mají celý počet bitů. Tím nelze zakódovat zprávu na nejkratší možnou délku, která by byla úměrná entropii četnosti symbolů.

Aritmetický kodér (někdy též nazývaný entropní) umožňuje rozdělení kódového prostoru na slova s necelým počtem bitů. Triku necelého počtu bitů je dosaženo postupným dělením zprávy.

Výstupní kód z aritmetického kodéru je optimální pro data s 0-DOF. Ponechám stranou zvýšenou výpočetní náročnost kódování. Jak Huffmanův tak i aritmetický kodér potřebuje ke své inicializaci tabulku s četnostmi jednotlivých symbolů. Tabulka pro popis intervalů aritmetického kodéru je mnohem delší, a proto se často používá plně adaptivní varianta aritmetického kodéru.

3.3.3 RLC - Proudové kódování (Run Length Coding)

Algoritmus proudového kódování se vzhledem ke své jednoduchosti řadí k metodám 1. generace. Je typickým představitelem řetězcových 1-DOF metod.

RLC pracuje na principu redukce opakovaných řetězců. Vstupní řetězec n stejných po sobě jdoucích znaků k je nazýván *proud*. Číslo k nazýváme *hodnota*

proudu. Tímto způsobem jsou vstupní data transformována na uspořádané dvojice, též nazývané *RLE*[6] *pakety*:

$$i_1, i_2 \ldots i_M \rightarrow [k_1, n_1][k_2, n_2] \ldots [k_N, n_N] \tag{3.4}$$

Výsledné RLE pakety je potřeba pro uložení na cílové médium kódovat. Kódovat je možno na dvou základních úrovních: bajtové nebo bitové. Rozdíl obou modifikací tkví ve skutečnosti, že při bajtovém kódování je bajt považován za již dále nedělitelnou jednotku. Tedy je nutno zarovnat všechna čísla na hranici osmi bitů. Bitové kódování je o něco složitější, poněvadž je potřeba číst/ukládat data neohraničená na jednotlivé bajty (slova). (Například délka paketu může být kódována 9 bity.)

Hlavní výhodou kódování RLC je velmi snadná implementace a rychlost kódování/dekódování.

3.3.4 LZW - Lempel Ziv (varianta GIF)

Jedná se o kódování, které je primárně určeno ke kompresi opakujících se podřetězců ve zprávě tj. mezi slovníkové metody s dynamickým vytvářením slovníku. Kódování je založeno na rozšíření kódované abecedy přidáním dalších symbolů. Dochází tedy ke zvýšení počtu bitů na symbol. Nové symboly totiž kódují podřetězce obsažené ve slovníku.

První část symbolů zprávy má stejný význam jako symboly původní zprávy. V originální dokumentaci (obsažena na CDROM v elektronické příloze ke knize [Dv94]) jsou označovány termínem *kořenové*. Dvěma nově přidaným symbolům je přiřazen speciální význam. Jedná se o *CLEAR* k mazání slovníku podřetězců a o *EOF* určený k zakončení přenosu. Při kódování zprávy získávají postupně jednotlivé nové symboly určitý význam. Symbol, na který zatím nedošla řada, nesmí být použit. Při dosažení posledního nového symbolu je v zásadě možno udělat dvě věci. Buďto lze slovník rozšířit o další množinu nových symbolů zvětšením bitové délky kódového slova anebo lze slovník automaticky smazat.

U varianty LZW použité pro grafické soubory GIF je zvolena následující strategie. Slovník podřetězců postupně narůstá do délky kódového slova 12 bitů. Při další potřebě zvětšení slovníku je celý slovník podřetězců automaticky zrušen a začíná se opět od začátku. Kodér však může kdykoliv celý slovník podřetězců smazat.

Způsob přiřazování a kódování podřetězců demonstrujme na následujícím příkladu:

Mějme zprávu: 1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4, 5, 6.
Maximální počet kódovaných znaků je 16 a tomu odpovídá délka kódového slova 4 bity.

[6]Název pochází ze slov Run Length Encoded a jedná se o datové struktury složené ze dvou čísel: hodnota a počet opakování.

V prvním kroku rozšíříme délku kódového slova na 5 bitů, symboly 0 až 15 jsou kořenové, nový symbol 16 dostane přidělenu značku *CLEAR* a symbol 17 budeme interpretovat jako *EOF*.

Prefix	Sufix	Výstupní symbol	Řetězec	Symbol řetězce
1	1	1	[1]1	18
1	2	2	[1]2	19
2	3	3	[2]3	20
3	4	4	[3]4	21
4	1	19 {1,2}	[4]1	22
19	3	21 {3,4}	[[1]2]3	23
21	1	23 {1,2,3}	[[3]4]1	24
23	4	4	[[[1]2]3]4	25
4	5	5	[4]5	26
5	6	6	[5]6	27
		17		

Tabulka 3.2: Ukázka způsobu kódování zprávy metodou LZW.

Popišme způsob tvorby kódu na tabulce 3.2. Každý řádek tabulky odpovídá jednomu kroku algoritmu. Sloupec prefix označuje symbol, který bezprostředně předchází aktuálnímu symbolu - jedná se vlastně o výstupní symbol z předchozí řádky. Ve sloupci sufix je zapsán aktuálně zpracovávaný symbol. Další sloupec obsahuje výstupní symbol. Ten může být buďto rozšířený (pokud se podařilo nalézt podřetězec přiřazený novému symbolu) a nebo kořenový (shodný se sufixem). Spolu s každým výstupním symbolem je generován řetězec a je přiřazen dalšímu novému symbolu. Číslo nového symbolu je obsaženo ve sloupci Symbol řetězce.

Jak je vidět z tab. 3.2, délka nezakódované zprávy dosahuje $18 * 4 = 60[bit]$ + EOF. Po zakódování se délka zprávy sníží na $11 * 5 = 55[bit]$.

Podstatnou výhodou popsaného algoritmu je značné rozšíření formátu GIF a jeho dekodérů. Poměrně jednoduchý kodér může provádět efektivní vyhledávání řetězců v historii.

Zde popisovaná varianta LZW dosahuje celkově poměrně špatného kompresního poměru. Vinu na tom má velké množství symbolů, které nemají po dlouhou dobu pevně přiřazen význam a se kterými je nutno stále počítat. Další nevýhoda se skrývá v narůstajícím slovníku podřetězců a v rostoucí délce kořenových symbolů. Poslední a též nezanedbatelná nevýhoda spočívá v patentové ochraně celého algoritmu a z ní vyplývajících omezení.

3.4 Základní myšlenky používané pro komprimaci dat

Všechny kompresní metody jsou založeny na poměrně jednoduchém předpokladu popřípadě na kombinaci více předpokladů. V této sekci je snaha analyzovat známé komprimační metody, extrahovat myšlenky v nich použité a uvést je samostatně. Tyto myšlenky jsou uváděny pouze v nejjednodušší podobě. Nechtěl bych tvrdit, že všechny myšlenky jsou stejně vhodné pro všechny třídy obrazových dat. Jedná se spíše o náměty, na čem všem lze postavit datový model. Aplikace každé myšlenky vede ke skupině komprimačních algoritmů.

Seznam základních myšlenek:

sousedi Sousední znaky mají přibližně stejnou číselnou hodnotu.

četnost Četnost (pravděpodobnost výskytu) znaků nebo symbolů se liší.

vzory Některé vzory nebo obrazce se mohou opakovat vícekrát.

derivace Derivace obrazové funkce ($df(x,y)/dx$ nebo $df(x,y)/dy$) je omezená.

oblasti V obraze se vyskytují přibližně homogenní oblasti.

vlastnost Určitá vlastnost je společná pro prvky příslušející do nějaké množiny dat.

global Je možno nalézt globální předpis pro výpočet celé obrazové funkce.

závislost Mezi sousedními prvky lze nalézt obecnou závislost.

Aplikace tvrzení *sousedi* vede přímo na RLC kód. Tento kód není příliš vhodný pro komprimaci obrázků vzhledem k tomu, že neodráží jejich 2D strukturu, což je ilustrováno v tab. 4.7. Kombinace myšlenek *četnost* a *vzory* vede na metodu LZH (popis viz sekce 3.5), která je dnes nejlepší používanou kompresní metodou pro 1-DOF data.

Tvrzení *derivace* implikuje fyzickou závislost ve vstupních datech. V podstatě lze říci, že tvrzení *sousedi* je jen speciálním případem *derivace*. U běžných[7] dat lze přibližně tvrdit, že vzájemná závislost klesá exponenciálně se vzdáleností dvou prvků.

IBM Q80 kodér (bezeztrátový JPEG) je založen na kombinaci *derivace* a *četnosti*. Dnes používané kvalitní kompresní algoritmy jsou nejčastěji založeny na myšlence *derivace* následované kombinací *četnost* a *vzory*.

[7] Tento termín bude ještě specifikován přesněji. Obecně lze říci, že za běžná data jsou v tomto kontextu považována data přenesená z reálného světa a nebo ručně kreslená bez speciálních efektů.

Tvrzení *oblasti* vede ke třídě pyramidových metod komprimace. Základní koncepce těchto metod je nastíněna v knize [HŠ92]. Diskusi o použití jedné z komplikovanějších technik založených na pyramidové kompresi je možno najít např. ve [Fry93].

Nejlepších výsledků kompresního poměru by teoreticky bylo možno dosáhnout při pochopení obsahu celé scény podle bodu *global*. Rozpoznanou scénu by bylo možno rekonstruovat předpisem vhodně zvolených elementárních operací. Podobným způsobem jsou vytvářeny vektorové obrázky. Tvorba takového popisu může být velmi složitá a je zvládnuta jen pro velmi jednoduché scény patřící do přesně vymezených kategorií.

Zobecněním myšlenky *sousedi* se dostaneme k myšlence *závislost*. Myšlenka *závislost* je prakticky používána u některých nelineárních prediktorů.

3.5 Známé bezeztrátové komprimační metody

V této sekci je uveden přehled autorovi známých algoritmů pro bezeztrátovou komprimaci obrázků. Přehled je doplněn stručným popisem algoritmů. Za popisem následuje způsob řazení algoritmů do jednotlivých tříd.

Následuje seznam nejznámějších komprimačních metod, které již byly experimentálně ověřeny. Každá metoda je stručně popsána a poté je diskutováno o jejích výhodách a nevýhodách.

PCX

PCX je formát grafických dat s pevně zabudovanou kompresí. Je zde využita poněkud modifikovaná 1-DOF metoda RLC (Run Length Coding). Obrazová data jsou nařezána na proužky odpovídající řádkům. Každý řádek je dále rozložen na tolik proužků, kolik má původní obrázek bitových rovin. Pouze v případě 8 nebo 24 bitů na pixel je řádek standardně rozkládán na proužky po 8 bitových rovinách. Každý proužek je komprimován odděleně. Jednotka, se kterou se v proudu dat vždy operuje, je jeden bajt. Označme nyní aktuální bajt $\{Data\}$. Opakuje-li se hodnota $\{Data\}$ n krát po sobě a platí-li ($n > 1$ nebo $\{Data\} > 191$) potom jsou do výstupního řetězce uloženy 2 hodnoty $\{n+190\}$ a $\{Data\}$. Není-li splněna tato podmínka, pak je aktuální bajt $\{Data\} < 191$ uložen beze změny. Více informací nalezne čtenář v knize [Dv94].

Výhody: Tento postup je velmi jednoduchý a vyžaduje pouze zpracování po bajtech. Dosahuje poměrně dobrých výsledků pro čárovou grafiku s malým počtem objektů.
Nevýhody: Kvalita komprese je velmi špatná. Model dat je založen na účelovém předpokladu, že pravděpodobnost čísel 0-191 je větší než 192-255.

LZH (obecná)

Tato metoda používá plovoucí okno[8] s historií. Kódovaná abeceda má n symbolů. V metodě jsou přidávány další symboly pro indexování řetězců v historii. Je-li podřetězec z aktuální pozice nalezen také v historii, pak je aktuální symbol zaměněn za index do historie a délku podřetězce. Všechny symboly jsou následně kódovány s využitím Huffmanova (nebo jiného prefixového) kódu. Pro podrobnější popis viz [Mel96].

Výhody: LZH je zatím nejlepším algoritmem pro 1-DOF data a jeho modifikace jsou dnes velmi často využívány v průmyslu (např. Stacker v 4.0 a PkZip).
Nevýhody: Tento algoritmus má v sobě obsažen pouze 1-DOF model dat. Pro rastrová obrazová data by měl existovat lepší datový model, který bude lépe modelovat jejich přirozenou strukturu.

IBM Q80 kodér (Bezeztrátový JPEG)

Kodér je založen na výpočtu diferencí obrazové funkce. Poté je výsledná matice diferencí komprimována s využitím prefixového kódu s pevným modelem dat. Model dat byl vytvořen na základě analýzy velkého množství testovacích obrázků. Další informace může čtenář nalézt v normě [ISO94].

Výhody: Ve standardu JPEG je implementována a dobře známa pouze ztrátová komprese. Umožnění a začlenění bezeztrátové komprese do tohoto standardu je dobrá myšlenka.
Nevýhody: V literatuře se uvádí, že algoritmus bezeztrátového JPEG kodéru je v současné době zastaralý a není podporován většinou dostupných implementací knihoven pro práci s obrázky. Statický model, který je zde použit, dává dobré výsledky jen pro velmi úzkou množinu dat. Proto využití Q80 kodéru v současné době nepředstavuje příliš šťastné řešení.

PNG grafika přenositelná po síti (Portable Network Graphics)

Stručný popis čtenář nalezne např. v [Kol97]. V případě hlubšího zájmu o problematiku je lépe nahlédnout do technické dokumentace [Tea96]. Algoritmus se skládá ze dvou částí. Každá část představuje jeden krok při zpracování dat. V prvním kroku je vypočtena matice diferencí $\Delta_x(x,y) = f(x-1,y) - f(x,y)$. Alternativně mohou být v tomto kroku užity diference $\Delta_y(x,y) = f(x,y-1) - f(x,y)$, Δ_{xy} nebo jednoduchý prediktor Pægas. Ve druhém kroku je matice kódována s využitím metody LZW. Obě části mohou zařazeny do série, protože paralelní

[8] Plovoucí okno je datová struktura, která uchovává informace o předchozích datech do určité hloubky. Struktura je využívána plně adaptivním algoritmem k jeho nastavování.

procesy nepotřebují vzájemnou zpětnou interakci. Tím lze umožnit paralelní běh na víceprocesorových systémech.

Výhody: Předpokládám, že PNG je dnes nejnadějnější metoda na dlouhodobější rozšíření. Zdrojový kód použitých algoritmů je volně k dispozici a účinnost komprese je srovnatelná s ostatními komprimačními algoritmy typu GIF.
Nevýhody: Zejména druhý krok není specializován na 2D obrazová data. Komprimace u metody PNG těží z předpokladu, že velikost oblastí s omezenou 1. derivací je menší než velikost oblastí s velkou 1. derivací. Podle mého názoru obsahuje 2D struktura obrázků mnohem více využitelných redundancí.

MLP - postupně po více úrovních (Multi Level Progressive)

Tato metoda je typickým příkladem pyramidová komprese. Detailní popis je obsažen v [Fry93]. Metoda je založena na rozdělení celého rastru do malých disjunktních čtvercových oblastí. Pro každou oblast je vypočtena charakteristická hodnota (např. průměr, medián atd.) a pomyslně uložena nad danou oblast. Tím se vytvoří malá pyramida. Všechny vrcholky[9] malých pyramid tvoří další bitovou mapu. V rastru vyšší úrovně jsou též vytvořeny malé disjunktní oblasti a vypočítány jejich vrcholky. Výpočet se zastaví v okamžiku dosažení hodnoty jediného vrcholu. Rezidua jsou vypočtena jako rozdíly mezi vrcholkem z vyšší úrovně a daty z aktuální úrovně ležící těsně pod vrcholkem. Všechna rezidua jsou následně kódována.

Výhody: Tato metoda poměrně dobře odráží přirozenou 2D strukturu obrazových dat. Při vhodné konfiguraci pyramid je možno prohlížet i neúplná data.
Nevýhody: Výpočet pyramid je velmi náročný na paměť a rovněž nároky na výkon počítače jsou poměrně vysoké. Dosažený kompresní poměr v experimentech byl poměrně nízký.

Segment

Metoda segment patří ke skupině metod založených na detekci a odděleném zpracování oblastí. Popis metody byl publikován v [RA96]. Obraz je rozdělen na oblasti s podobnou intenzitou. Pro segmentaci na oblasti je použita běžně známá metoda narůstání oblastí (ze semínka) popsána např. v [HŠ92]. Jednotlivé oblasti jsou vymezeny lomenými hraničními čárami popisujícími jejich hranice. Data z oblastí jsou komprimována odděleně. Do výsledných dat je navíc přidán popis hraniční linie.

[9] Termín vrcholek označuje lokální vrchol malé pyramidy. Pro nejvyšší bod celé pyramidy je ponechán termín vrchol.

Výhody: Metoda dává velmi dobré výsledky na publikované testovací sadě obrázků v [RA96] a je možné očekávat, že srovnatelných výsledků bude dosaženo i pro jiné obrázky.
Nevýhody: Klasifikace obrazu na oblasti má obrovské nároky na výpočetní kapacitu. Je vyžadován nesekvenční přístup k obrazovým datům. Rozdělení do oblastí není jednoznačné a závisí do značné míry také na poloze semínek (počátku segmentace) tedy ne pouze na obrazových datech.

Calic

Komprimační metoda Calic [WM97] patří do třídy metod, které jsou založeny na činnosti prediktoru. Prediktor si vytváří interní model dat na základě několika příznaků. Tyto příznaky jsou počítány z pixelů, které se nacházejí v malém okolí predikovaného bodu. Za příznaky jsou například používány směrové derivace. Aktuální predikovaný pixel je adaptivně odhadován na základě předchozích příznaků. Rezidua, která vznikla rozdílem odhadnuté a aktuální hodnoty, jsou kódována buď Huffmanovým nebo aritmetickým kodérem. Poté jsou uložena do výstupního souboru.

Výhody: Úspěšnost komprimace je u metody Calic podobná jako u metody navrhované v této práci. Pro komprimaci/dekomprimaci není potřeba znát všechna obrazová data najednou. Zpracování dat je plně sekvenční.
Nevýhody: Navrhovaná metoda je velmi dobře nastavena pouze pro šedotónové obrázky s 256 úrovněmi šedi. Ostatní typy rastrových obrázků nejsou zatím podporovány.

Kruhová transformace - Rounding transform

Jedná se o další příklad pyramidového přístupu. Kruhová transformace, citovaná v [JCP96], používá hierarchický estimátor ve tvaru kruhu. Osm pixelů okolo prostředního pixelu vytváří základní element - kroužek. Z kroužku je vypočítán příznak. Celý postup je opakován pro všechny pixely základní bitové mapy, které představují nejnižší úroveň. Následuje postup do vyšší úrovně, do které je vybráno menší množství bodů. Jedná se o výběr typu každý x-tý pixel. V nižší úrovni je matice příznaků transformována na matici diferencí od prostředního elementu dané skupiny. Takto je postupováno pro všechny vrstvy pyramidy. Celá metoda je založena na předpokladu, že se daný příznak mění v malé oblasti (lokálně) jen velmi málo. Autoři [JCP96] dokázali, že k popisované transformaci dat existuje jednoznačná inverzní transformace.

Výhody: Navrhovaný příznak je originální.
Nevýhody: Uvedená metoda se vzhledem k dosaženému kompresnímu poměru neřadí mezi nejlepší. Je překonána většinou metod uvedených v tomto oddílu.

Logické kódování

Autoři článku [CAJ96] se pokusili založit komprimační algoritmus na Booleově logice. Komprimovaný obrázek je rozdělen na jednotlivé bitové roviny. Pro každou bitovou rovinu je vytvořen binární logický prediktor, který je v následujícím kroku optimalizován na základě Booleovy algebry.

Logické kódování je svojí činností blízké navrhované metodě popsané v kapitole 4

Výhody: Teoreticky je možno provádět jakékoliv logické operace velmi rychle pomocí specializovaného hardware.
Nevýhody: Logická funkce nepředstavuje dobrý formalismus pro modelování obrazových dat. Navíc je informace z vyšších bitových rovin v prezentované implementaci ztrácena, což se negativně projevuje na dosahovaném kompresním poměru.

Víceúrovňové komprimační schéma

Víceúrovňové komprimační schéma patří mezi pyramidově orientované algoritmy. Jedná se o poměrně obecnou metodu. Jedna z provedených implementací je popisována v [PM96]. Dále existují úpravy popisovaného algoritmu používající vlnky (wavelets) [Kop95].

Při komprimaci je obraz v každém kroku rozdělen na dvě poloviny. Rozdělování je prováděno střídavě vodorovně a svisle. Do první poloviny přechází každý lichý řádek (sloupec) a ve druhé polovině zůstane každý sudý řádek (sloupec). Jedna z polovin je označena za základní a podle ní jsou predikována data z poloviny druhé. Namísto dat jsou do druhé poloviny ukládána rezidua jako rozdíly predikované a aktuální hodnoty. Nejjednodušší prediktor je založen na předpokladu, že pixel v první polovině má přibližně stejnou hodnotu jako pixel ležící na korespondující pozici ve druhé polovině.

Po uložení reziduí je zbylá část obrazu znovu rozdělena na dvě poloviny a celá operace se opakuje. Data z jedné poloviny jsou predikována pomocí dat z poloviny druhé. Druhá polovina je opět převedena na rezidua. Zbylá část původního obrázku se stále zmenšuje a ostatní obrazová data se mění na rezidua. Rezidua je možno kódovat standardními způsoby např. LZW.

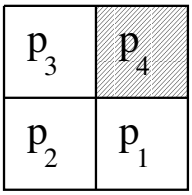
Výhody: Jedná se o pyramidový přístup s možností zobrazování jen části dat a jejich postupné aktualizace.
Nevýhody: Při výpočtu rozdílů se pracuje jen s jednoduchými diferencemi, které nemohou zachytit příliš mnoho informací v datech obsažených. Tomu pak odpovídá i nižší kompresní poměr dosahovaný na třídě reálných obrázků.

3.6 Binární Schlesingerův prediktor

Celá naše komprimační metoda je založena na transformaci binárních rastrových obrázků. Tato technika byla vytvořena prof. Schlesingerem z Ukrajiny [Sch89] a bude v následujícím textu odkazována zkratkou *FH-2-DOF* . Předpokládáme, že není běžně známa, a proto jejímu popisu bude věnováno více místa. Původní notace byla upravena z důvodů použití jednotného formalismu pro popis dalších navrhovaných technik.

Metoda *FH-2-DOF* předpokládá na svém vstupu binární obraz f. Nosič obrazu je označen symbolem T=$\{(x, y)$: $0 \leq x \leq M$; $0 \leq y \leq N\}$, kde M je šířka obrazu a N představuje jeho výšku. Binární obraz je tvořen zobrazením $f(x, y) \rightarrow \{0, 1\}$. Bez újmy na obecnosti můžeme předpokládat, že levý sloupec a spodní řádek jsou vyplněny nulami, tedy $f(0, y) = f(x, 0) = 0$. V případě, pokud tomu tak není lze vždy obraz rozšířit tak, aby tato podmínka splněna byla. Po dekompresi lze opět přidanou část oddělit od obrazu.

Předpokládejme, že levý sloupec a spodní řádek ve vstupním obrázku jsou rovny 0 pro zjednodušení dalších výpočtů. Teoreticky by mohly být nenulové. V tom případě musí zůstat nekomprimovány při zpracování celého rastru. Dodatečně je možno je komprimovat jinou kompresní technikou např. RLC.

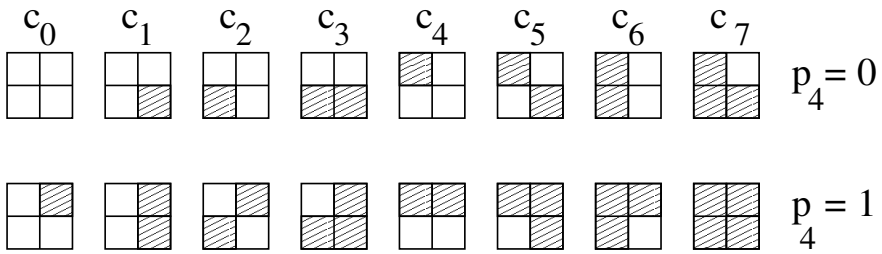


Obrázek 3.4: Testovací sonda 2×2.

Metoda *FH-2-DOF* je založena predikci 2D binárním prediktorem a následné transformaci binární bitové mapy f na rezidua. Prediktor, který prochází celý obraz f, je tvořen sondou skládající se z 2×2 buněk. Reprezentativní (aktuální) buňka[10] (x, y) je umístěna v pravém horním rohu sondy a její hodnota je $p_4 = f(x, y)$. Další buňky sondy jsou označeny $p_1 = f(x, y - 1)$; $p_2 = f(x - 1, y - 1)$; $p_3 = f(x - 1, y)$; viz obr. 3.4.

M. Schlesinger navrhl prediktor (používaný jako estimátor se sondou) e následujícím způsobem. Všech 16 možných kombinací buněk v sondě o velikosti 2×2 uspořádal do dvou řádek tak, aby se sondy umístěné v jednou sloupci lišily právě hodnotou jediného bitu odpovídajícímu horní pravé buňce. Takto uspořádané sondy jsou zobrazeny na obr. 3.5.

[10]Slovo buňka je použito místo slova pixel. V kontextu označuje elementárnější datovou strukturu než pixel tzn. pixel se může skládat z více buněk. Zavedení buňky umožňuje lepší přenesení úvah do vícerozměrného prostoru bez zavedení nechtěných předpokladů o jeho dimenzi. Pixel $\sim$ 2D; voxel $\sim$ 3D atd.



Obrázek 3.5: Všechny možné konfigurace sond pro prediktor s velikostí sondy 2×2. Sondy umístěné ve stejném sloupci se liší pouze hodnotou horního pravého pixelu (buňky).

Prediktor lze použít několika způsoby. Buďto s pevným datovým modelem. V takovém případě stačí přímo komprimovat. Prediktivní semiadaptivní modifikace metody vyžaduje pro komprimaci bitové mapy dva průchody obrazovými daty. Ke komprimační metodě samozřejmě existuje inverzní dekomprimační metoda. Ve všech dalších rozšířeních jsou komprimační částí prováděny alespoň tyto činnosti:

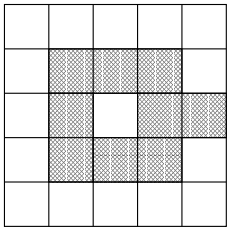
A. Návrh optimálního prediktoru je založen na tzv. analýze četností jednotlivých konfigurací sond. Četnosti sond jsou získány prvním průchodem obrazovou funkcí f.

B. Redukce dat na jejímž výstupu je řídká matice obsahující seznam reziduí.

C. Redukovaná data jsou optimálně kódována a společně s popisem prediktoru vytvářejí výsledný komprimovaný obrázek.

3.6.1 Analýza četností

Pro návrh optimálního prediktoru je použita *analýza četností* konfigurací sondy v obrázku. Analýza četností vyžaduje jeden průchod obrazovými daty a může být provedena paralelně mnoha procesory. Výsledkem je tabulka četností jednotlivých sond.



Obrázek 3.6: Hypotetický vstupní binární obrázek.

Celý výpočet je názorně ukázán na malém hypotetickém vstupním obrázku obr. 3.6. Tabulka četností pro sondu podle obr. 3.4 má dva řádky a osm sloupců viz tab. 3.3. Tabulka tab. 3.3 byla vytvořena pro obr. 3.6.

	c_0	c_1	c_2	c_3	c_4	c_5	c_6	c_7
$p_4 = 0$	0	1	1	2	1	0	0	2
$p_4 = 1$	1	2	0	1	2	1	2	0

Tabulka 3.3: Tabulka četností sond pro hypotetický obrázek.

$$\hat{p}_4 = e(p_1, p_2, p_3) \tag{3.5}$$

Prediktor e definovaný podle (3.5) odhaduje hodnotu obrazové funkce v aktuálním bodě $\hat{p}_4$ ze tří sousedních buněk p_1, p_2, p_3 viz též obr. 3.4. Za funkci e lze považovat libovolnou binární funkci proměnných p_1, p_2, p_3. Definujme pokutovou funkci g prediktoru (3.6), která pouze ověřuje shodu odhadnuté hodnoty s hodnotou predikovanou. Jedná se vlastně o nonekvivalenci.

$$g(\hat{p}_4, p_4) = \begin{cases} 0 & \text{pro } \hat{p}_4 = p_4, \\ 1 & \text{pro } \hat{p}_4 \neq p_4. \end{cases} \tag{3.6}$$

Tvorba optimálního prediktoru může být formulována jako optimalizační úloha, která je založena na minimalizaci hodnoty pokutové funkce. Pokutová funkce sčítá počet reziduí v celém obrázku. Rezidua vznikají v místech, kde dává prediktor nesprávnou hodnotu. Základní myšlenkou prof. Schlesingera je uchovat pro další zpracování pouze rezidua.

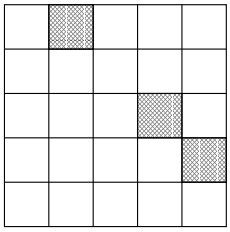
$$e^* = \operatorname{argmin} \min_e \sum_{x=1}^{M} \sum_{y=1}^{N} g(e(p_1, p_2, p_3), p_4). \tag{3.7}$$

Prediktor e je podle (3.7) navržen tak, že predikuje pouze četnější konfigurace. Funkce e má v každém sloupci tabulky četností jeden stupeň volnosti, protože v rámci jednoho sloupce jsou hodnoty sousedních buněk p_1, p_2, p_3 stejné.

V každém sloupci tabulky četností postačí nalézt a označit maximální hodnotu a jí odpovídající konfiguraci sondy. Tím lze jednoznačně definovat funkci e prediktoru.

Pro zakódování funkce prediktoru je potřeba 1 bitu pro každý sloupec. Podle jeho hodnoty lze jednoznačně poznat, která z obou konfigurací sondy je četnější. Pro uchování informace o všech osmi maximech (8 sloupců) postačí jeden bajt.

Hodnota $11001110b$ obsahuje zakódovaná maxima z tabulky četností podle tab. 3.3.



Obrázek 3.7: Rezidua vypočtená z obr. 3.6.

3.6.2 Tvorba reziduí

Reziduální bitová mapa vzniká jako rozdíl predikované a aktuální hodnoty ve všech bodech obrazové funkce. Funkce prediktoru e je výsledkem předchozího kroku analýzy četností. Nedojde-li ke shodě predikované a aktuální hodnoty v nějakém bodě $\hat{p}_4 = p_4$, pak je nutno na jeho místo zapsat reziduum (označujeme jej hodnotou 1). V opačném případě je zapsána hodnota 0.

Bitová mapa z testovacího obr. 3.6 je transformován na reziduální bitovou mapu viz obr. 3.7. Popis prediktoru (jeden bajt) je potřeba přidat ke komprimovanému obrázku za účelem pozdějšího obnovení dat.

Je možno též použít vylepšený algoritmus (Popisu vylepšení algoritmu je věnována samostatná sekce 4.7.), kterému postačí pouze jediná bitová mapa. Její obsah lze postupně transformovat na reziduální bitovou mapu. V tomto případě je nutno obraz procházet přesně definovaným způsobem. Pro 1. sloupec a 1. řádek (mají index 0 ve funkci $f(x, y)$) vyplněný nulami je správný směr zprava-doleva a po řádcích shora-dolů. Způsob procházení může být změněn, avšak poté musí být též změněna pozice nepredikovaných (vynulovaných) částí okraje obrazu a navíc též tvar prediktoru.

3.6.3 Rekonstrukce původních dat

Z bitové mapy obsahující rezidua a popisu prediktoru lze jednoznačně vytvořit původní obraz. Algoritmus, který provádí tuto činnost (o dekomprimaci se zde zatím nejedná, protože reziduální bitová mapa má shodnou velikost s původní bitovou mapou) má k dispozici popis prediktoru a reziduální bitovou mapu. V ní musí zůstat první řádek a první sloupec v původní podobě.

Sonda je umístěna na první pozici vlevo dole. Hodnoty jejích buněk p_1, p_2, a p_3 jsou známy, poněvadž uvedené buňky leží v okrajové oblasti, která byla vynulována. Tyto tři hodnoty postačí prediktoru e k výpočtu $\hat{p}_4$. Není-li na aktuální pozici uloženo reziduum, pak platí $p_4 = \hat{p}_4$. V opačném případě je použita opačná hodnota $p_4 = \overline{\hat{p}_4}$.

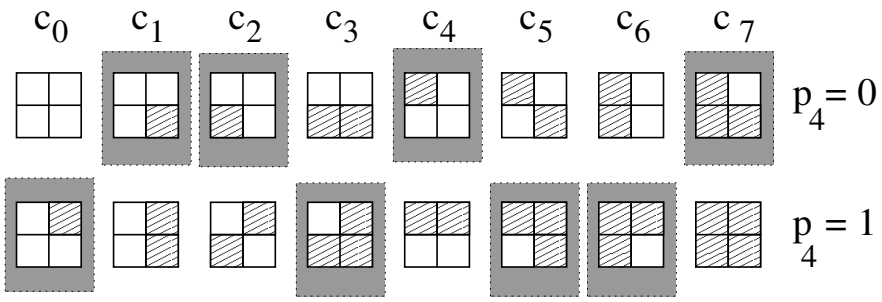
Reziduální bitová mapa je procházena sondou po řádcích zleva doprava a zdola nahoru dokud není obnoven původní obsah obrazu. Směr procházení je opačný oproti případu při tvorbě reziduí. I v případě rekonstrukce není potřeba

nová bitová mapa a data je možno ukládat v průběhu činnosti algoritmu do stejné bitové mapy, ve které byla uložena rezidua.

3.6.4 Koutečková transformace

Poznamenejme, že prof. Schlesinger [Sch89] postavil své 2D gramatiky tak, že zpracovávají seznam reziduí. Rezidua jsou vytvářena pevnou konfigurací sond viz obr. 3.8. Toto nastavení prediktoru je suboptimální z hlediska pohledu analýzy četností pro většinu případů binárních rastrů.

Rezidua jsou ukládána do seznamu x, y souřadnic. Na sadě reziduí vytvořených koutečkovým prediktorem je možno provádět standardní binární operace (AND, OR, XOR, NOT) bez potřeby provádět dekomprimaci (zde se již o komprimaci/dekomprimaci jedná, poněvadž rezidua jsou uložena ve zhuštěném tvaru).



Obrázek 3.8: Schlesingerova koutečková konfigurace: *méně* četné konfigurace-koutečky jsou orámovány.

Za zmínku stojí, že na obr. 3.8 jsou úmyslně zvýrazněny konfigurace, které jsou pevně nastaveny jako méně četné. V místech jejich výskytu se v reziduální bitové mapě objeví rezidua. Méně četné konfigurace svým tvarem připomínají tzv. koutečky. Podle nich je celá transformace nazývána koutečková. Podrobné analýze koutečkové transformace je věnována kapitola 6.

3.7 Ztrátová komprimace

Vedle komprimace bezeztrátové existuje dále komprimace ztrátová. Protože si neklade za cíl úplné obnovení původních dat, lze dosáhnout za cenu ztráty dat řádově vyšších kompresních poměrů. Protože většina algoritmů umožňuje nastavit velikost ztráty, je pro srovnávání kvality dvou ztrátových komprimačních algoritmů nutno spíše porovnávat funkce *kvalita=f(ztráty)*. Kritéria pro posuzování kvality jsou většinou subjektivní, což ztěžuje přesné ohodnocení a vzájemné srovnávání jednotlivých algoritmů ztrátové komprese.

Jedním z jednodušších ztrátových kompresních algoritmů je Diferenciální pulsně kódovaná modulace (Differential Pulse Coded Modulation zkratka DPCM)

s omezenou šířkou pásma [HS94]. Kódování DPCM je prováděno následujícím způsobem: Nejprve se vypočte diference číselných hodnot dvou sousedních pixelů. Ztrátovost spočívá v omezení rozsahu diferencí. Číselné ohodnocení jednotlivých pixelů v obrazu může nabývat hodnot v rozsahu $\langle min; max \rangle$. Pro zakódování jednoho pixelu je potřeba určitého počtu bitů. Pro zakódování diferencí bude zvolen počet bitů o něco menší. Výsledkem tohoto postupu je, že dojde k omezení diferencí sousedních bodů a tím též k rozmazání strmých hran, jejichž výška překračuje rozsah $\langle minDif; maxDif \rangle$. Všechna místa, kde nedochází k velkým změnám jasu budou zakódována správně. Samozřejmě je nutno při 'ořezání' aktuální diference upravit několik diferencí následujících, jinak by došlo k posunu absolutní hodnoty jasu ve všech následujících bodech. Uvedený algoritmus je pro svou jednoduchost využíván v televizní technice.

Další též velmi jednoduchou a účinnou ztrátovou kompresní techniku představuje pouhé podvzorkování. To znamená zařazení každého n-tého pixelu mezi výstupní data. Při dekompresi jsou chybějící data doplněna podle nejbližšího uloženého pixelu.

Jako u bezeztrátových algoritmů existují i u algoritmů ztrátových složitější metody druhé generace. Jmenujme nejznámější z nich:

A. Fraktálová komprese.

B. JPEG představuje již několik let standard ztrátové komprese zejména pro barevná obrazová data. Obecné pojednání lze nalézt např. v [ISO93]. Podrobnější informace jsou obsaženy v normě ISO/IEC 10910 [ISO94].

C. Wavelet (vlnka).

Kapitola 4

Návrh bezeztrátové komprimace rastrových obrázků

4.1 Myšlenky navržené bezeztrátové komprese

O zdrojích redundance v obrazových datech bylo pojednáno v oddíle 3.4. Jedním z nejsilnějších, a proto též v kompresních algoritmech nejčastěji využívaným zdrojem redundance je skutečnost, že sousední buňky mají s největší pravděpodobností stejnou nebo blízkou hodnotu. (Viz tvrzení *sousedi* ve výpisu základních myšlenek uvedeném v oddílu 3.4 na stránce 23). Z toho vyplývá, že je vhodné analyzovat lokální okolí aktuální buňky.

Schlesingerův kompresní algoritmus používal jednoduchou malou sondu o velikosti 2×2 buněk viz obr. 3.4. Podle četností jednotlivých konfigurací sond je navržen prediktor. Data zahrnutá v sondě jsou využita k predikci aktuální buňky. Celá metoda je popsána v sekci 3.6. V této sekci je uveden stručný popis nově navrhovaného komprimačního algoritmu následovaný podrobným rozborem celé metody.

Podařilo se mi dále rozšířit Schlesingerovu myšlenku týkající se prediktivního kódování. Obrazový rastr je systematicky procházen sondou a data z jednotlivých sond jsou využívána pro četnostní analýzu. Výsledky četnostní analýzy jsou použity k návrhu prediktoru. V každém místě rastru je aplikován prediktor na obrazová data. Z okolních dat je určena hodnota aktuálního pixelu. Pokud nedojde ke shodě predikované a aktuální hodnoty je buňka na aktuální pozici označena jako reziduum (poznamenávám, že se stále jedná o binární rastr a hodnotou 0 může být označena shoda, kdežto 1 reziduum).

Kvalitu prediktoru lze měřit počtem reziduí. Prediktor je natolik dobrý, kolikrát dojde k přesné shodě mezi odhadnutou a skutečnou hodnotou buňky. Protože je uvedená technika semiadaptivní, je do výsledného datového souboru uložen popis prediktoru spolu se zakódovaným popisem pozic reziduí v binárním rastru.

4.2 Zobecnění prediktoru pro data z šedotónových obrázků

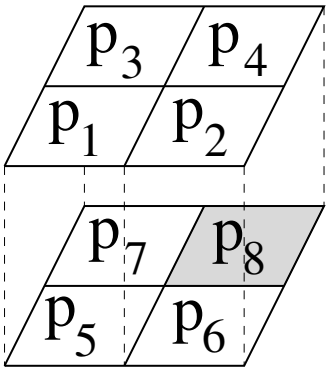
Podle [Spe69] může být každý šedotónový rastr rozložen na bitové roviny, které jsou uloženy nad sebou. Prediktor byl zobecněn tak, aby mohl predikovat nejen z aktuální bitové roviny, ale i z nejbližší vyšší bitové roviny viz obr. 4.1. Obrázek je z ilustrativních důvodů nakreslen prostorově metodou volného rovnoběžného promítání. Tím je umožněno zobrazit současně obě bitové roviny ležící nad sebou.

4.2.1 Prediktor pevného tvaru

Protože sonda může zaznamenat data ve třech rozměrech a prediktor data ze sousední vyšší bitové roviny aktivně používá, je symbol 3-DOF obsažen v názvu prediktoru. V dalším textu je tento prediktor odkazován *FH-3-DOF*. (Písmena F, H byla vzata podle příjmení autorů Fojtík a Hlaváč.)

V dané úvaze lze pokračovat a prediktor může být rozšířen do ostatních bitových rovin. Sonda může například zasahovat do sousedních barevných složek ve vícespektrálních datech (v nejjednodušším případě se jedná o RGB data). Jediné omezení tvaru prediktoru vyplývá z podmínky tzv. úhlu prediktoru, která byla uvedena v oddílu 3.2.4 na stránce 17. Při nesplnění uvedené podmínky by nebylo možno zpětně rekonstruovat data kauzálním rekonstruktorem. Nekauzální rekonstruktor obrazových dat by byl s největší pravděpodobností velmi složitý a zatím se mi nepodařilo dokázat, zda by bylo možno takto predikovaná data rekonstruovat jednoznačně.

4.2.2 3-DOF sonda



Obrázek 4.1: 3-DOF prediktor

V základní variantě FH-3-DOF prediktoru uvažujme sousední buňky ve

vzdálenosti[1] 1 od aktuální buňky viz obr. 4.1.

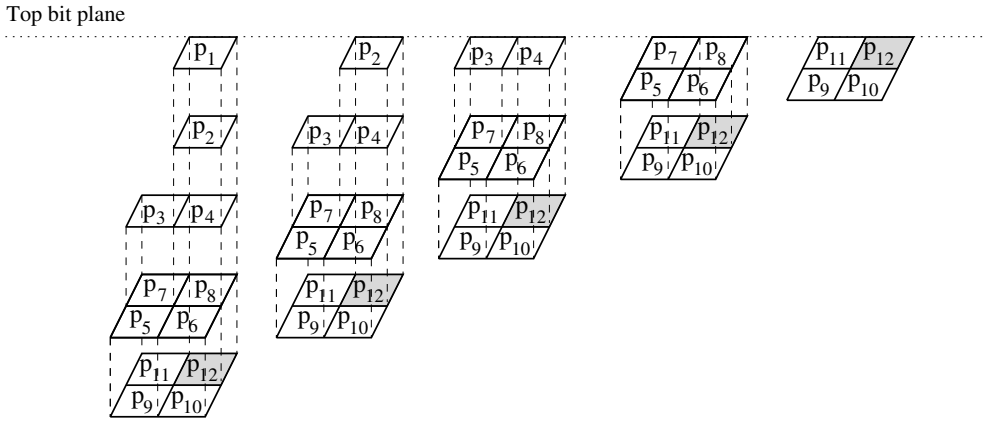
Definice optimalizační úlohy pro 3-DOF prediktor e, viz rovnice (4.1), je velmi podobná optimalizační úloze uvedené v předchozím případě (3.7) 2-DOF prediktoru. Funkce g je definována podle (3.6). Predikovaná buňka musí mít nejvyšší index, abychom se vyhnuli problémům s indexy v následných výpočtech.

$$e^* = \text{argmin} \min_e \sum_{x=1}^{M} \sum_{y=1}^{N} g(e(p_1, p_2, p_3, p_4, p_5, p_6, p_7), p_8). \tag{4.1}$$

Definice optimalizační úlohy je totožná pro všechny bitové roviny s malou výjimkou u nejvyšší bitové roviny. Při položení 3-DOF sondy do nejvyšší bitové roviny by tato sonda měla buňky p_1, p_2, p_3, p_4 nedefinovány. Úpravu pro nejvyšší bitovou rovinu lze zařídit položením nulové bitové roviny nad nejvyšší bitovou rovinu.

Je snadno dokazatelné (s využitím informací podle sekce 4.4), že při vložení prázdné bitové roviny nad libovolnou aktuálně zpracovávanou bitovou rovinu, navrhovaný 3-DOF prediktor degraduje v dané bitové rovině na původní 2-DOF Schlesingerův prediktor.

Frekvenční analýza konfigurací sond je analogická předchozímu 2-DOF případu. Pro uložení informací z celé tabulky četností je v případě sondy o velikosti osmi buněk potřeba 2^7 bitů = 16 bajtů. Obraz složený z bitových rovin je převeden na obraz reziduí, který lze ve většině případů přirovnat k řídké matici. Rezidua mohou být kódována v rámci každé bitové roviny stejným způsobem jako rezidua pocházející z prediktoru *FH-2-DOF*. Kódování vyžaduje další průchod obrazovými daty následující po analýze četností.



Obrázek 4.2: Sada sond prediktoru pro více bitových rovin.

[1] Vzdálenost dvou buněk je definována počtem přechodů mezi buňkami, včetně přechodů po úhlopříčkách, které leží na nejkratší cestě mezi oběma dotazovanými buňkami.

4.2.3 Proměnný tvar prediktoru

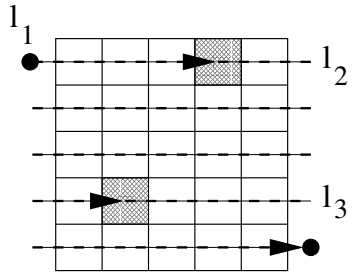
Pro více bitových rovin je velmi výhodné využít i informací z bitových rovin umístěných nad aktuální bitovou rovinou. Pro tento typ dat byl vyvinut jiný tvar sondy podle obr. 4.2. Jedná se o množinu podobných sond pro jednotlivé bitové roviny ležící pod sebou. Sonda nejvíce napravo je použita pro zpracování nejvyšší bitové roviny. Sonda o jedno nalevo je použita pro kódování první bitové roviny pod touto bitovou rovinou atd.

Pro kódování 3D dat lze použít pouze 2D sondy (za cenu většího množství reziduí) podle obr. 3.4 a nebo jednoduché 3D sondy podle obr. 4.1 pro všechny bitové roviny. Sada sond podle obr. 4.2 dosahuje vzhledem k větší velikosti sond lepší kvality prediktoru. Všechny zobrazené sondy lze jednoduše odvodit od jediné sondy za předpokladu, že všechny bitové roviny ležící nad nejvyšší bitovou rovinou jsou nulové. Nutnost vytvářet sondy pro jednotlivé bitové roviny odstraňuje až adaptivní prediktor (viz sekce 4.4), který umí automaticky vyjímat buňky ze sondy. Bez použití adaptivního prediktoru je vhodné vytvořit pro každou bitovou rovinu novou sondu.

Dosud byly jednotlivé sondy pro jednotlivé bitové roviny odvozeny od jediné základní sondy, která se "zanořovala" do jednotlivých bitových rovin. Dalšího zvýšení kvality komprese by bylo možno docílit použitím naprosto odlišných sond pro jednotlivé bitové roviny. Mělo by tím dojít ke zmenšení počtu reziduí za cenu zvýšení složitosti kompresního algoritmu. Musel by totiž být vytvořen prediktor a dekodér pro každou bitovou rovinu zvlášť. Na druhou stranu v některých aplikacích zvýšení složitosti nemusí představovat žádnou překážku a může být akceptovatelné vzhledem k požadavku na co možná nejvyšší kompresní poměr.

4.3 Kódování reziduí

Výsledkem předchozího kroku je reziduální bitová mapa popř. více bitových map odpovídajících jednotlivým bitovým rovinám. Reziduální bitová mapa obsahuje velmi malý počet reziduí (v naší implementaci kódována hodnotou 1) a lze ji přirovnat k řídké matici. Avšak dosud se o žádnou kompresi dat nejednalo.



Obrázek 4.3: Vzdálenosti pro kódování.

Pro dosažení vysokého kompresního poměru musí být reziduální bitová mapa efektivně kódována. Ukázka takovéto jednoduché reziduální bitové mapy je uvedena na obr. 3.7. Jedno z možných kódování představuje seznam všech bodů, který obsahuje jejich $[x, y]$ souřadnice. Toto řešení bylo použito prof. Schlesingerem a v některých případech usnadňuje následnou manipulaci se seznamem reziduí.

Pro dosažení vyššího kompresního poměru byl zvolen jiný způsob kódování. Nový způsob je založen na seznamu vzdáleností jednotlivých reziduí viz. obr. 4.3. Měření vzdáleností začíná z hypotetického startovacího bodu, který je umístěn nalevo od prvního sloupce v prvním řádku. Za vzdálenost je považována délka linie, měřená v počtu buněk, mezi předchozím a aktuálním reziduem. Sousedním bodem posledního bodu na řádku je první bod ležící v následujícím řádku. Měření je ukončeno v hypotetickém bodě umístěném napravo za posledním řádkem. V naší implementaci lze měření vzdáleností ukončit též v posledním bodě poslední řádky, vyskytuje-li se v něm reziduum. Výsledkem měření vzdáleností je seznam vzdáleností všech reziduí. Jednoznačná závislost mezi pozicí rezidua a vzdálenostmi $l_1, \ldots, l_n$ je vystižena v rovnici (4.2).

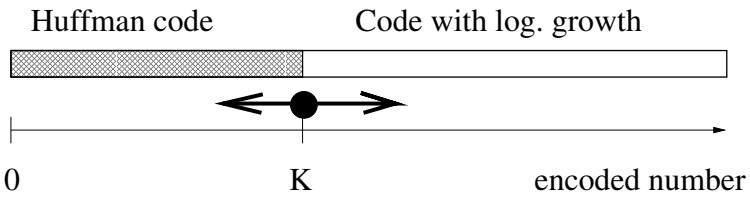
$$\begin{aligned} x_k &= (\sum_{i=1}^{k} l_i) \; mod \; M \\ y_k &= (\sum_{i=1}^{k} l_i) \; div \; M \end{aligned} \tag{4.2}$$

Navíc pro každý seznam vzdáleností korespondující s nějakým rozmístěním reziduí musí být splněna podmínka daná rovnicí (4.3). Tato podmínka může být použita pro kontrolu chyb, které mohou nastat při nahrávání kódovaných vzdáleností z méně spolehlivého média jakým je například disketa.

$$\sum_{i=1}^{n} l_i = M\,N + 1. \tag{4.3}$$

4.3.1 Kód s proměnnou délkou kódového slova

Bez efektivního kódování vzdáleností $l_1, \ldots, l_n$ by nebylo možno dosáhnout vysokého kompresního poměru. Proto je nutno se věnovat kódování podrobněji. Jednotlivé vzdálenosti představují symboly abecedy. Každému symbolu chceme přiřadit kódové slovo tak, aby celková délka zprávy byla nejkratší. Teoreticky je velikost zprávy kódované Huffmanovým kódem nejkratší poněvadž se vychází z četnosti jednotlivých symbolů. Přímé použití Huffmanova kódování však v našem případě není vhodné, protože maximální kódovaná vzdálenost může teoreticky dosáhnout hodnoty $l_{max} = M\,N + 1$. Toto číslo je pro výpočet Huffmanova kódu poměrně velké, protože je nutno uvažovat všechna čísla v rozsahu $\langle 1; l_{max} \rangle$.



Obrázek 4.4: Rozdělení kódované oblasti na dvě části.

Nejlepší algoritmy mohou vypočítat Huffmanův kód v čase[2] $O(n\,log(n))$ a v podstatě je výpočetní náročnost zdola limitována složitostí třídění položek. Následný výpočet Huffmanovy tabulky na setříděných datech lze udělat v lineárním čase $O(n)$. Číslo $O((M\,N+1)log(M\,N+1))$ je ve většině případů nepřijatelně vysoké.

Z tohoto důvodu byl použit jiný kód pro reprezentaci celých kladných čísel. Existuje několik typů kódů použitelných pro náš případ. Podle [Mel96] připadají v úvahu zejména tři typy kódů *Fibonacci*, *Elias* a "ω". Kód "ω" je pro náš případ nejméně vhodný, protože velikost kódového slova se zvyšuje velmi rychle se zvětšujícím se kódovaným číslem. Velká čísla je v našem případě potřeba kódovat velmi často. Fibonacciho kód se jeví jako úspornější, ale jeho výpočet je poměrně složitý.

Z těchto důvodů jsem se nechal inspirovat Eliasovým kódem. Eliasův kód se skládá ze dvou částí: První unární[3] část kóduje délku druhé binární části. Z provedených experimentů se ukázalo, že prvních několik čísel (typicky 1,2,3 ...) se v datech vyskytuje nejčastěji. Proto byl pro naše účely z Eliasova kódu odvozen kód nový. Nový kód jsme nazvali *kód s logaritmickým růstem*. Délka kódu totiž logaritmicky narůstá s kódovaným číslem l_i. Počátek kódu je zobrazen ve druhém sloupci tab. 4.1.

4.3.2 Kombinace Huffmanova kódu a kódu s logaritmickým růstem

Po analýze dat byla pro dosažení nejlepších výsledků vybrána kombinace Huffmanova kódu a kódu s logaritmickým růstem. Použitý kód u něhož logaritmicky narůstá délka je zobrazen v tab. 4.1. Obrázek obr. 4.4 ukazuje způsob rozdělení množiny kódovaných čísel mezi oba způsoby kódování. Číslo K představuje nejvyšší číslo kódované Huffmanovým kódem.

Pro $K = 0$ jsou všechna čísla kódována kódem s logaritmickým růstem. Jako nejvýhodnější se pro tento případ jeví použití modifikace základního kódu podle druhého sloupce v tab. 4.1. Tato verze byla zvolena za základní variantu kódu s logaritmickým růstem.

[2]Symbol $O()$ představuje funkci pro výpočet složitosti a n je počet prvků.

[3]Unární kód si lze představit jako sérii stejných znaků zakončenou jiným znakem.

Číslo	Základní kód	1. modifikace	2. modifikace	3. modifikace
1	00	00	000	0000
2	01	01	001	0001
3	100	1000	010	0010
4	101	1001	011	0011
5	11000	1010	10000	0100
6	11001	1011	10001	0101
7	11010	110000	10010	0110
8	11011	110001	10011	0111
9	1110000	110010	10100	100000
10	1110001	110011	10101	100001
11	1110010	110100	10110	100010
12	1110011	110101	10111	100011
13	1110100	110110	1100000	100100
14	1110101	110111	1100001	100101
15	1110110	11100000	1100010	100110
16	1110111	11100001	1100011	100111
17	111100000	11100011	1100100	101000

Tabulka 4.1: Kód s logaritmickým růstem délky

Nyní uvažujme nejjednodušší variantu, ve které je délka binární části přímo rovná číslu n. Celý kód se skládá ze 2 částí. První část obsahuje unární kód a druhá část je binární. Unární část je tvořena posloupností znaků 1 zakončených nulou. Např. řetězec 1110 představuje zakódované číslo 3. Počet jedniček v unární části označme číslem n. Druhá binární část má délku, která je obecně funkcí čísla n. Po jejím načtení dostaneme číslo k. Hodnota přečteného symbolu je:

$$x = 1 + k \ + \sum_{i=0}^{n-1} 2^i. \tag{4.4}$$

Dále je potřeba objasnit jednotlivé modifikace kódu. Nechť je načteno n jedniček v $q-$té variantě našeho kódu. Pak délka binární části dosahuje $n + q$ bitů. Výsledná hodnota přečteného symbolu je dána rovnicí (4.5).

$$x = 1 + k \ + \sum_{i=0}^{n-1} 2^{i+q}. \tag{4.5}$$

Ukažme několik příkladů výpočtu zakódovaného čísla:
Mějme řetězec 1100010, o kterém víme, že je kódován ve druhé variantě. Unární část je 110. Potom $n = 2$; $n + q = 4$. Délka binární části je 4 bity. Binární část obsahuje číslo $k = 0010b = 2$. Celková hodnota zakódovaného symbolu je $x = 1 + k \ + \sum_{i=q}^{n-1} 2^{i+q} = 1 + 2 + \sum_{i=0}^{1} 2^{i+2} = 1 + 2 + 2^2 + 2^3 = 15$.

Mějme řetězec 0001, o kterém víme, že je kódován ve třetí variantě. Unární část je '0'. Potom $n = 0$; $n + q = 3$. Délka binární části je 3 bity. Binární část obsahuje číslo $k = 001b = 1$. Celková hodnota zakódovaného symbolu je $x = 1 + k \ + \sum_{i=q}^{n-1} 2^{i+q} = 1 + 1 + \sum_{i=0}^{-1} 2^{i+3} = 1 + 1 = 2$.

Mějme řetězec 100011, o kterém víme, že je kódován ve třetí variantě. Unární část je '10'. Potom $n = 1$; $n+q = 4$. Délka binární části je 4 bity. Binární část obsahuje číslo $k = 0011b = 3$. Celková hodnota zakódovaného symbolu je $x = 1 + k \ + \sum_{i=q}^{n-1} 2^{i+q} = 1 + 3 + \sum_{i=0}^{0} 2^{i+3} = 1 + 3 + 8 = 12$.

Jediný prohřešek proti tomuto pravidlu byl úmyslně udělán na prvním řádku v 0-té variantě kódu s exponenciálním růstem označené jako základní. Namísto kódových symbolů $0, 100, 101, 11000, ...$ byla použita posloupnost $00, 01, 100, 101, 11000, ...$ Důvodem vedoucím k této změně bylo zvýšení efektivity kódování.

Pro číslo $K \neq 0$ může být použita v závislosti na velikosti čísla K jedna z výše uvedených modifikací variant základního kódu. Podle experimentů se nejlépe se osvědčuje použít variantu č. $round(\log_2(K))$, která dává uspokojivé výsledky ve většině případů.

Pro volbu čísla K jsou v zásadě možné tři přístupy:

A. $K = 0$, tedy je použit pouze kód s logaritmickým růstem (jedná se o základní variantu).

B. K je nastaveno na pevnou hodnotu. Podle provedených experimentů se nejlépe osvědčuje $K = 32$ nebo $K = 64$. Pro technické výkresy je výhodnější větší hodnota K.

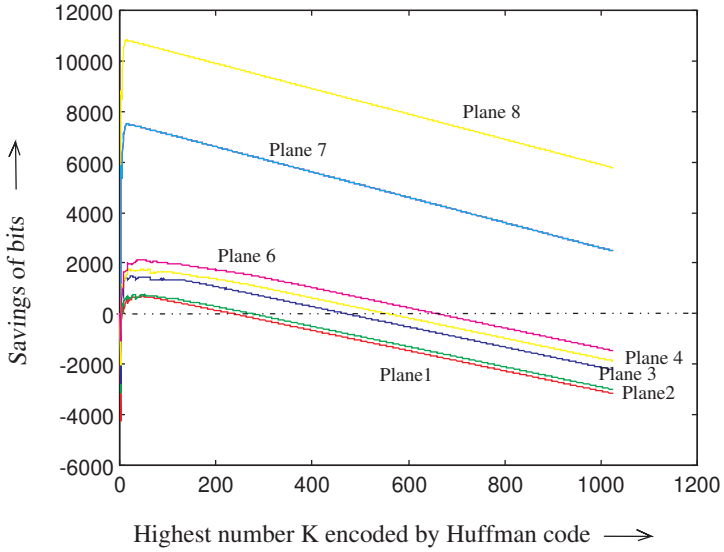
C. Číslo K je nastaveno optimálně podle komprimovaných dat. Tato operace je podrobně popsána v dalším textu.

Nalezení optimální hodnoty K

Proveďme další rozbor varianty C. Naším cílem je nalezení takového čísla K, pro které je výsledná délka kódovaných dat minimální.

Závislost mezi velikostí čísla K a délkou kódované zprávy je ukázána na obr. 4.5. V grafu je zobrazeno 8 závislostí, každá závislost se vztahuje k jedné bitové rovině vyjmuté z původního obrázku Lena s 256 úrovněmi šedi. Poznamenávám, že v grafu je vynesena úspora v bitech, kterou je možno určit podle vztahu: $Savings(K) = L_{message}(0) - L_{message}(K)$. Z hodnoty úspory $Savings$ je velmi dobře zřetelný vliv rozdělení intervalu na část kódovanou logaritmickým kódem a Huffmanovým kódem.

Proveďme nyní analýzu délky výsledné zprávy. Při bližším pohledu lze výslednou délku zprávy $L_{message}$ rozložit do tří složek. První složka L_{log} odpovídá součtu délek kódu všech položek s indexem větším nebo rovným K. Tyto položky



Obrázek 4.5: Závislost celkové délky zprávy na hodnotě čísla K

jsou kódovány kódem s logaritmickým růstem. Druhá složka $L_{Huffman}$ odpovídá součtu délek kódu všech položek s indexem menším než K. Poslední položka L_{tab} odráží velikost tabulky četností symbolů (v našem případě symboly představují délky kódových slov) nutné pro výpočet Huffmanova kódu.

$$L_{message} = L_{log} + L_{Huffman} + L_{tab} \tag{4.6}$$

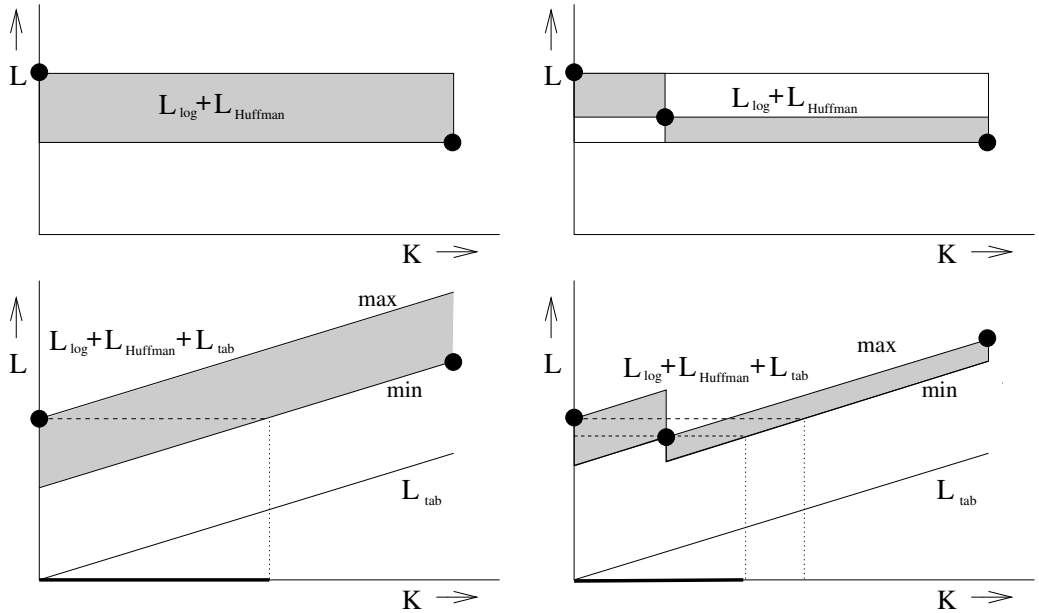
Délka L_{tab} je stále rostoucí funkcí v závislosti na počtu položek kódovaných Huffmanovým kódem. Číslo $(K + 1)$ udává počet položek v tabulce pro výpočet Huffmanova kódu. K položek je použito pro délky kódu jednotlivých vzdáleností l. Jedna položka je užita pro prefix, pomocí něhož jsou kódovány položky $l > K$. Délka L_{tab} je též závislá na velikosti Huffmanova kódového prostoru S (odpovídá nejdelšímu kódovému slovu). Proto je velikost jedné položky v tabulce v celých bitech dána vztahem $ceil(\log_2 S\,)$. Funkce $ceil()$, která je převzata z jazyka C, nalezne nejbližší vyšší celé číslo. 4 bity v rovnici (4.7) pro L_{tab} slouží k zaznamenání velikosti jedné položky v tabulce.

$$L_{tab} = 4 + (K + 1)\, ceil(\log_2 S\,) \qquad [bit]. \tag{4.7}$$

Délka Huffmanovy tabulky je potřebná pouze pro semiadaptivní kodér/dekodér. Samozřejmě při použití adaptivního Huffmanova kodéru/dekodéru poslední položka L_{tab} zmizí. Úspora místa při vyřazení poslední položky u adaptivního kodéru je z části ztracena v důsledku horší efektivity kódování prvních několika symbolů. Adaptivním kódováním by bylo možno o něco málo zvýšit výsledný kompresní poměr za cenu zvýšení složitosti a výpočetní náročnosti výsledného algoritmu. Plně adaptivní algoritmus by totiž musel počítat s poměrně

velkými indexy (1000 a více). Náročnost výpočtu totiž roste rychle s rostoucí velikostí abecedy. V případě, kdy je snaha ještě o něco zvýšit kompresní poměr, se jeví jako výhodná alternativa použití aritmetického kodéru. Uvedený návrh nechám prozatím otevřený a poukazuji na budoucí možné zvýšení kompresního poměru.

Námi použitý kód s logaritmickým růstem je prefixovým kódem. Pro určitou kombinaci pravděpodobností kódovaných symbolů by byl kódem optimálním. Pro obecnou kombinaci symbolů je optimálním kódem Huffmanův kód. Při zvyšujícím se K je stále více symbolů kódováno ve výsledném součtu o něco lépe. Funkce $L_{log}(K) + L_{Huffman}(K)$ je nerostoucí a má limitu danou entropií symbolů.



(a) 2 body známy (b) 3 body známy

Obrázek 4.6: Intervaly pro hledání minima funkce $L_{message}$.

Analýza funkce $L_{message}(K)$

Věnujme se nyní hledání extrému podrobněji. V předchozím textu bylo popsáno rozdělení výsledné zprávy na 3 složky a byly diskutovány jejich vlastnosti.

Bohužel je funkce $L_{log}(K) + L_{Huffman}(K)$ závislá na všech datech $l_1, \ldots, l_n$. Její funkční hodnotu lze získat novým výpočtem Huffmanova kódu. Výpočet funkční hodnoty v jednom bodě je velmi složitý, a proto je snaha neprovádět výpočet pro všechny hodnoty čísla K, které by přicházely v úvahu. Bylo by samozřejmě možno vypočítat délku zprávy ve všech bodech, a tímto způsobem nalézt optimální hodnotu K. Výhodné je, že K může být pouze celé číslo. Na obr. 4.5 je

úmyslně ukázána monotónní část pro velká K, ve které již vyhodnocovat délku zprávy očividně nemá smysl.

Podle předchozích znalostí lze snadno nalézt poměrně efektivní algoritmus pro určení čísla K, který již po několika málo výpočtech funkční hodnoty $L_{log}(K) + L_{Huffman}(K)$ je schopen se výrazně přiblížit k extrému. Algoritmus je založen na stanovení a postupném zužování oblasti, ve které se může extrém nalézat. Pro výpočet je použit interval možných hodnot $min(K) \leq L_{message}(K) \leq max(K)$. Každý výpočet funkční hodnoty $L_{message}(k)$ vede ke zůžení intervalu i pro větší množství okolních bodů. Zůžení intervalu se následně promítne do zmenšení oblasti, ve které by se mohl nacházet extrém $L_{message}$.

Bez výpočtu Huffmanova kódování lze poměrně snadno najít dva krajní body funkce $L_{message}(K)$ viz obr. 4.6(a). Oba krajní body jsou zobrazeny na obrázku černými tečkami. Již z těchto dvou bodů lze vyloučit velkou část intervalu možných K. Místa, kde může ležet extrém jsou v grafu označena plnou silnou čarou na ose x. Do grafu jsou vynesena minima a maxima funkce $L_{message}$, která byla zavedena v rovnici (4.6). U každého bodu K si můžeme být jisti, že délka kódu nebude větší než maximální. Pokud by pro nějaký bod K platilo, že jeho nejlepší zakódování bude horší než nejhorší zakódování odpovídající jiné hodnotě K, lze takový bod okamžitě vyloučit.

Délka zprávy pro další bod již musí být vyčíslena plným výpočtem. Po přidání dalšího bodu, viz obr. 4.6(b), dojde k dalšímu omezení oblasti, ve které je možno hledat lokální extrém. Pro velmi úzké vymezení oblasti s extrémem stačí výpočet hodnoty v několika málo bodech.

U funkce L_{tab} na obr. 4.6 byl zanedbán vliv velikosti Huffmanova kódového prostoru. Ke změně velikosti položky v tabulce bitových délek jednotlivých symbolů ze 3 bitů na 4 bity dojde při zvětšení Huffmanova kódového prostoru nad 255. S jistotou lze tvrdit, že více jak 255 různých nenulových položek vytvoří Huffmanův prostor větší než 255. Prostor větší než 65535 většinou nevzniká, a proto jej lze zanedbat. Znovu připomínám, že se nám jedná o odhad minima funkční hodnoty v nějakém bodě. Když tento odhad bude menší než vypočtená hodnota, pak je z hlediska použitého algoritmu vše v pořádku.

Podle provedených experimentů se funkce $L_{message}$ jeví poměrně hladká a má jeden výrazný mírně zvlněný extrém. Nacházejí se na ní pouze drobné perturbace způsobené zejména změnami exponentu např. z 2^3 na 2^4. V tomto případě je možno použít další heuristické informace [VŠ93] pro výrazné zrychlení nalezení extrému.

V implementovaném algoritmu je použita o něco jednodušší strategie. Je vyčíslována hodnota funkce $L_{message}$ pro $K \in \{4, 8, 16, 32, 64, 256, 512, 1024, \ ...\}$. Pokud dojde k výraznému nárůstu funkce $L_{message}$, vyčíslování bude zastaveno a za extrém bude vzato K, při němž byla určena minimální hodnota $L_{message}$. Tímto způsobem lze získat hodnotu K ležící velmi blízko extrému. Cena za nenalezení globálního extrému je v našem případě několik bitů a dosaženou hodnotu lze považovat za uspokojivou.

4.3.3 Zamítnutí kódování

Při kódování stále méně korelovaných dat dochází ke zhoršování činnosti prediktoru. Navrhovaný typ kódování dává nejlepší výsledky pro malé množství reziduí. Je samozřejmé, že se při postupném snižování kvality prediktoru dostaneme do bodu, kdy již sebelepší kódování nepomůže ke zvýšení kompresního poměru, viz též oddíl 3.1.2.

Je vhodné takovou situaci nějakým způsobem detekovat. V předchozím textu byl používán k ohodnocení úspěšnosti prediktoru počet reziduí. Z tohoto čísla nelze poznat kvalitu prediktoru, poněvadž nezávisí na velikosti rastru (přesněji počtu buněk). Pro účely hodnocení prediktoru bylo navrženo nové kritérium: *průměrná vzdálenost dvou reziduí* l_{avg}.

$$l_{avg} = \frac{\sum_{i=1}^{n} l_i}{n} = \frac{M\,N + 1}{n}. \tag{4.8}$$

Po aplikaci rovnice (4.3) lze l_{avg} vypočítat pouze z počtu reziduí n a rozměrů rastru. Hodnota l_{avg} se běžně pohybuje v řádu desítek až stovek. Pokud hodnota l_{avg} poklesne pod určitou hodnotu (podle experimentů je tato hodnota asi 2,4), nemá již smysl rezidua kódovat způsobem uvedeným v sekci 4.3 a je lépe data vůbec nekódovat. Pro nízké hodnoty $l_{avg} \in \{2; 2, 4\}$ by bylo možno navrhnout jiný způsob kódování, který by byl optimalizován právě pro tento případ. Tím by bylo možno o něco málo zvýšit kompresní poměr u špatně komprimovatelných dat. Zatím tato myšlenka nebyla realizována.

4.4 Prediktor s adaptivní velikostí sondy

Předchozí dva prediktory, které byly popsány v sekcích 3.6 a 4.2 mají pevně uskupené buňky prediktoru (3 v případě prediktoru *FH-2DOF* a 7 v případě *FH-3-DOF* + 1 aktuální buňka). Naším cílem je prozkoumat okolí predikované buňky za účelem zvětšení velikosti prediktoru. V oddílu 4.2.3 je rozebírána možnost přidávání dalších buněk v dalších dimenzích (Je potřeba mít na paměti omezení výběru buněk dané tzv. úhlem prediktoru.). Prediktor s větším množstvím buněk má větší schopnost se naučit obrazová data a produkuje tedy méně reziduí. Lze dokázat, že počet reziduí většího prediktoru bude v nejhorším případě stejný, při srovnání s počtem reziduí prediktoru jednoduššího. Podle našich předpokladů by se dalo očekávat, že se takový prediktor bude schopen naučit i nějaké jednodušší textury.

Prediktor s větší sondou může být zkonstruován podobným způsobem, jako v předchozích případech. Je dokonce možno vytvořit n-DOF prediktor bez jakýchkoliv omezujících požadavků na dimenzionalitu prostoru zahrnutého sondou. Při zpracování RGB dat může sonda zasahovat do sousedních barevných složek. Takový prediktor lze nazvat 4-DOF. Tři rozměry pocházejí z jednoho barevného kanálu a další rozměr prochází napříč barvami. Je žádoucí, aby větší sonda v sobě

obsahovala buňky sondy menší. Sondy jsou opět párovány tak, že se v rámci jednoho páru liší pouze hodnotou aktuální (odhadované) buňky.

Takto zkonstruovaný prediktor je v dalším textu označován *FH-Adapt*.

4.4.1 Omezení počtu buněk v sondě

Z praktického pohledu je počet buněk limitován velikostí kódované tabulky četností. Pro zaznamenání každého sloupce této tabulky je potřeba 1 bit. Počet sloupců tabulky je funkcí počtu buněk $columns = 2^k$ a roste v závislosti na tomto počtu exponenciálně. (Ve skutečnosti má sonda $k+1$ buňku. Buňka $k+1$ je však aktuální predikovaná, a proto není do seznamu buněk započítána.) Např. pro 20 buněk dosahuje velikost tabulky četností $2^{20} = 1$Mbit. Pokud by se jednalo pouze o dočasnou datovou strukturu, uloženou na chvíli v paměti, pak by tato velikost ještě nemusela být na závadu. Ale pokud má být tato informace uložena spolu s komprimovanými daty, pak dojde ke znehodnocení kompresního poměru.

Podařilo se nám navrhnout postup, jakým lze výše uvedený problém eliminovat. Vycházíme z předpokladu, že ne všechny buňky nesou stejné množství informace o hodnotě predikované buňky. Proto navrhujeme některé buňky *vyloučit* ze sondy na základě analýzy dat v obraze jak je popsáno v oddíle 4.4.2.

4.4.2 Modifikovaná analýza četností

I v případě adaptivní velikosti sondy je potřeba udělat v prvním kroku analýzu četností jednotlivých konfigurací sond. Tuto analýzu je potřeba provádět s maximální velikostí sondy. Výsledkem analýzy četností konfigurací sond je tabulka četností. Ta obsahuje 2^{k+1} položek. Číslo k popisuje velikost nosiče sondy (tj. počet buněk sondy). Vypočítaná tabulka je při větších velikostech sond obyčejně velmi velká z důvodu exponenciálního nárůstu počtu položek v závislosti na k. Naším úkolem je vybrat pouze i takových buněk tak, že $i < k$ a tyto buňky nesou nejvíce informace o predikované buňce. Tímto krokem bude provedena redukce tabulky četností na 2^i položek. Po zakódování je délka zakódované položky 1 bit a celá kódovaná tabulka má délku 2^i bitů.

> Důležité je zdůraznit, že krok výběru a vyřazování prvků sondy již nepotřebuje další průchod přes data obrazu. Veškerá potřebná informace je obsažena v tabulce četností.

Na jednotlivé buňky prediktoru lze nahlížet jako na příznaky běžně užívané v teorii rozpoznávání podle statistiky. Každý příznak přispívá nějakou hodnotou k odhadu aktuálního bodu sondy.

Po ukončení analýzy četností je známo, jak často se vyskytuje daná konfigurace sond v obraze. Z tabulky četností lze přesně určit, nakolik který příznak přispívá k odhadu aktuální buňky. Míra příspěvku je měřena v počtu reziduí,

která přibudou po vyjmutí daného příznaku. Dále byl vyvinut algoritmus, který umožňuje vyloučit kterýkoliv příznak z tabulky četností a eliminovat jeho vliv. Velikost tabulky četností se poté sníží na 1/2 předchozí hodnoty.

Dosud bylo pohlíženo na tabulku četností jako na 2D strukturu obsahující řádky a sloupce. Vhodná reprezentace tabulky četností umožní použití velmi efektivních algoritmů určených pro práci s ní, které jsou založeny na bitových operacích. Protože je paměť počítačů lineární a z důvodů potřeby efektivní práce s touto strukturou, jsou dvourozměrná data z tabulky ukládána do lineárního pole. Celá tabulka četností je umístěna do vektoru $\vec{q}$ s 2^{k+1} prvky[4]. Do prvních 2^k prvků vektoru $\vec{q} = (q_0, \ldots, q_{2^k})$ je uložen první řádek tabulky četností a těsně za ním řádek druhý. Připomeňme si, že aktuální buňka má vždy nejvyšší index. Této notaci odpovídají následující rovnice.

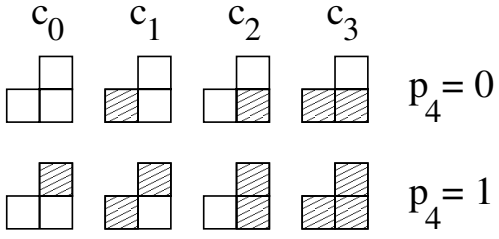
Nechť e_{all} je počet výskytů všech méně četných konfigurací sond (v rámci každého sloupce tabulky četností), který vlastně představuje počet reziduí. Nechť b je index prvního prvku z druhé poloviny tabulky četností $b = 2^k$ a c se rovná 2^ω, kde ω představuje je pořadové číslo vyřazovaného (testovaného) příznaku.

$$e_{all} = \sum_{i=0}^{b-1} \min(q_i, q_{i+b}). \tag{4.9}$$

Počet reziduí lze určit z rovnice (4.9). *Estimativnost* je v našem případě definována jako číslo, které udává, o kolik se zvětší počet reziduí (chyb prediktoru) v případě vyloučení příznaku číslo c. Definujme *estimativnost* e_c příznaku s přiřazeným číslem c pomocí následující rovnice:

$$e_c = \frac{1}{2} \sum_{i=0}^{b-1} \min\left(q_i + q_{i\oplus c}, q_{i\oplus b} + q_{i\oplus c\oplus b}\right) - e_{all}. \tag{4.10}$$

Operátor $\oplus$ v kontextu celé práce znamená logickou operaci nonekvivalence XOR počítanou po bitech v binárním tvaru čísel. Operátor + v rovnici (4.9) vykonává stejnou činnost, jaká by nastala při aplikaci operátoru $\oplus$.



Obrázek 4.7: Vyjmutý příznak č. 3.

V rovnici (4.10) musí být před sumací uveden člen $\frac{1}{2}$, protože estimativnost příznaku c je ve skutečnosti počítána dvakrát. Poznamenejme, že hodnota $e_c \geq 0$.

[4] k je velikost nosiče sondy.

Vysvětleme vyřazování jednotlivých příznaků na příkladu. Ze sondy podle obr. 3.4 chceme vyřadit příznak p_3 (nachází se v horním levém rohu sondy). Z původní konfigurace sond obr. 3.5 vznikne nová konfigurace sond. Ta je zobrazena na obr. 4.7. Dále je potřeba vytvořit novou tabulku četností jednotlivých příznaků. Pro její vytvoření je vhodné uvědomit si fakt, že jedno políčko nové tabulky vzniklo sloučením dvou políček tabulky předchozí. Políčka sloučená dohromady se liší hodnotou vyjímaného příznaku. Hodnota nové položky je proto rovna součtu hodnot obou položek z předchozí tabulky, které jsou v ní implicitně obsaženy.

Ukázková tabulka četností tab. 3.3 na straně 31 bude v případě vyloučení příznaku p_3 redukována na tab. 4.2. Po vyřazení příznaku jsou všechny zbylé příznaky přečíslovány. Pro rekonstrukci v dekodéru musí být zapamatována čísla všech vyřazených příznaků, která vlastně definují nový tvar sondy.

	c_0	c_1	c_2	c_3
$p_4 = 0$	1	1	1	4
$p_4 = 1$	3	3	2	1

Tabulka 4.2: Tabulka četností s vyřazeným příznakem p_3.

Algoritmus vyřazování příznaků může pracovat též iterativně. To znamená, že po vyřazení jednoho příznaku může být vyloučen příznak druhý atd. Na pořadí vylučování příznaků nezáleží. Pokud si vybereme i příznaků tak, že $i \in k$, pak je prediktor s i vybranými příznaky vždy stejný.

Pro vyloučení dalších příznaků není potřeba provádět změny ve vyřazovacím algoritmu. Pouze je potřeba si pamatovat pro pozdější rekonstrukci čísla vyřazovaných příznaků. Protože nezáleží na pořadí, v jakém jsou příznaky vyřazovány, postačí pro uchování informace o vyřazených příznacích bitové pole s velikostí (každý příznak má přiřazen vlastní bit 0-nevyřazen, 1-vyřazen) rovnou počtu příznaků.

Nejvýhodnější strategie spočívá ve vyřazování příznaku s nejnižší hodnotou estimativnosti (hladový algoritmus). Je potřeba stanovit okamžik, kdy dojde k zastavení iterací. Jinak by mohlo dojít k degradaci prediktoru na prediktor jednobitový, který je velmi špatný. Je potřeba uvážit, jaká data mají vliv na délku výstupního souboru. Jedná se o kódovaná rezidua a o popis stavu prediktoru. Výsledná funkce délky dat je tvořena součtem dvou dílčích funkcí $L_{full} = \hat{L}_{message}(e_{all}) + 2^k$. Funkce $\hat{L}_{message}(e_{all})$ představuje odhad $L_{message}$ pouze na základě počtu reziduí bez znalosti jejich rozložení. Odhad roste monotónně v závislosti počtu reziduí a tedy i na zmenšení velikosti sondy prediktoru odpovídající číslu iterace. Délka kódovaného stavu prediktoru exponenciálně klesá v závislosti na klesající hodnotě k. Funkce $L_{message}$ je závislá na datech a v naší implementaci je proveden

pouze její odhad na základě heuristiky. Při lepším odhadu (nebo dokonce přesném určení, které by vyžadovalo další průchod daty) by mohlo dojít k dalšímu zvýšení kompresního poměru. Podle předpokladů a provedených experimentů by však zvýšení nebylo nijak výrazné.

Je hledán extrém délky $length$. Za extrém je považován takový bod, ve kterém je zvětšení délky kódovaných reziduí větší (pro příznak s nejmenší estimativností e) než pokles délky tabulky. V tomto bodě dojde k zastavení algoritmu. Druhou možnost zastavení algoritmu představuje vyřazení všech příznaků ze sondy (tabulka četností má v takovém případě pouze dvě položky). Vyřazení všech příznaků nastane při dvou typech obrazových dat: Buď se jedná o nekomprimovatelný bílý šum a nebo o prázdnou bitovou rovinu.

Vzhledem k limitovanému počtu příznaků je zaručeno zastavení algoritmu za krátkou dobu.

4.5 Snížení paměťové náročnosti analýzy četností

Všechny zde navrhované prediktory počínaje základní variantou popisovanou v sekci 3.6 až po n dimenzionální prediktor s proměnnou velikostí sondy vyžadují provedení analýzy četností jednotlivých konfigurací sondy. Výsledek analýzy četností je ukládán do tabulky četností. Její velikost je dána vztahem (4.11) a roste s exponentem počtu buněk v sondě. Pro větší počet buněk se může tato velikost stát neúnosnou, a proto je vhodné analyzovat, zda jsou všechna data obsažená ve frekvenční tabulce bezpodmínečně nutná pro prováděné výpočty. Připomínám, že dekodér vyžaduje mnohem méně paměti, než kodér.

Mějme obraz s velikostí rastru $M \times N$ a sondu s k buňkami (aktuální buňka opět není do počtu buněk sondy k započtena). Pro tabulku četností je potřeba vyhradit $TabSize$ bitů v paměti počítače.

$$TabSize = (log_2(MN))\, 2^{k+1} \quad [bit]. \tag{4.11}$$

První člen rovnice (4.11) udává minimální velikost jedné položky v tabulce četností. Většinou není vhodné přepisovat aritmetiku pro manipulaci s n bitovými čísly a je lepší použít standardní 32 nebo 64 bitová čísla. Ve druhém členu je zahrnut počet buněk tabulky četností. Příklad takto utvořené tabulky může být nalezen v tab. 3.3.

Připomeňme si, že v každém sloupci tabulky jsou uloženy četnosti dvou konfigurací sond lišící se pouze hodnotou aktuální buňky. Analyzujme, na co všechno jsou tyto dvě hodnoty potřebné. Pro popis prediktoru postačí znát, které číslo z dvojice je větší. Prediktor s adaptivní velikostí sondy využívá tabulku četností pro výpočet estimativnosti jednotlivých příznaků.

	c_0	c_1	c_2	c_3	c_4	c_5	c_6	c_7
$p_4 = 0$	-1	-2	1	1	-1	-1	-2	2

Tabulka 4.3: Redukovaná tabulka četností pro hypotetický obrázek.

Ani v jednom případě nejsou obě hodnoty ve sloupci tabulky četností potřebné a pro všechny dříve popisované výpočty postačí jejich rozdíl, jak je ukázáno na následujících řádcích. Vytvořme si novou tabulku četností tab. 4.3, do které jsou ukládány pouze rozdíly obou řádek a jejíž velikost je téměř poloviční v porovnání s předchozím případem viz následující rovnice:

$$TabSize_{new} = (\log_2(MN) + 1)\, 2^k \quad [bit]. \tag{4.12}$$

Minimální velikost položky nové tabulky je o jeden znaménkový bit větší, než u předchozího případu (proto je v rovnici (4.12) uveden člen +1). Zvětšení o jeden bit je zanedbatelné vzhledem k rezervě dané použitím nejbližší vyšší bitové délky čísel, která je podporována danou architekturou počítače. Na první pohled se zdá, že navrhovanou transformací dojde ke ztrátě některých dat. Ale není tomu tak. V následujícím textu je dokázáno, že všechny operace včetně výpočtu estimativnosti a vyřazování příznaků ze sondy lze udělat podle redukované tabulky četností. Dokonce může dojít ke zrychlení algoritmů vzhledem ke snížení množství zpracovávaných dat.

Následující příklad ukazuje na způsob, jakým lze určit počet reziduí z redukované tabulky četností. Nechť $c_{(i,j)}$ jsou hodnoty z původní tabulky četností. Pak symbol $Major$ podle rovnice (4.13) představuje součet všech správných predikcí pro daná data. Symbol $Minor$ z rovnice (4.14) je součet všech chybných predikcí, který je roven počtu reziduí.

$$Major = \sum_{i=0}^{2^k-1} max(c_{(i,0)}, c_{(i,1)}). \tag{4.13}$$

$$Minor = \sum_{i=0}^{2^k-1} min(c_{(i,0)}, c_{(i,1)}). \tag{4.14}$$

Je jasné, že tyto dvě hodnoty nemohou být určeny přímo z redukované tabulky četností. Ale protože existuje mezi oběma hodnotami $Major$ a $Minor$ určitá závislost, lze ji výhodně využít:

$$\begin{aligned} Minor + Major &= M * N, \\ Major - Minor &= \sum_{i=0}^{2^k-1} |cnew_{(i)}| \,. \end{aligned} \tag{4.15}$$

Symbol *cnew* označuje redukovoanou (zmenšenou) tabulku četností. Ta již ztratila druhý rozměr a stala se vlastně vektorem. Ze dvou rovnic (4.15) je určení počtu reziduí $Minor$ již velmi jednoduché. Symbol $Minor$ označuje v podstatě totéž, co znamená e_{all} v oddílu 4.4.2.

$$Minor = \frac{(MN) - \sum_{i=0}^{2^k-1} |cnew_{(i)}|}{2}. \tag{4.16}$$

4.5.1 Výpočet estimativnosti z redukované tabulky četností

I z redukované tabulky četností lze vypočítat estimativnosti jednotlivých příznaků. Nechť číslo k představuje opět počet příznaků sondy a c se rovná 2 umocněno na pořadové číslo vyřazovaného (testovaného) příznaku. Rovnici (4.10) bude potřeba upravit speciálně pro výpočet estimativnosti z redukované tabulky četností. Je potřeba zavést symboly $Minor_c$ a $Major_c$, které se týkají sondy s vyloučeným příznakem a mají téměř shodný význam jako $Minor$ a $Major$.

$$\begin{aligned} Minor_c + Major_c &= M * N, \\ Major_c - Minor_c &= \frac{1}{2} \sum_{i=0}^{2^k-1} |cnew_{(i)} + cnew_{(i \oplus c)}|. \end{aligned} \tag{4.17}$$

Hledáme hodnotu estimativnosti $e_c = Minor_c - Minor$. Tu lze získat po vyřešení dvou soustav dvou jednoduchých rovnic (4.17) a (4.15). Je vhodné poznamenat, že první soustavu rovnic postačí vyřešit pouze jednou pro všechny příznaky.

$$Minor_c = \frac{(MN) - \frac{1}{2}\sum_{i=0}^{2^k-1} |cnew_{(i)} + cnew_{(i \oplus c)}|}{2}. \tag{4.18}$$

4.5.2 Vyřazování příznaků z redukované tabulky četností

V textu pojednávajícím o redukci tabulky četností bylo řečeno, že nové políčko redukované tabulky četností vzniká sloučením dvou políček lišících se hodnotou vyřazované buňky. Má-li tabulka četností dva řádky, pak pro druhý řádek jsou provedeny obdobné operace jako pro řádek první. Hodnoty v rámci jednoho sloupce budou sloučeny s hodnotami ze sloupce jiného, a tím vytvoří sloupec v tabulce s vyřazeným indexem.

A jak se situace změní pro případ redukované tabulky četností? Demonstrujme celou situaci na příkladu. Mějme za úkol sloučit dva sloupce A a B do sloupce C. Potom bude platit $c_1 = a_1 + b_1$; $c_2 = a_2 + b_2$. Podobná situace nastává i v případě redukované tabulky četností tj. $a = a_1 - a_2$; $b = b_1 - b_2$ a $c = c_1 - c_2$. Pak ze zákonů platných pro operátory "+" a "-" platí:

$$c = c_1 - c_2 = (a_1 + b_1) - (a_2 + b_2) = (a_1 - a_2) + (b_1 - b_2) = a + b. \qquad (4.19)$$

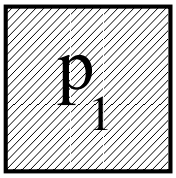
4.6 Dodatečná analýza reziduální bitové mapy

Za reziduální bitovou mapu je považována bitová mapa do níž jsou uložena rezidua z původní bitové mapy. Pro takto vzniklou bitovou mapu platí určitá omezení, která lze s výhodou využít. Právě tato omezení je potřeba blíže analyzovat.

Počet reziduí je omezen. Připomeňme si, že binární rastr lze popsat zobrazením $f(x, y) \rightarrow \{0, 1\}$, která je definována na nosiči T=$\{(x, y)$: $0 \leq x < M$; $0 \leq y < N\}$. Číslo M představuje šířku rastru a N jeho výšku.

V předchozím textu byla definována průměrná vzdálenost mezi dvěma rezidui, viz rovnice (4.8). Je zřejmé, že průměrná vzdálenost mezi dvěma rezidui závisí na počtu reziduí a tedy i na pravděpodobnosti rezidua. Věnujme se pravděpodobnosti rezidua v rastru.

Nechť proměnná B' (Černá-Black) nabývá hodnoty dané celkovým počtem černých buněk v binárním rastru a proměnná W' (Bílá-White) obsahuje počet bílých buněk rastru. Na prediktor lze nahlížet jako na funkci, která transformuje jednu binární bitovou mapu na druhou. Výstupní bitová mapa obsahuje rezidua a prázdné buňky. Označme si stejným způsobem počty položek v reziduálním rastru. Nechť počet reziduí je označen symbolem B a počet prázdných položek jako W.



Obrázek 4.8: Nejjednodušší sonda prediktoru.

Kvalitu prediktoru je potřeba nějakým způsobem hodnotit. V předchozím textu bylo použito pro hodnocení kvality prediktoru celkového počtu reziduí a průměrné vzdálenosti mezi dvěma rezidui. Pro účely analýzy zaveďme ještě kritérium pravděpodobnosti rezidua:

$$p(B) = \frac{B}{B + W}\,. \qquad (4.20)$$

Uvedené kritérium je téměř shodné s převrácenou hodnotou $1/e_{avg}$ až na jedničku, kterou lze zanedbat. V extrémním případě může mít obrazová funkce všechny buňky bílé nebo černé. Hodnota pravděpodobnosti se může pohybovat

v intervalu $p(B) \in \langle 0;1\rangle \rightarrow l_{avg} \in \langle 1; M\,N\rangle$. Po transformaci obrazu na rezidua pomocí prediktoru se situace o něco změní.

Pro náš příklad použijme nejjednodušší možný binární prediktor, který je zobrazen na obr. 4.8. Zde je implicitně předpokládáno optimální nastavení prediktoru podle analýzy četností. Tento prediktor obsahuje pouze aktuální buňku a tabulka četností má pouze jediný sloupec (takovou tabulku lze spíše nazvat histogramem). Pro zakódování popisu prediktoru je třeba 1 bitu.

Z definice prediktoru lze postulovat následující tvrzení: Pravděpodobnost $p(B)$ musí ležet v intervalu $p(B) \in \langle 0;0,5\rangle$.

Zdůvodnění:
Před aplikací prediktoru jsou v zásadě možné dvě situace:

1. $p(B') < p(W') \rightarrow$ Prediktor bude predikovat bílé buňky. $W' = W$ a všechny černé buňky se stanou rezidui: $B' = B \rightarrow p(B) \leq p(W)$.

2. $p(B') \geq p(W') \rightarrow$ Prediktor bude predikovat černé buňky. $B' = W$ a všechny bílé buňky se stanou rezidui: $W' = B \rightarrow p(B) \leq p(W)$.

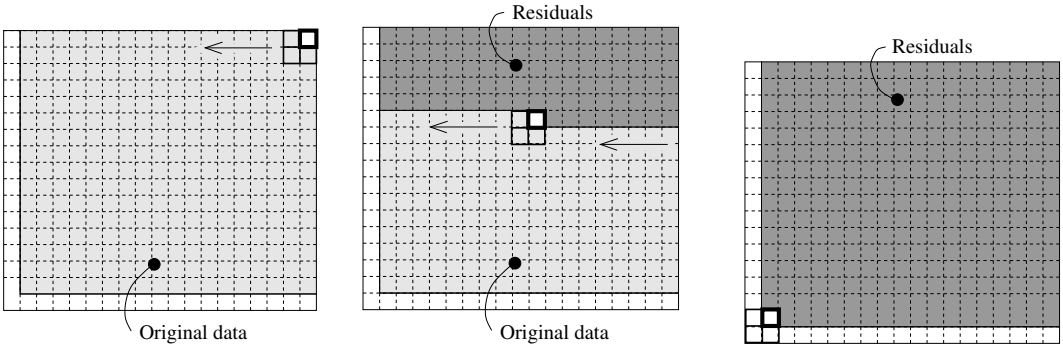
B' je počet bílých buněk v původním rastru a W' je počet černých buněk. Celé zdůvodnění předchozích tvrzení je též obsaženo v následujících vztazích:

$$\begin{aligned} p(B) &\leq p(W) \qquad p(B) + p(W) = 1, \\ p(B) &\leq 1 - p(B), \\ 2p(B) &\leq 1, \\ p(B) &\leq 0.5\,. \end{aligned} \tag{4.21}$$

U jakéhokoliv prediktoru navrhovaného typu s větší sondou musí být pravděpodobnost reziduí menší nebo rovna pravděpodobnosti reziduí nejjednoduššího prediktoru. Tvrzení může být též dokázáno s využitím rovnice (4.10). Počet reziduí se zvyšuje v závislosti na zmenšování počtu buněk sondy. Největší možné zmenšení jakékoliv sondy je právě na sondu s jedinou aktuální buňkou, a proto je úvaha o četnostech reziduí v předchozím textu platná pro libovolný typ sondy.

4.7 Úspora místa při ukládání reziduí

Již prof. Schlesinger ve své práci [Sch89] ukázal, že je možno ukládat rezidua do stejné datové struktury jako původní data. Postup spočívá v postupné přeměně původní matice na matici reziduí. Tuto myšlenku je dokonce možno využít při důkazu jednoznačnosti transformace rezidua $\leftrightarrow$ původní data. Nyní se věnujme uvedenému jevu podrobněji.



(a) počátek redukce (b) průběh (c) rezidua vytvořena

Obrázek 4.9: Jednotlivé fáze tvorby reziduí z původní bitové matice.

Celá situace je zachycena na obr. 4.9. Na počátku máme originální binární rastr společně s popisem prediktoru obr. 4.9(a). Sonda je umístěna do pravého horního rohu. Podle tabulky četností zjistíme typ konfigurace. V případě více četné konfigurace přepíšeme hodnotu aktuální buňky 0 a v případě méně četné konfigurace přepíšeme hodnotu původní buňky 1. Poté lze posunout sondu o jednu pozici doleva a celý postup opakovat až do konce řádku. Po dosažení konce řádku lze přesunout sondu na nižší řádek a zpracovat jej stejným způsobem. Hodnoty přepsané při popisovaném systematickém postupu již nikdy nebudou potřebné pro prediktor k predikci nových reziduí. Situace, kdy se sonda nachází přibližně v polovině dat je zobrazena na obr. 4.9(b).

Po neustálém opakování algoritmu bude nakonec dosaženo posledního bodu, který se nachází v levém dolním rohu obr. 4.9(c). Po jeho transformaci na reziduum je požadovaného cíle dosaženo $\mapsto$ celá matice obsahuje rezidua (samozřejmě s výjimkou prvního řádku a prvního sloupce, které jsou na obr. 4.9 zobrazeny bíle).

K postupu redukce binárního rastru na rezidua existuje inverzní operace. Po každém kroku lze jednoznačně udělat též zpětný krok v opačném směru, který ruší všechny změny v rámci posledního kroku.

Důkaz jednoznačnosti transformace bitové mapy na rezidua:

Důkaz se týká prediktoru popsaného v sekci 4.2. Použijme techniku matematické indukce. Navraťme se do konfigurace, kdy máme transformovánu pouze jedinou buňku na reziduum obr. 4.9(a). Známe všechny buňky sondy s výjimkou aktuální buňky. Tím je jednoznačně dán sloupec v tabulce četností. Existují pouze dvě možnosti:

Aktuální buňka obsahuje reziduum - v tomto případě se podíváme do aktuálního sloupce v tabulce četností na konfiguraci, která byla označena jako méně četná. Podle této konfigurace nastavíme aktuální buňku na původní hodnotu.

Aktuální buňka je prázdná - aktuální sloupec v tabulce četností má konfiguraci, která byla označena jako četnější. Podle četnější konfigurace nastavíme aktuální buňku na původní hodnotu.

Každý sloupec může obsahovat pouze jednu dvojici sond, z nichž jedna musí být označena jako méně četná a druhá jako více četná. Žádná další možnost již v dané konfiguraci sond nastat nemůže. Podle nalezené konfigurace sondy lze obnovit původní hodnoty v aktuálním bodě sondy.

Nyní předpokládejme, že se nacházíme se sondou na libovolné pozici v binární matici. V této pozici lze udělat naprosto totožnou operaci s výše popsanou operací. Po jejím provedení jsme jednoznačně transformovali reziduum v aktuální pozici na původní hodnotu buňky ležící na daném místě. Vzhledem k jednoznačnosti popsaného kroku lze vždy při n reziduích alespoň jedno ubrat. Z toho vyplývá, že lze posunout sondu doprava a celý postup opakovat až do dosažení posledního rezidua v matici.

Vzhledem k existenci a jednoznačnosti transformace reziduí na původní data lze tvrdit, že ke každé matici reziduí lze jednoznačně přiřadit jedinou jí příslušející binární matici (pro jednu konfiguraci prediktoru). Uvedené tvrzení platí v plném rozsahu i opačně.

Ukládání reziduí na jiná místa

Podotýkám, že nikde není striktně požadováno, aby byla rezidua ukládána do stejné datové struktury jako původní data. Při paralelním zpracování obrazových dat více procesory se jeví naopak výhodnější rozdělit celou plochu na dlaždice a každému procesoru přidělit jednu dlaždici. Procesor dlaždici zpracuje a výsledná data může odeslat do jiné datové struktury.

Další možností, u které uvedená konfigurace vlastně není vůbec potřeba, je plně adaptivní komprese vzdáleností jednotlivých reziduí. Pokud sonda generuje na pozici x, y reziduum, plně postačí si tuto pozici dočasně zapamatovat. Ze dvou po sobě následujících pozic již lze vypočítat vzdálenost l_i dvou reziduí a tuto vzdálenost předat adaptivnímu kodéru. Souřadnici předchozího rezidua je v tomto bodě možno již zapomenout a pokračovat sondou na další pozici tak dlouho dokud nebude detekována minoritní konfigurace vyžadující reziduum.

Poslední dva náměty bohužel nebyly otestovány, ale není důvod se obávat jejich nefunkčnosti. Podle mého názoru sice nedojde ke zlepšení kompresního poměru, ale některé specifické aplikace mohou vyžadovat např. úsporu paměti při tvorbě reziduí.

4.8 Výsledky experimentů

Většina navrhovaných řešení byla experimentálně ověřena. Pro testování byly použity standardní testovací obrázky, které jsou používány i v jiných článcích.

Nejdůležitější testovací obrázky jsem nalezl na internetu a umístil na ně odkaz ze své domovské www stránky: *http://cmp.felk.cvut.cz/~fojtik/*.

Hlavní cíle experimentů byly:

Vyzkoušení funkčnosti navrhovaných technik. Aby bylo možno porovnat výsledky navrhovaných algoritmů s algoritmy jiných autorů bylo potřeba uvést do provozu funkční implementaci navrhované metody.

Porovnání dosažených výsledků s ostatními algoritmy. Hlavním účelem tohoto bodu je vedle sebe postavit jednotlivé kompresní algoritmy a do tabulky vynést jejich kompresní poměry. Tím je čtenáři dána možnost přehledně porovnávat kompresní poměry dosahované porovnávanými algoritmy na jednotlivých typech obrázků.

Zobrazení zajímavých hodnot z činnosti algoritmu. Jako výsledek experimentu je někdy vhodné zobrazit hodnoty, s nimiž se sice interně počítá, ale které nejsou přímo viditelné na výstupu. V případě metody FH-Adapt se jedná např. o estimativnost jednotlivých buněk sondy. Demonstrace takových hodnot umožní lepší pochopení teoretické části.

Nalezení problémů při implementaci. Při realizaci funkčního algoritmu většinou vyvstane několik problémů, které v teoretickém návrhu nebyly uvažovány. Právě k jejich zdárnému vyřešení a popisu slouží experimentální implementace.

Zjištění možných zlepšení v účinnosti komprese a v efektivitě algoritmu. Nejčastěji vznikají při realizaci a experimentech nápady směřující k budoucímu vylepšení vyvíjené techniky. Zjištěné návrhy jsem umístil spíše do teoretických částí této práce.

Binární prediktor pro šedotónové obrázky

Demonstrujme chování navrhovaného *FH-3-DOF* kompresního algoritmu na standardním šedotónovém testovacím obrázku Leny. Pro testovací účely byla použita varianta 256×256 s 8 bity na jeden pixel tzn. 256 úrovněmi šedi viz obr. 4.10.

Na obr. 4.11 jsou ukázány jednotlivé bitové roviny původního obrázku Leny obr. 4.10. Dále se zde nachází pro každou bitovou rovinu jí odpovídající reziduální bitová rovina. Podle množství reziduí (v našem případě černých bodů) lze usuzovat na kvalitu komprese pro jednotlivé bitové roviny. Povšimněte si, nakolik rezidua označují důležitá místa v obrázku např. tvar klobouku.

Data z experimentu jsou zapsána též v tab. 4.4. V prvních dvou řádcích tabulky, které jsou označeny návěštími *FH-3-DOF*, *FH-2-DOF* je zapsán celkový

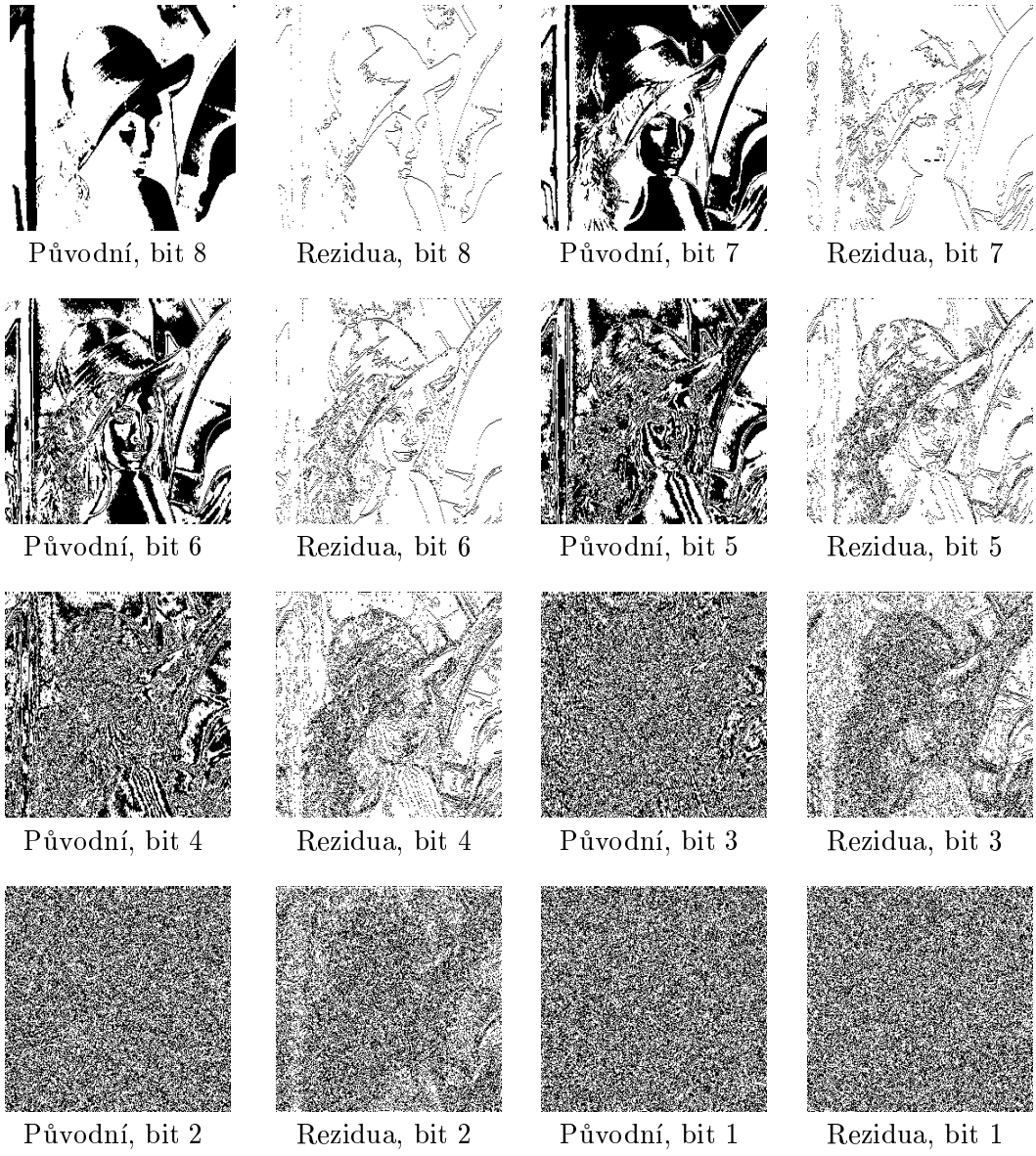
Obrázek 4.10: Lena

počet reziduí pro danou bitovou rovinu. Za pozornost stojí se podívat na to, jak dochází ke zvyšování počtu reziduí směrem od nejvyšší bitové roviny k nejnižší bitové rovině. Výsledky pozorování jsou zřejmé. V nižší bitové rovině je soustředěno mnohem více šumu než ve vyšších bitových rovinách, viz též obrázky jednotlivých bitových rovin na obr. 4.11. Náhodná data je mnohem těžší predikovat, což se odráží na počtu reziduí.

Na posledních sedmi řádcích tab. 4.4 je ukázána estimativnost jednotlivých buněk sondy *FH-3-DOF* prediktoru. Konfiguraci buněk *FH-3-DOF* prediktoru lze nalézt v obr. 4.1 na stránce 36. Některé položky v tabulce jsou označeny jako prázdné. Slovo "prázdný" bylo použito pro zvýraznění skutečnosti, že takto označené příznaky neleží v datech obrazu. Do uvedených políček tabulky by bylo možno beze ztráty obecnosti napsat také hodnotu 0. Hodnoty estimativnosti v rámci jedné bitové roviny ukazují, že ne všechny okolní buňky jsou pro odhadování aktuální buňky stejně důležité. Příznak č. 6 (ležící pod[5] aktuální buňkou) a příznak č. 4 (ležící nad aktuální buňkou ve vyšší bitové rovině) nesou největší množství informace pro prediktor. Poznamenejme, že estimativnost příznaku č. 6 je v tabulce označena symbolem e_6 a uvedené číslo je bezrozměrné, viz oddíl 4.4.2. Zde presentovaná data ospravedlňují adaptivní prediktor (viz sekce 4.4), který se snaží špatné (nevýznamné) příznaky ze sondy jednoduše odstranit.

Vliv jednotlivých modifikací kódování a jednotlivých modifikací prediktoru je prezentován v tab. 4.5. Pro vytvoření tabulky byl zakódován obrázek Leny 512 × 512 s 256 úrovněmi šedi a obrázek Garfield se šestnácti úrovněmi šedi devíti různými způsoby. Způsoby odpovídají modifikacím předkládané techniky. Jednotlivé sloupce popisují typ kódování a jednotlivé řádky typ prediktoru. Na průsečíku sloupce a řádky je zapsána délka výsledných dat po komprimaci hledanou kombinací prediktor/kodér v bajtech. Tabulka umožňuje sledovat vliv jednotlivých vylepšení navrhovaného algoritmu na délku komprimovaných dat. Velikost souboru se podle očekávání snižuje jak ve vodorovném tak i ve svislém směru. Levé

[5]Termín 'pod' je použit pro nižší bitovou rovinu a 'nad' pro vyšší bitovou rovinu. Prostorové relace vycházejí z obr. 4.2.



Obrázek 4.11: Grafické zobrazení původních bitových rovin a bitových rovin s uloženými rezidui po predikci algoritmem *FH-3-DOF*.

Bitová rovina	rov. 7	rov. 6	rov. 5	rov. 4	rov. 3	rov. 2	rov. 1	rov. 0
2-DOF rezidua	2010	5477	10177	15067	21865	28595	31771	32214
3-DOF rezidua	prázdný	3841	5935	9421	15045	21690	28622	31272
estimativnost e_1	prázdný	25	160	218	233	91	125	320
estimativnost e_2	prázdný	616	2165	3108	4930	4919	2521	457
estimativnost e_3	prázdný	52	180	219	233	91	128	352
estimativnost e_4	prázdný	1892	4101	5543	6611	6600	2813	447
estimativnost e_5	0	14	123	143	92	26	112	361
estimativnost e_6	1457	2097	3477	4169	5807	5031	2312	467
estimativnost e_7	0	23	118	136	170	85	71	220

Tabulka 4.4: Kvantitativní porovnání algoritmů *FH-2-DOF* a *FH-3-DOF* podle počtu reziduí.

horní políčko tabulky je vyhrazeno nejjednodušší metodě predikce *FH-2* v kombinaci s nejjednodušším kodérem *Log. kód* (kód s logaritmickým růstem) a podle očekávání je délka zakódovaného obrázku v obou případech největší. Opačná situace nastane u kombinace prediktoru *FH-Adapt* v kombinaci s kodérem nazvaným *Plovoucí Huffman*, viz pravé dolní políčko tab. 4.5.

Obrázek Lena:
512 × 512 × 8

Metoda/Kódování	Log. kód	Pevný Huffman	Plovoucí Huffman
FH-2	203000	200758	200599
FH-3	178655	176021	175922
FH-Adapt	178130	175541	175432

Obrázek Garfield:
640 × 480 × 4

Metoda/Kódování	Log. kód	Pevný Huffman	Plovoucí Huffman
FH-2	5636	4687	4496
FH-3	4759	3971	3819
FH-Adapt	4171	3490	3328

Tabulka 4.5: Porovnání účinnosti komprese 3 navrhovaných typů prediktorů a 3 typů kódovacích technik. Hodnoty jsou udávány v bajtech.

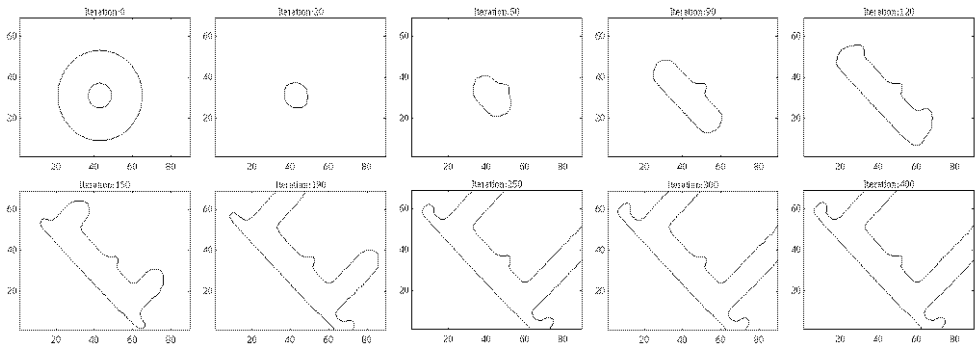
4.8.1 Srovnání s ostatními komprimačními algoritmy

Pro porovnání účinnosti komprese byly vybrány obrázky ze standardní testovací sady, která byla navržena komisí JPEG [AT94]. Jedná se zejména o obrázky

Lena obr. 4.10, Baboon (hlava opice), Boats (dvě lodě u pobřeží), viz obr. 4.12. Dále bylo zvoleno několik dalších vlastních obrázků. Na nich jsou obsaženy určité rysy, které jsem ve standardní sadě nenašel. Hlavně se jedná o obrázek Drawing (technický výkres).

Opice Baboon Dvě lodě Boats Tapeta Tartan

Obrázek 4.12: Ukázka tří použitých testovacích obrázků.



Obrázek 4.13: *Drawing* obrázek obsahující technickou kresbu.

Informace o účinnosti komprimace jednotlivých metod byly převzaty z následujících publikací: SEGMENT z [RA96], Výzkumná skupina pro přenositelné video ze Stanfordu (Portable Video Research Group at Stanford) PVRG-JPEG z [CAJ96], kruhová transformace (Rounding transform) z [JCP96], logické kódování z [CAJ96], Calic z [WM97]. Jedinou dostupnou implementaci, kterou se nám podařilo sehnat je od algoritmu Calic. Všechny údaje u ostatních komprimačních algoritmů byly převzaty z příslušných článků.

Dále byly k porovnání účinnosti komprese použity dobře známé formáty se zabudovanou kompresí (PCX, GIF) a komprimační algoritmy pro 1-DOF data (ARJ, ZIP). Programy pro komprimaci do těchto formátů je možno nalézt na internetu, poněvadž se jedná o shareware.

Metoda/Obrázek	Lena [%]	Baboon [%]	Boats [%]
PVRG-JPEG	22,7-29,4	6,7-13,7	24,9-33,1
Logic	26,1	12,6	29,3
SEGMENT	*54,6*	*36,3*	prázdný
Rounding Tr.	37,6	prázdný	prázdný
PNG	20,7	*41,7*	33,7
FH-Adapt	**34,0**	**19,1**	**41,2**

Tabulka 4.6: Porovnání účinnosti komprese mezi jednotlivými metodami.

Podle hodnoty CE v tab. 4.6 lze porovnat navrhovanou metodu s metodami ostatními. Bohužel nemáme pro většinu porovnávaných algoritmů funkční implementaci, a proto některé položky v tabulce zůstávají prázdné. Lepších výsledků dosahuje algoritmus SEGMENT a v případě obrázku opice Baboon obr. 4.12(a) též PNG. U obrázku Baboon si poměrně malá sonda prediktoru obtížně poradí s komplikovanou strukturou srsti opice. Proto u obrázku opice dochází ke zhoršení kompresního poměru.

Komprimační algoritmus PNG má podobnou strukturu jako kodér IBM Q80 [ISO94] a navíc dosahuje o něco lepších výsledků. IBM Q80 kodér je v dokumentaci označován jako zastaralý, nebude již déle podporován a není možno sehnat funkční implementaci.

Typ dat	Lena [Bajtů]	CE Lena [%]	Drawing [Bajtů]	CE Drawing [%]
Nekomprimován	262656	0	106940	0
ZIP	225007	15,62	10224	90,44
bzip2	185213	29,48	8475	92,07
ARJ	228389	14,35	10380	90,29
PCX	274256	-2,85	38624	63,88
GIF	278771	-4,54	17290	83,83
FH-Adapt.	**175434**	**34.21**	**7297**	**93.18**

Tabulka 4.7: Porovnání účinnosti běžných komprimačních algoritmů pro případ obrazových dat s metodou FH.

V tab. 4.7 je ukázáno porovnání výsledků naší komprimační metody *FH-Adapt* s ostatními běžně dostupnými metodami. Metoda *bzip2* byla popsána v [Pav98]. Velmi dobrých výsledků je dosaženo pro případ obrázku Drawing obr. 4.13.

4.8.2 Časová náročnost algoritmu

Z hlediska výpočetní náročnosti je metoda *FH-Adapt* silně asymetrická. To znamená, že doba komprimace je delší, než doba dekomprimace. Tato vlastnost je výhodná zejména pro archívy obrázků. Obrázky postačí jednou zkomprimovat a poté jsou s nižší náročností mnohokrát čteny.

Vlastní měření bylo provedeno na počítači s procesorem Intel Pentium 120MHz a operačním systémem DOS. Testovaný obrázek byl uložen na RAM disku, aby byl eliminován náhodný a pomalý přístupu k datům na disku. Čas potřebný k uložení dat na RAM disk včetně komprimace a k jejich opětnému načtení s dekomprimací je pro několik testovacích obrázků obsažen v tab. 4.8. Poznamenejme, že načtení anebo uložení obrazových dat bez komprimace trvá na uvedené konfiguraci počítače 0,1 vteřiny.

Obraz	Uložení s komprimací	Načtení s dekomprimací
Lena 256×256	2,6s	1,1s
Lena 512×512	8,2s	3,6s
Baboon	8,1s	3,5s
Boats	13,6s	6,7s
Drawing	3,4s	1,2s

Tabulka 4.8: Doba komprimace/dekomprimace jednotlivých obrázků metodou FH-Adapt.

Kapitola 5

Návrh komprimace obrázků s paletou

Vzhledem ke špatné korelovanosti dat obsahujících pouze indexy do palety není vhodné komprimovat pseudobarevné obrázky s paletou stejným algoritmem jako obrázky šedotónové. Této problematice je věnována sekce 5.2. Podrobnější vysvětlení struktury paletových obrázků lze nalézt např. v knize [Ska93].

V této kapitole je navržena úprava komprimačního algoritmu z kapitoly 4 pro paletové obrázky. Jedná se o výpočetně efektivní metodu, která k původní matici indexů a paletě nalezne novou matici indexů spolu s paletou plně automaticky tak, že ve většině případů dojde ke zvýšení kompresního poměru matice indexů.

Použití uvedené metody nemá smysl pro 0-DOF komprimační algoritmy (Huffman) a některé 1-DOF algoritmy (PCX, GIF) založené na opakování řetězců. Avšak po její aplikaci před jmenovanými algoritmy nedojde ani ke zhoršení kompresního poměru.

5.1 Definice nových pojmů

Nosič rastrového obrazu je definován následujícím způsobem: $T=\{(x,y)\colon 0 \leq x < M;\ 0 \leq y < N\}$.

Paletový obraz lze považovat za zobrazení $barva = [R,G,B] = paleta(f(x,y))$, kde $[R,G,B]$ reprezentuje individuální barevné složky, které odděleně představují tři intenzitní obrazy. Výstupem funkce $f(x,y)$ je index. Z tohoto důvodu funkci f nazýváme *paletová indexová funkce*. Dále v textu ji nazýváme zkráceně *indexová funkce*.

Paleta je tabulka (look up table) s [R,G,B] položkami. Pro navrhovanou metodu přeuspořádání indexů na obsahu položek v paletě v podstatě nezáleží a může být jakýkoliv (např. CMY, CMYK aj.). [R,G,B] bylo uvedeno jako nejčastěji používaný příklad. Velikost jednotlivých položek palety se může též měnit případ od případu. Nejčastěji užívané hodnoty jsou 256 úrovní pro každou složku RGB

(struktura 3*8 bitů); 65536 úrovní pro jednotlivé složky RGB (struktura 3*16 bitů).

5.2 Shrnutí předchozích prací

Popisované algoritmy, které nejsou specializovány pro pseudobarevné obrázky s paletou, dosahují nejlepších výsledků pro šedotónové obrázky, protože v jejich modelu dat je velmi často implicitně zahrnut požadavek na spojitost obrazové funkce. Obrazová funkce většinou spojitá bývá, pokud na pseudobarevné obrázky nahlížíme přes paletu jako na RGB barevné obrázky. Pracovat s daty z pseudobarevných obrázků, které byly převedeny do plných barev, není vždy nejlepší vzhledem k citelnému nárůstu jejich velikosti. Komprimační algoritmy založené na tomto přístupu nedosahují dobrých výsledků.

5.2.1 Typy pseudobarevných obrázků

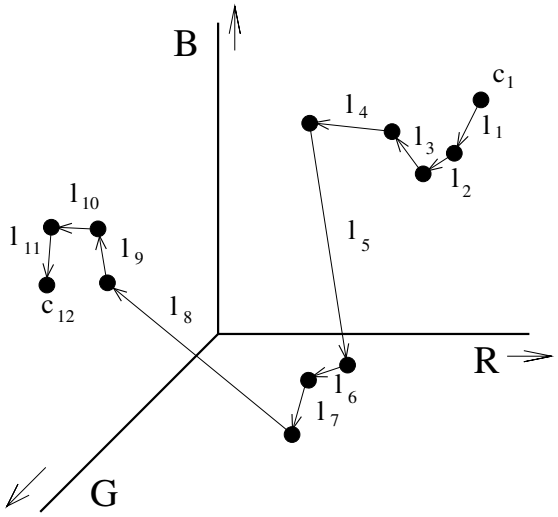
Existují dva základní typy pseudobarevných obrázků:

Uměle vytvořené. Do této kategorie spadají ručně kreslené obrázky a výstupy z programů pro tvorbu technických výkresů. K obrázkům tohoto typu patří obrázky z komiksů nebo jednotlivé obrázky z kreslených filmů. Dále je možno sem zařadit různé uměle vytvořené fraktální obrazce. U obrázků patřících do této kategorie je počet barev obvykle nižší a pohybuje se od 16 do 256 barev.

Vzniklé kvantováním z plně barevných RGB obrázků. Lidské oko není schopno rozlišit 16,7 miliónů barev. Při snížení počtu barev na 256 lze dosáhnout uspokojivé napodobeniny původního obrázku. Někdy se používá více barev, např. 65536. Algoritmy pro transformaci RGB $\rightarrow$ paleta jsou popsány např. v [Ska93]. V popisovaných algoritmech většinou není obsažen algoritmus pro uspořádání indexů na výstupu a výsledná funkce tvořená z indexů je ve většině případů silně nespojitá. Paleta může být buď univerzální pro nějakou třídu obrázků anebo dynamicky generovaná pro každý konvertovaný RGB obraz zvlášť.

5.2.2 Způsoby komprimace pseudobarevných obrázků od jiných autorů

Postup při komprimaci pseudobarevných obrázků se velmi často téměř shoduje s postupem komprimace šedotónových obrázků. Paleta je uložena zvlášť a s maticí indexů je nakládáno stejně jako by se jednalo o data šedotónových obrázků. Tento



Obrázek 5.1: Způsob řazení indexů v paletě.

přístup sice umožní dosažení určitého kompresního poměru, ale indexová data jsou velmi často v této podobě špatně komprimovatelná.

Je zapotřebí komprimovatelnost dat nějakým způsobem kvantifikovat a použít exaktní kritérium pro její ohodnocení. Míra korelovanosti dat je dána entropií prvního řádu (first order entropy). Abeceda S obsahuje K prvků, které představují možné indexy do palety. *Entropie prvního řádu* H_1 je definována na abecedě D, která se skládá z $2K-1$ možných hodnot rozdílů mezi prvky abecedy S. Detailnější popis je obsažen v [HS94].

$$H_1 = -\sum_{j=1}^{2K-1} P(D_j) log_2 P(D_j). \tag{5.1}$$

Tuto konstantu nazýváme zjednodušeně hladkost obrazové funkce a do jisté míry na její hodnotě závisí kompresní poměr. Protože při komprimaci nakládáme s indexovou funkcí jako s funkcí obrazovou, ohodnocujme též indexovou funkci pomocí entropie prvního řádu. Naštěstí lze hladkost indexové funkce, a tím i kompresní poměr, o něco zvýšit přerovnáním indexů a jim odpovídajících položek v paletě. Po přerovnání indexů spolu s položkami v paletě bude celý pseudobarevný obrázek navenek vypadat stejně, přičemž bude možno dosáhnout lepšího kompresního poměru.

Nové setřídění palety je vhodné využít zejména v následujících případech:

A. Nové uspořádání indexů za účelem zvýšení kompresního poměru.

B. Přerovnání indexů za účelem zlepšení lidského vnímání v případě nahrazení barvy odstíny šedi. Tato metoda je založena na vynesení indexů do RGB

prostoru vyhledání minimální spojnice (s délkou l) mezi nimi $l = \sum_{i=1}^{K} l_i$, kde K je počet indexů. Pro ilustraci této metody viz obr. 5.1.

C. Pro snížení počtu indexů v případě ztrátové komprimace pseudobarevného obrázku.

D. Přerovnání indexů pro možnost steganografického[1] ukrytí dat do pseudobarevného obrázku [NJ98].

První dva případy jsou diskutovány v [HS94]. Jejich požadavky na algoritmus pro uspořádání indexů nejsou úplně protikladné. V této práci pojednávající o kompresi bude věnováno nejvíce prostoru prvnímu případu.

Nejjednodušší způsob, jak udělat přerovnání indexů pro účely komprimace je popsán v [Fry93]. Ke každému indexu je přidána jasová hodnota Y, která je vypočtena podle vzorce $Y = 0.3R + 0.6G + 0.1B$ nebo ve zjednodušené variantě $Y = R+G+B$. Podle jasu jsou setříděny všechny položky v paletě a jim odpovídající indexy. Nepoužité položky v paletě mohou být vyřazeny a položky se stejnou hodnotou mohou být sloučeny. Matice s přerovnanými indexy je komprimována stejným způsobem jako šedotónová data.

Nejbližší příspěvek ke zde navrhovanému algoritmu je uveden v [MV96]. Přerovnání indexů je formulováno jako optimalizační úloha. Pro její vyřešení byly navrhnuty a ověřeny tři heuristické algoritmy. Dva z nich jsou velmi výpočetně náročné vzhledem k použití simulovaného žíhání. Třetí řešení využívající hladový algoritmus dosahuje nižší výpočetní náročnosti za cenu zhoršení kompresních poměrů. Metoda je optimalizována pro případ, kdy jako následný komprimační algoritmus indexů je použit lineární prediktor pro bezeztrátovou kompresi.

Další článek zabývající se problémem bezeztrátové komprimace pseudobarevných obrázků je [ZL93]. Algoritmus začíná vytvářením optimální nejkratší cesty vedoucí přes RGB (nebo LUV) barevný prostor a procházející všemi vyznačenými body. Cesta přes RGB barevný prostor je zobrazena na obr. 5.1.

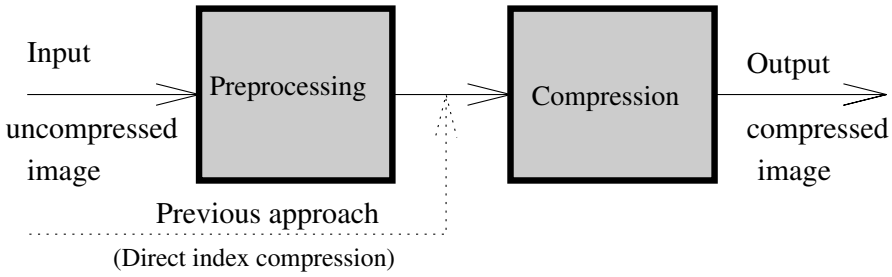
Pro redukci počtu barev jsou barvy vzájemně blízké na vytvořené cestě seskupovány do shluků. Každý shluk je převeden na novou barvu.

5.3 Formulace úlohy

Jak již bylo v sekci 5.2 popsáno, pseudobarevné obrázky s paletou mají většinou vysokou entropii 1. řádu (hladkost) indexové funkce. Zejména algoritmy pro kvantování RGB obrázků (např. z kamery nebo barevného scanneru) se příliš nezabývají způsobem řazení položek palety.

Základní myšlenka spočívá ve výrazném zvýšení hladkosti indexové funkce $f(x, y)$ před vlastní komprimací. Předkládaný algoritmus je předřazen komprimačnímu kroku viz obr. 5.2. Cílem je, aby bylo možno upravenou indexovou

[1] Steganografie je nauka zabývající se ukrýváním dat do jiných dat.



Obrázek 5.2: Navrhovaný postup pro kompresi pseudobarevných obrázků.

funkci úspěšně komprimovat algoritmem původně určeným pro šedotónové obrazy. Nově vytvořená indexová funkce $f'(x, y)$ spolu s jí odpovídající tabulkou palety *paleta'* vznikla takovou modifikací původní indexové funkce a palety, že všechny výsledné barvy - tzn. hodnoty $[R, G, B]$ - jsou stejné ve všech pixelech původního a modifikovaného obrazu (5.2).

$$paleta(f(x, y)) = paleta'(f'(x, y)). \tag{5.2}$$

Algoritmy pro přeuspořádání palety, s nimiž jsem se dosud setkal, byly optimalizovány pouze pro lineární prediktor. Navrhovaný komprimační algoritmus (viz kapitola 4) pracuje s jednotlivými bitovými rovinami. Proto bylo potřeba vytvořit úplně nový algoritmus vhodný pro předzpracování dat před vlastní kompresí.

Matice indexů $f(x, y)$ u obrázků s paletou má stejné uspořádání dat jako u šedotónového obrázku. Je samozřejmé, že jednotlivé položky mají jinou interpretaci. Každý rastr obsahující buňky, které mohou nabývat více hodnot než 0 a 1, lze rozložit na více bitových rovin. Toto lze udělat vzhledem k možnosti převodu jakéhokoliv čísla do dvojkové soustavy. Z tohoto důvodu je možno optimalizovat jednotlivé bitové roviny odděleně i v případě obrázků s paletou.

5.3.1 Navrhované řešení

Navrhovaná metoda se skládá z následujících kroků:

A. Provedení statistické analýzy sousedností jednotlivých pixelů.

B. Počáteční odhad rozdělení množiny indexů do 2 podmnožin s co možná nejnižší entropií prvního řádu.

C. Minimalizace entropie prvního řádu H_1 v matici indexů.

Minimalizace entropie prvního řádu (kroky B a C) je prováděna postupně pro jednotlivé bitové roviny. Algoritmus nejprve provede optimalizaci nejvyšší bitové roviny, a pak postupně pokračuje do nižších bitových rovin ležících pod nejvyšší

bitovou rovinou. Minimalizace entropie prvního řádu v nejvyšší bitové rovině se projeví nárůstem entropie prvního řádu v nižších bitových rovinách.

Nastíněný postup je výhodný zejména pro kompresní algoritmy, které zpracovávají obrazová data po bitových rovinách. U našeho algoritmu není kompresní poměr pouze lineárně závislý na entropii prvního řádu, ale čím je entropie prvního řádu nižší, tím (více než lineárně) je kompresní algoritmus úspěšnější. Proto není vhodné globálně ohodnocovat entropii 1. řádu pro všechny bitové roviny současně. Další výhodou kompresního algoritmu pracujícího po bitových rovinách je možnost neprovádět kompresi poslední bitové roviny, do které přejde veškerý šum obsažený v datech.

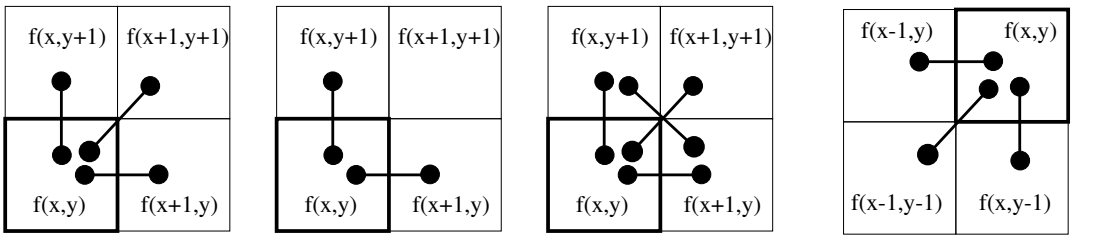
Námitka, že by bylo vhodné sledovat způsob seskupení indexů ve vyšší bitové rovině při zpracovávání nižší bitové roviny je sice oprávněná, ale algoritmus by se tím zbytečně komplikoval a stal by se velmi závislým na následném komprimačním algoritmu. Tato cesta by na druhou stranu mohla vést pro daný komprimační algoritmus k dalšímu zlepšení kompresního poměru.

5.4 Popis algoritmu pro přeindexování palety

5.4.1 Tabulka sousedností indexů

Tabulka sousedností indexů je datová struktura, jež obsahuje informace o vzájemné pozici jednotlivých indexů. Pro její vytvoření je potřeba definovat relaci sousednosti *rel* dvou indexů v rastru.

Nechť u a v jsou dvě hodnoty (tedy indexy do *palety*) z indexové funkce $f(x, y)$ umístěné ve dvou různých pixelech, které spolu sousedí. Relace sousednosti je symetrická. Z důvodů obsažení celého rastru a neopakování téže relace vícekrát je jeden prvek z relace považován za aktuální (x, y) a sousednost je vyhodnocována vzhledem k tomuto prvku. Pro zpracování celé indexové funkce se každý její prvek (s výjimkou okrajů) stane na chvíli aktuálním.



(a) Tři spony (b) Dvě spony (c) Čtyři spony (d) Tři spony obráceně

Obrázek 5.3: Konfigurace spon pro vyhodnocování relace sousednosti.

Několik způsobů vytvoření okolí aktuálního prvku je zobrazeno na obr. 5.3. Podrobnější diskuse o způsobech tvoření relace sousednosti je uvedena v oddílu 5.4.2.

Pro popis relace dvou sousedních indexů je užito termínu spona (anglicky clasp). Intuitivního názvu *spona* bylo použito z důvodu připomenutí si skutečnosti, že spona představuje předmět pro spojení dvou věcí. V našem případě spojuje dva indexy dohromady, a tím vlastně i celé oblasti vytvořené shluky indexů. Relace *rel* je symetrická, a proto platí $clasp(u, v) = clasp(v, u)$. Tabulka sousedností je vytvořena podle jednotlivých spon. Pro vyčíslování hladkosti indexové funkce $f(x, y)$ lze použít jednoduché statistiky založené na základě počtů spon. Globální informace o sponách je obsažena v tabulce sousedností.

Zajímá nás počet výskytů relace sousednosti *clasp* v celé indexové funkci $f(x, y)$ příslušející jednoznačně obrázku. *Tabulka sousedností* $\mathbf{S}(i, j)$ je datová struktura, která je vytvořena takovým způsobem, aby se po průchodu celé indexové funkce do ní nakumulovaly veškeré potřebné informace o sponách. Tabulka sousedností $\mathbf{S}(i, j)$ je maticí se stejným počtem řádek a sloupců. Počet řádek (sloupců) je roven maximálnímu počtu indexů indexové funkce. Pro běžné paletové obrázky s 256 barvami je velikost $\mathbf{S}(i, j)$ 256×256. Položka na pozici $\mathbf{S}(u, v)$ nám říká, kolikrát spolu sousedí indexy u a v v obrázku. Statistická informace uložená v tabulce $\mathbf{S}(i, j)$ připomíná o něco obecnější *matici vzájemných souvislostí*(co-occurrence matrix), která je často používána při analýze textur [HS92].

Protože je relace sousednosti symetrická, je symetrická i tabulka sousedností indexů $\mathbf{S}$. Celá informace o sousednosti je uložena v tabulce sousedností 2 krát, a proto tabulka sousedností je symetrickou maticí. Pro uchování informace by postačila pouze horní nebo dolní trojúhelníková matice. Avšak z důvodů následných operací, které jsou prováděny v navrhovaném algoritmu, je vhodné použít celou čtvercovou matici.

Dále je dobré předpokládat, že maximální počet indexů ve zpracovávaném obrázku je mocninou čísla 2. Pokud není tato podmínka splněna, je možno si pomoci doplněním tabulky $\mathbf{S}$ do nejbližší vyšší mocniny 2 nulami. Výpočet s jinou velikostí tabulky je v zásadě možný na úkor výrazného zvýšení složitosti algoritmu, které by nastalo v důsledku vyřazení efektivních bitově orientovaných operací.

Hodnoty na hlavní diagonále nejsou navrhovaným algoritmem vyžadovány. Jedná se identickou relaci, která nám říká kolikrát sousedí stejný index se stejným indexem. V následujícím algoritmu jsou hodnoty na diagonále ignorovány. Krok optimalizace je popsán v oddílu 5.5.2. Pro zjednodušení výpočtu je možno všechny hodnoty na diagonále vynulovat.

Tvorba tabulky sousedností

Tabulku sousedností indexů $\mathbf{S}$ je možno vytvořit jediným průchodem maticí indexů[2]. V každém bodě matice indexů je použita jedna z masek konfigurací okolí aktuálního bodu. Pro příklad zvolme masku podle obr. 5.3(a) a předpokládejme,

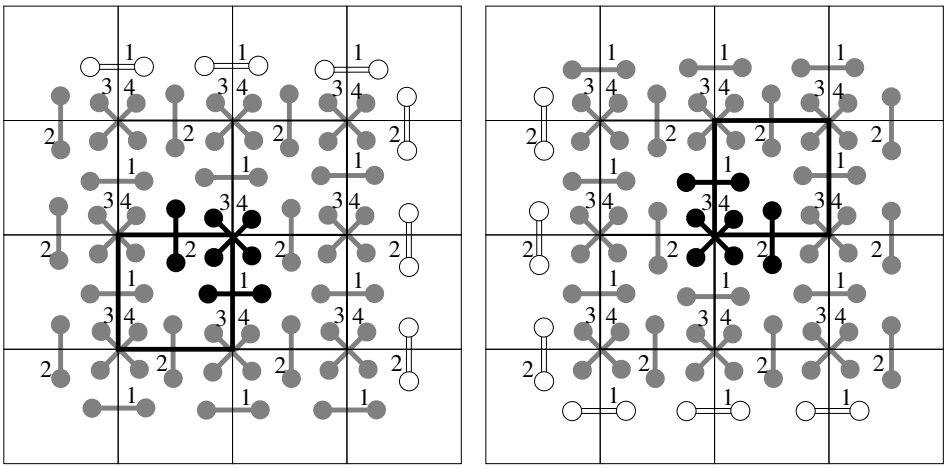
[2]Matice indexů představuje analogii obrazového rastru u pseudobarevných obrázků. Na místě uspořádaných pixelů jsou však umístěny indexy do palety.

že se aktuální pozice indexu nachází na souřadnicích $[i, j]$. Podle spon v masce je potřeba pro aktuální pozici inkrementovat následující položky z tabulky četností: $\mathbf{S}(f(i,j), f(i+1,j))$; $\mathbf{S}(f(i,j), f(i,j+1))$; $\mathbf{S}(f(i,j), f(i+1,j+1))$; a protože počítáme se symetrickou relací, je nutno z důvodu symetrie inkrementovat i jejich protějšky $\mathbf{S}(f(i+1,j), f(i,j))$; $\mathbf{S}(f(i,j+1), f(i,j))$; $\mathbf{S}(f(i+1,j+1), f(i,j))$. Výše uvedenou akci je potřeba udělat ve všech bodech rastru obsahujícího indexy.

> Celá statistika sousedností indexů v indexové funkci je umístěna v tabulce **S**. Při vhodném způsobu výpočtu se již nebude potřeba vracet k indexové funkci. Způsob optimalizace hladkosti indexové funkce je diskutován v sekci 5.5.

5.4.2 Ladění metody pomocí změny relace sousednosti indexů

V úvodu této kapitoly jsme tvrdili, že zvýšení hladkosti indexové funkce pomocí navrhované metody napomáhá ke zvýšení kompresního poměru pro většinu komprimačních algoritmů. U některých komprimačních algoritmů dochází jen k malému zlepšení. Pro dosažení optimální spolupráce s kompresním algoritmem lze velmi jednoduše doladit činnost algoritmu pro optimalizaci indexové funkce pomocí změny relace sousednosti.



(a) Přímé konfigurace (b) Obrácené konfigurace

Obrázek 5.4: Vytváření masky z jednotlivých spon.

Několik možných relací sousednosti je zobrazeno na obr. 5.3. Konfigurace relací byly pečlivě vybrány tak, aby žádná relace nebyla počítána dvakrát. Pro popisovanou metodu tento požadavek v odůvodněných případech splněn být nemusí. V odůvodněných případech může být jedna relace počítána tolikrát, kolikrát je zapotřebí. Nyní analyzujme, jakým způsobem lze vytvářet nové konfigurace relací sousednosti.

Na obr. 5.4 jsou zakresleny všechny možné spony. Jedna pozice je vybrána a zvýrazněna silným rámečkem. Do každé jednotlivé pozice, pro případ neopakování spony, je možno umístit nejvýše 4 spony. Tyto čtyři spony jsou pro názornost očíslovány čísly 1, 2, 3, 4. Spony umístěné v aktuální pozici jsou zvýrazněny černou barvou. Spony dosažitelné z jiných aktuálních bodů jsou šedé. Okrajové spony, které nejsou dosažitelné daným typem lokální konfigurace, jsou zobrazeny bíle.

Uživatel si může vybrat libovolnou kombinaci spon 1, 2, 3, 4. Podle provedených experimentů se ukazuje, že horizontální spona č. 1 a vertikální spona č. 2 mají vysokou informační hodnotu a neměly by být zbytečně vyřazovány.

Za povšimnutí stojí, že některé jiné konfigurace mohou dávat (až na okrajové indexy) stejné výsledky. Dvě takovéto konfigurace jsou zobrazeny na obrázcích obr. 5.4(a) a obr. 5.4(b). Všechny konfigurace odvozené od popisovaných dvou, které vznikly vyřazováním jedné nebo více spon, nezapočítávají jednu relaci dvakrát. Jediný rozdíl mezi konfigurací podle obr. 5.3(a) a konfigurací obr. 5.3(d) nastává u zpracování první nebo poslední řádky nebo sloupce. V případě potřeby je možno vynechanou řádku nebo sloupec doplnit do tabulky četností zvlášť, ale ve většině případů lze okrajové relace zanedbat.

Na první pohled by se zdálo, že konfigurace podle obr. 5.3(c) představuje nejlepší řešení pro všechny případy, poněvadž obsahuje úplnou množinu spon. Vzhledem ke struktuře použitého binárního prediktoru obr. 3.4 (popsaného v [Sch89]) se ukazuje výhodné vyřadit sponu č. 4 a použít obrácenou konfiguraci obr. 5.3(d). Obrácená konfigurace nepokrývá sponami nultý řádek a nultý sloupec stejně jako navrhovaný binární prediktor. Pro jiný typ prediktoru může být vhodnější použít jinou konfiguraci spon.

Ukázka výpočtu relace sousednosti

Demonstrujme celý postup na příkladu za účelem lepšího porozumění způsobu vytvoření tabulky sousedností indexů. Výchozí indexová funkce s indexy pohybujícími se v rozmezí od 0 do 7 je ukázána v tab. 5.1. Rozmístění indexů je pouze ilustrativní a nemá jiný význam.

Pro výpočet tabulky sousedností indexů byla využita konfigurace relací sousednosti podle obr. 5.3(d). Hotová tabulka sousedností indexů je zobrazena v tab. 5.3. Výsledná indexová funkce po optimalizaci je zobrazena v tab. 5.2. Pro názornost jsou ukázány pouze indexy a paleta byla vynechána. Je samozřejmé, že v reálném případě by bylo potřeba ještě přeuspořádat paletu.

5.4.3 Formulace přerovnání indexů jako optimalizační úlohy

Cílem je nalezení způsobu, jak lze přerovnat indexy za účelem zvýšení hladkosti indexové funkce. K tomu je potřeba stanovit vhodné kritérium ohodnocující inde-

1	3	5	7	5	3	1	0	0	0	0	2	2	2	2
3	1	0	0	0	0	0	0	0	0	0	0	2	2	2	2
5	0	1	0	0	0	0	0	0	0	0	0	2	2	2	2
7	0	0	1	0	0	0	0	0	0	0	0	2	2	2	2
5	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	2
4	0	4	4	0	0	0	0	0	0	1	0	0	0	4
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	6
4	4	4	4	0	0	0	0	0	0	0	0	0	1	0	4
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	2
4	4	4	4	0	0	0	0	0	2	4	6	4	2	1

Tabulka 5.1: Indexová funkce demonstračního testovacího obrázku.

0	4	5	7	5	4	0	1	1	1	1	2	2	2	2
4	0	1	1	1	1	1	1	1	1	1	1	2	2	2	2
5	1	0	1	1	1	1	1	1	1	1	1	2	2	2	2
7	1	1	0	1	1	1	1	1	1	1	1	2	2	2	2
5	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1
4	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1
0	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	2
3	1	3	3	1	1	1	1	1	1	1	0	1	1	1	3
1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	6
3	3	3	3	1	1	1	1	1	1	1	1	1	0	1	3
1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	2
3	3	3	3	1	1	1	1	1	2	3	6	3	2	0

Tabulka 5.2: Indexová funkce optimalizovaná navrhovaným algoritmem.

	0	1	2	3	4	5	6	7
0	467	58	17	6	44	8	3	4
1	58	14	3	6	0	0	0	0
2	17	3	27	0	2	0	0	0
3	6	6	0	0	0	4	0	0
4	44	0	2	0	6	0	2	0
5	8	0	0	4	0	0	0	4
6	3	0	0	0	2	0	0	0
7	4	0	0	0	0	4	0	0

Tabulka 5.3: Tabulka relací sousednosti indexů pro testovací obrázek.

xovou funkci. Počet možných uspořádání je obrovské číslo - $K!$, kde K představuje počet indexů. Z tohoto důvodu není možné vyzkoušet všechny možné kombinace a vybrat tu nejlepší.

V popisovaném řešení je na celou indexovou funkci pohlíženo jako na množinu bitových rovin se vzájemnými vztahy mezi nimi. V každém binárním obrázku lze najít mnoho černých nebo bílých oblastí. Naším cílem je proto ovlivnit počet osamocených oblastí tak, aby se v každé bitové rovině vyskytovalo co možná nejméně co možná největších oblastí.

Z důvodů výpočetní náročnosti není vhodné optimalizovat všechny bitové roviny současně. Zjednodušme si problém tím, že budeme optimalizovat po jednotlivých bitových rovinách. Tím nedojde k podstatnému zhoršení výsledku a navíc je tento postup v souladu s navrženou metodou komprese.

Nejvyšší bitová rovina je nejdůležitější, a proto je zpracována jako první. Al-

goritmus je navržen takovým způsobem, že nižší bitové roviny mohou být modifikovány podobným způsobem. Avšak je zaručeno, že všechny bitové roviny ležící nad aktuální bitovou rovinou zůstanou nezměněny.

Nyní se zabývejme náročností zpracování jediné bitové roviny. Jedná se o problém s kombinatorickou složitostí. Nechť K indexů (jedná se zatím jen o nejvyšší bitovou rovinu) je rozděleno do dvou skupin, tedy v každé skupině se nachází $k = \frac{K}{2}$ indexů. Množství možných rozdělení, tedy kombinací, je stále neakceptovatelné i v případě 256 indexů:

$$kombinace(8,8) = C_k^K = C_{128}^{256} = \frac{K!}{k!(K-k)!} = \frac{256!}{128!\,128!} \approx 0.58\,10^{76}\,. \qquad (5.3)$$

Pro výpočet je použita funkce *kombinace* se dvěma argumenty. První argument $MP = MaxPlanes$ definuje množství bitových rovin ($\log_2 K$) potřebných k zakódování všech K indexů a druhý argument $PP = ProcessedPlane$ odkazuje aktuální bitovou rovinu. Obecný zápis funkce pro vyhodnocení počtu možných uspořádání indexů v dané bitové rovině lze určit podle rovnice (5.4) pro případ stejného počtu prvků v obou skupinách G_1 a G_2.

$$kombinace(MP, PP) = \left(C_{2^{PP-1}}^{2^{PP}}\right)^{MP-PP}\,. \qquad (5.4)$$

5.5 Optimálně uspořádaná indexová funkce

V předchozím oddílu bylo předloženo odůvodnění přerovnání indexů v paletě za účelem zvýšení kompresního poměru. Pro větší názornost je vhodné se podívat na obrázek červených a zelených paprik obr. 5.15, ve kterém jsou úmyslně zaměněny hodnoty barvy za intenzity šedi podle velikosti indexu. Bitové roviny původní indexové funkce jsou umístěny v levém sloupci. Tzv. diskontinuity jsou přímo vidět v jednotlivých bitových rovinách indexové funkce $f(x, y)$. Za povšimnutí stojí, že i v nejvyšší bitové rovině #8 obr. 5.15 dochází k příliš mnoha změnám.

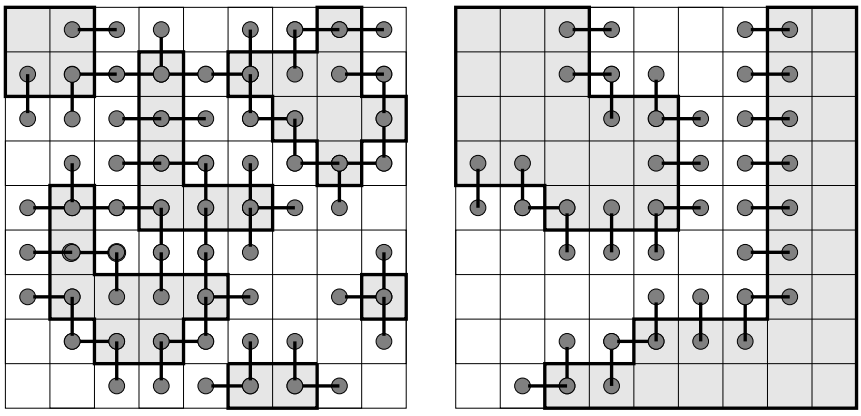
A nyní zaměříme svoji pozornost na pravý sloupec obrázku obr. 5.15. Ten obsahuje optimálně přeuspořádanou indexovou funkci rozloženou na jednotlivé bitové roviny. Bitové roviny vypadají na první pohled mnohem "čistější". Tomuto intuitivnímu vjemu říkáme hladkost indexové funkce.

5.5.1 Popis optimalizačního algoritmu

Pro výpočet optimalizačního kritéria je použita tabulka sousedností indexů (upozorňuji na obr. 5.3). O relaci sousednosti indexů označované $clasp(u, v)$ je pojednáváno v oddílu 5.4.1.

Mějme indexy rozděleny do dvou disjunktních množin G_1 a G_2 se stejnou kardinalitou. Rozdělení je buďto libovolné anebo je výsledkem počátečního odhadu.

Požadavek na stejný počet prvků v každé skupině a počtu skupin rovnému dvěma není striktní. Je dán hlavně dvojkovou soustavou používanou na počítačích, jejíž užití napomůže tvoření efektivních algoritmů. Teoreticky by bylo možno počítat s libovolným počtem skupin a s libovolným počtem prvků ve skupinách. Ale všechny výpočty by byly zbytečně komplikované. Proto je vhodné počítat s co nejmenším počtem skupin, tedy se dvěma.



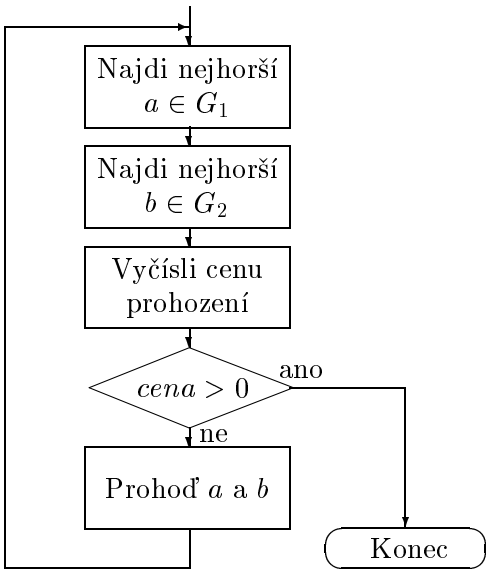
Nesetříděno Setříděno

Obrázek 5.5: Spony mezi skupinami v setříděné a nesetříděné bitové rovině.

Počet spon (relací) mezi množinami $clasp(u, v)$, $u \in G_1$, $v \in G_2$, $u \neq v$ nás informuje o jejich vzájemné hranici. Dále nás zajímá počet relací uvnitř množiny G_1 $\{clasp(u, v)$, $u \in G_1$, $v \in G_1$, $u \neq v$ $\}$ a podobně je tomu i u skupiny G_2. Z těchto čísel je možno vypočítat jakousi celkovou kvalitu rozdělení. Tato kvalita je nazývána v dalším textu *globální kritérium rozdělení indexů* do dvou skupin a je mu věnován oddíl 5.5.5. Počty relací též napomáhají k nalézání kandidátů pro vyřazení ze skupiny. Čím je indexová funkce hladší, tím je hladší a kratší i vzájemná hranice obou množin. Pro takto formulované kritérium je veškerá informace potřebná pro jeho výpočet obsažena v tabulce sousedností indexů **S** a není potřeba se vracet k indexové funkci.

Na obr. 5.5 je zobrazena jedna bitová rovina indexové funkce před a po setřídění indexů. Pro názornost jsou zakresleny spony mezi jednotlivými oblastmi. Za povšimnutí stojí, že počet černých (a tedy i bílých) buněk v jedné bitové rovině nemusí zůstat po optimalizaci zachován. Každému indexu totiž přísluší určitý počet buněk. Při prohození dvou indexů může dojít ke změně počtu bodů, které odpovídají jednotlivé skupině indexů (Z toho vyplývá, že není zachován počet černých a bílých bodů ve zobrazené rovině indexové funkce.) Spony jsou zakresleny podle konfigurace na obr. 5.3(b) vzhledem k jednoduchosti a přehlednosti vzoru.

Vývojový diagram algoritmu prohazování indexů je zobrazen na obr. 5.6. V každé skupině je nalezen index, jehož příslušnost k vlastní skupině je podle



Obrázek 5.6: Algoritmus prohazování indexů mezi dvěma skupinami

nějakého kritéria nejmenší. Ten je označen za kandidáta k výměně. Dva nejhorší indexy v obou skupinách jsou vzájemně prohozeny mezi skupinami. Při každém kroku je vyhodnoceno nové globální kritérium popisující rozdělení indexů do dvou skupin. Uvedený proces je opakován do té doby, dokud se snižuje globální kritérium. Pak je prohazování indexů zastaveno a algoritmus pokračuje v další nižší bitové rovině.

Vzhledem k množství lokálních extrémů může algoritmus uvíznout v lokálním minimu. Experimenty ukazují, že i jednoduchá optimalizace poskytne dobré výsledky. Uvedená jednoduchá metoda dává ve většině případů na svém výstupu mnohem hladší indexovou funkci. Pro vyhnutí se lokálním extrémům je použita poměrně úspěšná heuristika (viz následující text).

5.5.2 Počáteční odhad

Na konci oddílu 5.4.3 byl ukázán výpočet počtu všech možných kombinací uspořádání indexů v jedné bitové rovině. Díky kombinatorické složitosti vychází velmi vysoké číslo, a proto není v možnostech výpočetní techniky[3] vyzkoušet všechny možnosti.

Proto je volen optimalizační algoritmus stoupání do vrchu[4]. Aktuální výšku určuje kriteriální funkce. Kdyby byla kriteriální funkce hladká bez lokálních ex-

[3] Uznávám, že výpočetní technika udělala za deset let výrazný pokrok a další ještě jistě udělá. Ale pro řešení problémů s faktoriální složitostí to bude pořád nedostatečné.

[4] *Optimalizační algoritmus stoupání do vrchu* je termín pro určitý typ algoritmu, který hledá extrém. V našem případě algoritmus vyhledává lokální minimum.

trémů, pak by nezáleželo na výchozím bodu. Bohužel tomu tak ve většině případů není a ukazuje se, že lokálních extrémů bývá velmi mnoho. V takovém případě dává algoritmus hledání extrému nejlepší výsledky při vhodné inicializaci. Inicializace je tím lepší, čím je umístěna blíže ke globálnímu minimu. Z výše uvedených důvodů je třeba udělat kvalitní počáteční odhad.

Na počátku máme množinu K indexů a potřebujeme ji rozdělit do dvou skupin G_1 a G_2. Nejjednodušší a ve většině případů také poměrně špatné řešení spočívá v rozdělení do dvou skupin podle čísla indexu. Indexy $i < \frac{K}{2}$ připadnou do první skupiny a indexy $i \geq \frac{K}{2}$ jsou umístěny do druhé skupiny.

O mnoho lepší výsledky dává rozdělení indexů na světlé a tmavé podle barvy převzaté z palety. Tento způsob rozdělení indexů do dvou skupin je velmi podobný situaci, při které jsou indexy setříděny podle intenzity jim odpovídající položky v paletě. Protože se jedná o nenáročný způsob s uspokojivými výsledky, byl použit v naší implementaci.

5.5.3 Pohled na optimalizaci z hlediska vysokodimenzionálního prostoru

Každá permutace indexů představuje nějaký stav. Množinu všech možných stavů lze uspořádat do hypotetického stavového prostoru.

Není-li o vzájemném vztahu dvou stavů nic známo, pak simulované žíhání představuje nejlepší možný způsob pro určení lepšího uspořádání indexů [HS94, MV96]. Stavový prostor vzniklý v tomto případě má pomyslný počet rozměrů roven počtu stavů, tzn. $K!$. Jakákoliv informace o vzájemném vztahu dvou stavů výrazně snižuje počet rozměrů stavového prostoru. Je možno něco zjistit o vzájemném vztahu dvou stavů nebo nevíme vůbec nic?

Chtěli bychom měřit vzdálenost mezi dvěma stavy (možnými konfiguracemi indexů). Nalezneme-li nějakou vzájemnou závislost mezi indexy, umožní nám to definovat metriku v našem stavovém prostoru. Znalost metriky bude s výhodou použita pro měření vzdáleností mezi jednotlivými stavy.

Mějme dva různé stavy (dvě různá rozdělení indexů do dvou skupin). Definujme metriku jako minimální množství operací prohození indexů mezi skupinami, po jejichž provedení přejdeme z výchozího stavu do stavu cílového. Měření vzdálenosti opačným způsobem (tedy z druhého stavu do stavu prvního) je také možné a výsledná vzdálenost musí vyjít stejně. Aplikujeme-li posloupnost operátorů prohození vedoucí z cílového stavu do stavu výchozího v opačném pořadí, získáme posloupnost operátorů, která nás povede z druhého stavu do stavu prvního.

Předpokládejme, že indexy jsou rozděleny do dvou skupin se stejným počtem položek. Pak je možný počet prohození indexů v jednom stavu $(\frac{K}{2})^2 - 1$. Toto číslo je sice stále poměrně vysoké, ale je o mnoho nižší než $K!$.

Předpoklad stejné kardinality obou skupin není striktní. V případě rozdílné

kardinality obou skupin je nutno přidat k operátoru prohození další operátor přesunu indexu z jedné skupiny do druhé. Výsledná vzdálenost obou skupin je dána minimálním možným počtem všech operátorů, které musely být aplikovány na stav 1, abychom jej transformovali do stavu 2.

Zde uvedené přirovnání ke stavovému prostoru usnadňuje pochopení způsobu hledání lokálního extrému v prostoru s faktoriálním počtem stavů.

5.5.4 Kvalita indexu k pro skupinu.

Pro vyměňování indexů mezi skupinami je potřeba nějakým způsobem ohodnocovat, nakolik se určitý index hodí do své skupiny. Ohodnocení indexu by mělo být snadno určitelné, nejlépe bez nutnosti procházet mnohokrát celou indexovou funkci. Index, který se hodí do své skupiny nejméně, se stane kandidátem na vyloučení.

Každému indexu k je přiřazeno číslo w_k, které udává jeho příslušnost do vlastní skupiny. Definujme kvalitu indexu jako počet relací sousednosti w_k^+ s indexy z vlastní skupiny mínus počet relací sousednosti w_k^- s indexy z druhé skupiny. V dalším textu jsou čísla w_k^+ a w_k^- nazývána složkami čísla w_k. Takto definovaná *kvalita indexu* dává poměrně uspokojivé výsledky.

Relace sousednosti indexů je počítána pomocí spon. Číslo w_k^+ představuje počet spon (relací sousednosti indexů) mezi indexem k a ostatními indexy z téže skupiny. Naopak číslo w_k^- udává počet spon, jimiž je index k spojen s indexy příslušející do konkurenční skupiny. Vlastní kvalitu indexu k pak určíme rozdílem těchto dvou složek.

Patří-li index k do skupiny G_1, pak jeho příslušnost k vlastní skupině vypočteme podle vzorce:

$$w_k = w_k^+ - w_k^- = \sum_{i \in G_1;\, i \neq k} \mathbf{S}(i,k) - \sum_{i \notin G_1;\, i \neq k} \mathbf{S}(i,k)\,; \quad k \ \in G_1\,. \tag{5.5}$$

Podobně vypadá výpočet kvality indexu pro index k, který patří do skupiny G_2:

$$w_k = w_k^+ - w_k^- = \sum_{i \in G_2;\, i \neq k} \mathbf{S}(i,k) - \sum_{i \notin G_2;\, i \neq k} \mathbf{S}(i,k)\,; \quad k \ \in G_2\,. \tag{5.6}$$

Velikosti kvalit všech indexů vytvářejí vektor kvalit indexů $\vec{w} = (w_1, w_2, ..., w_k)$.

5.5.5 Globální optimalizační kritérium

Bylo ukázáno, jak určit kvalitu indexu pro vlastní skupinu a jak toto kritérium použít k výběru kandidátů na výměnu prvků mezi skupinami. Sama kvalita jednotlivých indexů je však nedostatečná pro posouzení vhodnosti prohození indexů mezi skupinami. Proto bylo potřeba zavést ještě globální kritérium, které provádí

ohodnocení způsobu rozdělení všech indexů do skupin. Při prohazování dvou indexů je potřeba sledovat změnu globálního kritéria. Je dokonce vhodné vybírat si takovou dvojici indexů, pro niž dojde k maximálnímu poklesu globálního kritéria. V podstatě lze použít mírně modifikovanou metodu stoupání do vrchu. Globální kritérium je též užitečné pro zjištění bodu, ve kterém má dojít k zastavení prohazování indexů. Každé další možné prohození dvou indexů by totiž vedlo jen ke zvýšení tohoto kritéria.

Pro výpočet globálního kritéria lze s výhodou použít tabulku sousedností indexů $\mathbf{S}(i,j)$. Tabulka sousedností byla ve skutečnosti ke snadnému výpočtu globálního kritéria přímo navržena. Definujme si globální kritérium. Globální kritérium je definováno jako součet spon uvnitř jednotlivých skupin mínus součet spon ležících mezi jednotlivými skupinami. Podobně jako kvalita samostatného indexu se i globální kritérium skládá z několika částí. Nejprve popišme jednotlivé složky globálního kritéria a nakonec uveďme definici globálního kritéria.

Pro každou skupinu lze vypočítat vlastní kritérium a globální kritérium představuje součet dílčích kritérií. Kritérium skupiny lze obdobně rozdělit na dvě části, počet spon uvnitř dané skupiny a počet spon s ostatními skupinami[5].

Pro dvě skupiny a nejvyšší bitovou rovinu jsou všechny čtyři složky globálního kritéria zobrazeny na obr. 5.7(a). V matici sousedností indexů jsou zvýrazněny oblasti přispívající ke čtyřem různým složkám globálního kritéria W_1, W_{21}, W_{12}, a W_2. W_1 je počet spon uvnitř skupiny G_1, W_2 je počet spon uvnitř skupiny G_2, W_{12} je počet orientovaných spon mezi G_1 a G_2 a W_{21} je počet orientovaných spon mezi G_2 a G_1.

Pro výpočet každé zmiňované složky globálního kritéria je potřeba sečíst všechny položky tabulky $\mathbf{S}(i,j)$, které leží v oblasti označené symbolem W_{xx} příslušející dané složce. Za povšimnutí stojí, že $W_{12} = W_{21}$ vzhledem k symetrii relace sousednosti indexů, která způsobuje též symetrii tabulky $\mathbf{S}$. Ze znalosti kvality každého izolovaného indexu ve skupině lze též vypočítat kvalitu celé skupiny. Výpočet spočívá v součtu kvalit všech jejích členů.

Rovnice pro výpočet jednotlivých dílčích složek globálního kritéria:

$$W_1 = \sum_{i \in G_1} w_i^+ \; ; \quad W_{12} = \sum_{i \in G_1} w_i^- \; ; \quad W_2 = \sum_{i \in G_2} w_i^+ \; ; \quad W_{21} = \sum_{i \in G_2} w_i^- \; . \qquad (5.7)$$

Pro nižší bitové roviny je možné, že položky patřící nějaké složce nebudou tvořit souvislou oblast, a proto se jedno značení může na ploše matice objevit i vícekrát, viz např. obr. 5.8. V takovém případě je potřeba sečíst všechny položky se stejným značením ze všech oblastí dohromady.

[5] Zatím máme pouze 2 skupiny. Je snaha psát popis tak obecně alespoň v základních částech, aby platil i pro více skupin.

Kvalita celých skupin tedy $Q(G_1)$ a $Q(G_2)$ se počítá ze svých složek následujícím způsobem:

$$Q(G_1) = W_1 - W_{12} \,, \qquad Q(G_2) = W_2 - W_{21} \,. \tag{5.8}$$

Globální optimalizační kritérium tzn. celková kvalita Q rozdělení indexů do dvou skupin G_1 a G_2 je definována takto:

$$Q = Q(G_1) + Q(G_2) = W_1 - W_{21} + W_2 - W_{12} \,. \tag{5.9}$$

Globální optimalizační kritérium Q lze též vypočítat jako součet všech kvalit indexů ze všech skupin. Jedná se vlastně o vyhodnocení dat z celé tabulky sousedností indexů $\mathbf{S}$.

Za povšimnutí stojí, že celý způsob výpočtu kvality rozdělení indexů byl navržen tak, aby byl striktně hierarchický z pohledu množství zpracovávaných dat. Množství dat je na každé vyšší úrovni zpracování redukováno. Na počátku do algoritmu vstupuje pseudobarevný obrázek s paletou. Z pseudobarevného obrázku je v dalším kroku vypočítána tabulka sousedností indexů. Vektor kvalit jednotlivých indexů $\vec{w}$ je možno určit pouze z tabulky sousedností indexů. Postupujeme dále ke globálnímu kritériu Q. Pro jeho určení postačí znalost vektoru kvalit jednotlivých indexů. Takto navržený způsob práce s daty usnadňuje výpočet snižováním objemu zpracovávaných dat. Poznamenejme, že velikost tabulky sousedností indexů má pevnou délku, která nezávisí na velikosti rastru. Pro velmi malé obrázky může být i větší než původní rastr.

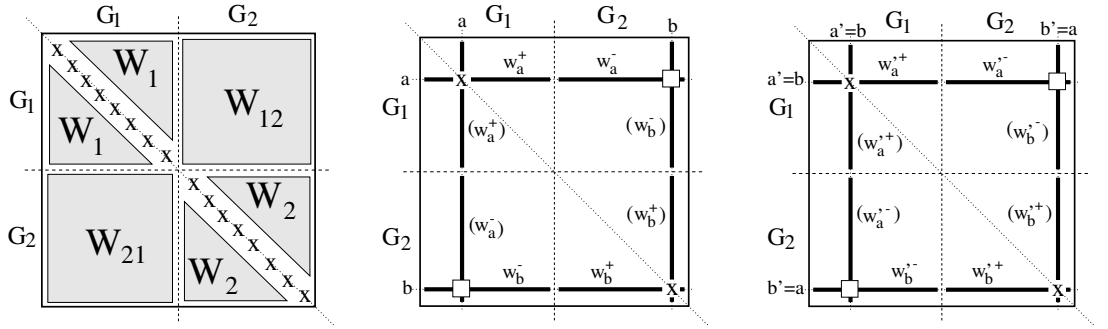
5.5.6 Algoritmus prohazování indexů

Mějme nalezeny takové dva indexy, které bychom chtěli prohodit. Pro jednoduchost si lze představit, že v každé skupině byl vybrán ten nejhorší index (s nejnižší hodnotou w_k). Takto zkonstruovaným algoritmem lze skutečně dospět k určitým výsledkům. Ale po mnoha experimentech se podařilo vyvinout lepší strategii - vybrat k indexu s nejnižší hodnotou w_k z obou skupin index ze sousední skupiny, tak aby došlo k co největšímu poklesu kritéria. Po prohození indexů se změní celý vektor kvalit indexů. Analyzujme vzniklou situaci, aby bylo možno efektivně vypočítat nový vektor kvalit indexů, bez nutnosti znovu přepočítávat všechna data z tabulky sousedností indexů.

Mějme dva indexy k prohození, každý je prvkem jiné skupiny $a \in G_1$ a $b \in G_2$. Operace prohození indexů je ukázána na obrázcích obr. 5.7(b)(c).
Nové hodnoty kvalit prohozených indexů jsou označeny čárkou. Číselné hodnoty lze určit z následujících rovnic:

$$\begin{aligned} w'^{+}_a &= w^{-}_b - \mathbf{S}(b,a) \quad ; \qquad w'^{-}_a = w^{+}_b + \mathbf{S}(b,a) \,; \\ w'^{+}_b &= w^{-}_a - \mathbf{S}(a,b) \quad ; \qquad w'^{-}_b = w^{+}_a + \mathbf{S}(a,b) \,. \end{aligned} \tag{5.10}$$

$$w'_a = w'^{+}_a - w'^{-}_a = -w_b - 2\mathbf{S}(b,a) \,; \qquad w'_b = w'^{+}_b - w'^{-}_b = -w_a - 2\mathbf{S}(a,b) \,. \tag{5.11}$$



(a) Vytvořené oblasti (b) Vybrané indexy a a b (c) Zaměněné indexy a a b

Obrázek 5.7: Prohození indexů v tabulce $\mathbf{S}$.

U symetrické relace spony platí $\mathbf{S}(a,b) = \mathbf{S}(b,a)$, což umožní zjednodušit výpočet globálního kritéria. Důkazu zbytečnosti asymetrické relace je věnována samostatná sekce 5.7.

Dále je potřeba vyšetřit změnu globálního optimalizačního kritéria Q po prohození indexů. Zaveďme si pomocné kritérium Q^*, které obsahuje ohodnocení celé tabulky sousedností indexů s výjimkou sloupců a řádek, které se vztahují k prohazovaným indexům a a b. Tímto trikem rozdělíme globální kritérium na dvě části. První část je závislá na prohazovaných indexech a druhá, označená symbolem Q^*, závislá není.

$$Q = Q^* + 2\,(w_a + w_b) \quad \rightarrow \quad Q^* = Q \; - 2\,(w_a + w_b)\,. \tag{5.12}$$

Nyní prohoďme dvojice indexů. Nová kritéria si označme čárkou např. Q'. Za povšimnutí stojí, že Q^* a Q'^* jsou stejné, poněvadž změna indexů a a b se projeví pouze na hodnotách w_a^+, w_a^-, w_b^+, w_b^-. Situaci po prohození indexů zachycují následující rovnice:

$$\begin{aligned} Q' = Q'^* + 2\,(w'_a + w'_b) \;\rightarrow\; Q'^* &= Q' - 2\,(w'_a + w'_b)\,; && (5.13)\\ Q'^* &= Q' - 2\,(-w_b - w_a - 4\,\mathbf{S}(a,b))\,; && (5.14)\\ Q'^* &= Q' + 2\,(w_a + w_b) + 8\,\mathbf{S}(a,b)\,. && (5.15) \end{aligned}$$

Více než absolutní hodnota globálního kritéria nás samozřejmě zajímá změna k níž dojde po prohození indexů. Pro výpočet změny optimalizačního kritéria stačí vyhodnotit rozdíl rovnic (5.15) a (5.12):

$$\begin{aligned} 0 \;=\; Q'^* - Q^* &= Q' - 2\,(w_a + w_b) - (Q \; + 2(w_a + w_b) + 8\,\mathbf{S}(a,b))\,; && (5.16)\\ Q \;-\; Q' &= \quad - 4\,(w_a + w_b) - 8\,\mathbf{S}(a,b)\,. && (5.17) \end{aligned}$$

Při prohazování dvou indexů je potřeba udělat několik výpočtů za účelem určení nového globálního optimalizačního kritéria. Všechny tyto akce nazýváme

souhrnně termínem transakce. Jedna transakce zahrnuje přepočítání vektoru kvalit indexů $\vec{w}$ a výpočet nové hodnoty globálního kritéria Q.

Pokud při transakci dojde ke snížení globálního optimalizačního kritéria, pak platí ($Q' < Q \quad \rightarrow \quad 4\Delta = Q' - Q < 0$). Proměnná Δ zachycuje počet spon mezi skupinami, které se v obrázku objeví v případě $\Delta > 0$ nebo zmizí pro $\Delta < 0$. Konstanta 4 vychází ze skutečnosti, že při změně kritéria Q se projeví jedna hraniční spona vícekrát. Pro přímý výpočet hodnoty Δ je možno vyjít z rovnice (5.17) čímž dostaneme rovnici (5.18).

$$\frac{(Q' \; - \; Q)}{4} \;\; = \;\; \Delta = w_a + w_b + 2\,\mathbf{S}(a,b)\,. \tag{5.18}$$

Všimněte si, že pro výpočet hodnoty Δ jsou potřeba pouze hodnoty w_k a nikoli jejich složky w_k^- a w_k^+. V každé iteraci je hledán takový pár indexů, pro který je hodnota Δ minimální. Je samozřejmě možné vyzkoušet všechny možné dvojice indexů z obou skupin. Ukazuje se, že takový algoritmus je zbytečně výpočetně náročný. Navrhovaný algoritmus má složitost $O(k\,n)$, zatímco ověření všech dvojic lze provést s výpočetní náročností $\left(\frac{n}{2}\right)^2$.

V naší implementaci je použit o něco jednodušší, avšak poměrně účinný algoritmus (Podle provedených experimentů se zdá, že je o něco méně náchylný k uváznutí v lokálním minimu než algoritmus předchozí.), který je založen na vyhledání dvou nejhorších prvků, každého v jiné skupině. K nalezeným prvkům je dohledán protějšek ve druhé skupině tak, aby Δ bylo minimální pro daný pár. Ze dvou párů je vybrán pro výměnu ten, který dosahuje nižší hodnoty Δ. Pokud se náhodou stane, že oba páry splynou v jeden, tzn. že k nejhoršímu prvku v jedné skupině je dohledán nejhorší prvek ve druhé skupině, algoritmu to rozhodně nevadí a vybere se tento jediný pár.

Indexy se prohazují tak dlouho, dokud dochází ke snižování optimalizačního kritéria. Přesněji pokud existuje alespoň jedna taková kombinace dvou indexů $a \in G_1$ a $b \in G_2$, pro niž platí $\Delta < 0$. Pokud nelze v aktuální iteraci snížit hodnotu optimalizačního kritéria, dojde k zastavení algoritmu.

Algoritmus sice provádí výpočet po iteracích, ale k jeho zastavení musí dojít po konečném počtu iterací. Teoreticky je maximální počet iterací dán omezením absolutní hodnoty velikosti globálního kritéria Q. Kritérium Q nemůže být větší, než součet všech spon v obraze. Hodnota kritéria nemůže poklesnout pod záporně vzatý součet všech meziskupinových spon a spon uvnitř skupin v obraze. Nejvyšší možný počet spon je dán vztahem $MaxSpon = (M-1)(N-1)Spon1$, kde konstanta $Spon1$ obsahuje počet spon v použité konfiguraci. Všechny spony mezi skupinami pro jeden konkrétní obrázek jsou zakresleny na obr. 5.5 a pro jednotlivé konfigurace viz též obr. 5.3. V každé iteraci musí dojít ke snížení hodnoty Q. Minimální možné snížení Q je o čtyři. Čtyřka vychází ze skutečnosti, že v množině viditelných (meziskupinových) spon jedna spona ubyde a v množině neviditelných spon se jedna spona objeví a tyto jsou počítány 2x.

Z předchozích údajů lze určit horní mez počtu iterací, která je rovna $MaxSpon$. Provedený odhad je velmi pesimistický a ve skutečnosti dochází k zastavení algoritmu po několika málo desítkách iterací.

Další výpočty prováděné během transakce

Při prohazování indexů je potřeba též přepočítat všechny hodnoty vektoru kvalit indexů $\vec{w}$. V předchozím textu byl pouze uveden způsob výpočtu nových hodnot w_a a w_b.

Další položky vektoru kvalit indexů $\vec{w}$ lze aktualizovat následujícím způsobem:

$$\begin{aligned} w'_x &= w_x - 2\mathbf{S}(x,a) + 2\mathbf{S}(x,b)\,, \quad x \in G_1; x \neq a\,; \\ w'_x &= w_x - 2\mathbf{S}(x,b) + 2\mathbf{S}(x,a))\,, \quad x \in G_2; x \neq b\,. \end{aligned} \tag{5.19}$$

Výpočetní náročnost jedné transakce

Zabývejme se výpočetní složitostí algoritmu. Již bylo řečeno, že výpočet probíhá po jednotlivých iteracích. Všechny akce provedené v průběhu jedné iterace byly nazvány *transakce*. Jako první je potřeba znát výpočetní složitost jedné transakce.

Transakce zahrnuje prohození indexů a současné udržování aktuální hodnoty globálního kritéria Q a vektoru kvalit indexů $\vec{w}$. Globální kritérium lze vypočítat jako součet všech prvků z vektoru $\vec{w}$. Pro hledání dvou prvků na prohození však není potřeba znát absolutní hodnotu globálního kritéria. Znalost o kolik se dané kritérium změní v případě prohození indexů a a b je pro náš případ plně postačující. Navíc pro výpočet Δ není absolutní hodnota globálního kritéria Q potřeba. Proto výpočet Q není zahrnut do vyčíslení výpočetní složitosti transakce.

Počet operací nutný k provedení jedné transakce je možno zjistit na základě rovnic (5.19) a (5.11) a je shrnut v následující tabulce.

Sčítání a odečítání	Přístup se do tabulky
$2(n-2)\ +\ 2$	$2(n-2)\ +\ 2$

Číslo $(2n-2)\ +\ 2$ je úmyslně uvedeno v rozepsaném tvaru. Cena za nové určení kvalit prohazovaných indexů je 2 a $2(n-2)$ je cena za určení nových kvalit ostatních indexů.

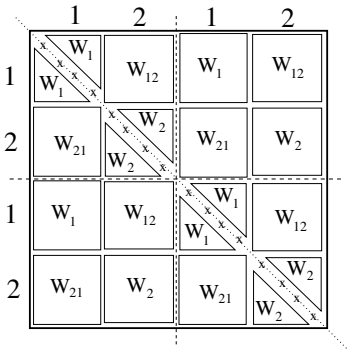
Z předchozího je zřejmé, že výpočetní složitost v rámci jedné iterace závisí pouze lineárně na počtu indexů n. Pro jednu iteraci již není potřeba žádný další výpočet. Do této úvahy však nebyla zahrnuta cena nalezení indexů a a b určených k prohození.

Náročnost jednoho ověření hodnoty Δ pro nějaké 2 indexy lze určit z rovnice (5.18). Jedná se o dvě sčítání a jeden přístup do tabulky. Nejjednodušší algoritmus pro hledání indexů a a b vyhledá dva indexy z různých skupin, oba s nejmenší

hodnotou w_k. Vyhledání vyžaduje $n-2$ operací porovnání. Lze navrhnout takový algoritmus pro hledání kandidátů k prohození s lineární náročností vyhledávání kandidátů závislou na číslu n.

Nejhorší algoritmus, který by vyzkoušel všechny možné dvojice (což se jeví ve většině případů naprosto zbytečné), má náročnost úměrnou $\left(\frac{n}{2}\right)^2$.

5.5.7 Nižší bitové roviny



Obrázek 5.8: Vyhodnocení **S** pro druhou bitovou rovinu.

Algoritmus při přerovnávání indexů postupuje od nejvyšší bitové roviny směrem k nižším bitovým rovinám. Požadavek na zachování všech vyšších bitových rovin beze změny mírně komplikuje optimalizaci dané bitové roviny. Proto je popisu algoritmu pro nižší bitové roviny věnován nový oddíl.

Původní zadání je stejné: Rozdělit indexy do dvou skupin tak, aby mezi oběma skupinami leželo co možná nejméně spon a téměř všechny spony byly umístěny uvnitř jednotlivých skupin. K tomuto cíli se však není možno tak jednoduše dostat vzhledem k vlivu vyšších bitových rovin, které musí zůstat beze změny.

Věnujme se případu druhé bitové roviny. Nechť je nejvyšší bitová rovina již optimálně rozdělena do dvou skupin G_1 a G_2. Chceme rozdělit prvky do jiných dvou skupin G_1^2 a G_2^2. Horní index znamená číslo bitové roviny, pro niž je uvažováno dělení do dvou skupin. Jednička nebo žádný index je volena pro nejvyšší bitovou rovinu. Z rozdělení je jasné, že část prvků ze skupiny G_1^2 bude ležet ve skupině G_1 (označme si ji G_{11}) a další část bude ležet uvnitř skupiny G_2 (označme si G_{21}). Podobná situace nastane i pro skupinu G_2^2. Označme si části nových skupin:

$$\begin{aligned} G_1^2 &= G_{11} \cup G_{21}, \\ G_2^2 &= G_{12} \cup G_{22}. \end{aligned} \tag{5.20}$$

Pro původní skupiny z vyšší bitové roviny pak platí:

$$\begin{aligned} G_1 &= G_{11} \cup G_{12}, \\ G_2 &= G_{21} \cup G_{22}. \end{aligned} \tag{5.21}$$

Vzhledem k podmínce nenarušitelnosti vyšší bitové roviny mohou být indexy vyměňovány pouze mezi skupinami G_{11} a G_{12} a také mezi skupinami G_{21} a G_{22}. Dočasně si nazvěme indexy patřící do jedné dvojice podskupin jako *volné*. Části G_{21} a G_{22} se stanou následně *blokované*. Množiny G_1^2 a G_2^2 pak obsahují volné a blokované prvky. Vlastní optimalizace indexů spočívající v jejich prohazování

mezi skupinami lze udělat pouze s volnými indexy. Avšak výpočet kvality indexů musí zahrnovat obě kategorie indexů. Proto všechny předchozí rovnice zůstanou nezměněny a jediná změna se týká způsobu výběru páru indexů k prohození.

Totéž platí i pro každou jinou další konfiguraci volných a blokovaných indexů. Skupiny G_{21} a G_{22} se mohou stát volnými a skupiny G_{11} a G_{12} se pak stanou blokovanými. Označme si skupinu jako volnou, pokud jsou všechny její prvky volné.

Optimalizuje se pro jedno nastavení volných a blokovaných indexů tak dlouho, dokud dochází ke snižování globálního optimalizačního kritéria. Poté je změněna konfigurace volných a blokovaných indexů a celý výpočet se opakuje. K zastavení algoritmu dojde v situaci, kdy nelze v žádné konfiguraci volných/blokovaných indexů takovým způsobem prohodit indexy, aby došlo ke snížení globálního optimalizačního kritéria.

Rozmístění jednotlivých dílčích složek globálního kritéria v matici **S** je ilustrováno na obr. 5.8. Složky mají naprosto stejný význam jako pro případ nejvyšší bitové roviny a je možno je přímo použít v rovnici (5.7). Uvedené rozdělení je platné pro druhou bitovou rovinu (ležící těsně pod nejvyšší bitovou rovinou).

Přístup k dílčím složkám **S** z obr. 5.8 vypadá poněkud komplikovaný. V případě stejné kardinality obou skupin a počtu indexů, který je zarovnán na mocninu čísla 2, je přístup k diskutovaným složkám poměrně jednoduchý. Vyžaduje navíc jen jednu operaci nonekvivalence XOR.

Dodatek pro bitové roviny pod druhou bitovou rovinou

Mohlo by se zdát, že se počet podskupin zvyšuje exponenciálně pro případ bitových rovin ležících pod druhou bitovou rovinou a tím dochází k výrazné komplikaci výpočtu. Tato situace však v případě navrhovaného algoritmu nenastává. Vždy můžeme každou skupinu rozdělit pouze na dvě podskupiny: jednu volnou a druhou blokovanou.

Je pravda, že v nižších bitových rovinách existuje velké množství výběrů volné skupiny. Všechny ostatní podskupiny jsou slity do jediné blokované podskupiny. Vlastní výběr kandidátů na prohození pak probíhá jen v rámci rozdělení na dvě podskupiny.

Extrémní situace nastává pro případ nejnižší bitové roviny. Poznamenávám, že i v nejnižší bitové rovině lze provádět optimalizaci. Volné podskupiny mají po jednom prvku a vše ostatní je zahrnuto do blokovaných podskupin. V této konfiguraci existuje jediná možnost prohození indexů. Ta buď vede ke snížení globálního optimalizačního kritéria a nebo ne. V prvním případě dojde k prohození indexů. Ve druhém případě se neděje nic. Poté je vytvořena nová volná podskupina a výpočet pokračuje tímto způsobem dále.

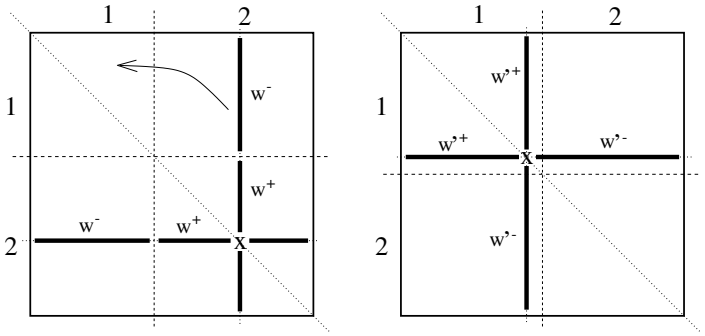
5.6 Skupiny s různou kardinalitou

5.6.1 Úvod do problematiky

V předchozím textu jsme pracovali se dvěma disjunktními skupinami se stejným počtem prvků. Stejný počet prvků v obou skupinách je velmi užitečný pro případ předzpracování obrazových dat za účelem zvýšení kompresního poměru.

V tomto oddílu je popsán způsob, jakým lze omezení na stejnou kardinalitu skupin odstranit. Celý přístup k přerovnávání indexů vytváří poměrně zajímavý systém manipulace s daty. Při požadavku na stejnou nebo neměnnou kardinalitu nám postačí pouze prohození dvou indexů. Při rozdílném počtu prvků ve skupinách je vhodné uvažovat navíc přemístění indexu z jedné skupiny do druhé. Tímto způsobem lze dynamicky měnit kardinalitu obou skupin.

5.6.2 Přesunutí indexu z jedné skupiny do druhé



(a) Vybraný index (b) Přesunutý index

Obrázek 5.9: Přesun indexu z jedné skupiny do jiné.

Jak již bylo v předchozím textu naznačeno, je možné přesunout index z jedné skupiny do druhé za současného udržování všech informací o počtu spon mezi skupinami a o kvalitách jednotlivých indexů pro obě skupiny. Nyní si definujme operátor, který vykoná tuto činnost.

Navrhovaný postup nevyžaduje výpočet globálního kritéria (popř. vektoru kvalit indexů) znovu od začátku, ale umožňuje jeho snadnou aktualizaci stejně tak, jako tomu bylo v případě prohození indexů. Navíc je možno dopředu velmi snadno zjistit změnu globálního optimalizačního kritéria pro případ přesunu jakéhokoliv indexu a vybrat si index, pro který dojde k největšímu snížení globálního optimalizačního kritéria.

Celá operace přesunu indexu je zobrazena na obr. 5.9, kde je schematicky nakreslena matice sousedností indexů **S**. Počáteční stav je zachycen na obr. 5.9(a). Naším cílem je vyjmout index s pořadovým číslem a ze skupiny G_2 a vložit jej do skupiny G_1, jak je ukázáno na obr. 5.9(b). Jednotlivé složky kvality indexu

a jsou označeny w^+ a w^-. Po přesunu indexu dojde k jejich změně následujícím způsobem (hodnoty po provedení přesunu jsou označeny apostrofem $'$):

$$w'^+ = w^- \; ; \quad w'^- = w^+ \; ; \quad w = w^+ - w^- \; ; \quad w' = w'^+ - w'^- = -w. \tag{5.22}$$

Chceme znát velikost globálního kritéria po provedení přesunu indexu. Globální kritérium lze rozložit do čtyř částí W_1, W_{12}, W_{21}, W_2 (je možno si připomenout obr. 5.7(a)). Analýza změny v těchto čtyřech částech je poměrně jednoduchá:

$$W'_1 = W_1 + 2w^- \; ; \quad W'_2 = W_2 - 2w^+ \; ; \quad W'_{12} = W'_{21} = W_{12} - w^- + w^+ \; . \tag{5.23}$$

A nyní se vraťme ke globálnímu kritériu:

$$Q = W_1 - W_{12} \; + \; W_2 - W_{21} \; . \tag{5.24}$$

$$\begin{aligned} Q' &= W'_1 - W'_{12} \; + \; W'_2 - W'_{12} = \\ &= W_1 + 2w^- - (W_{12} - w^- + w^+) \; + \; W_2 - 2w^+ - (W_{21} - w^- + w^+) = \\ &= Q - 4w \; . \end{aligned} \tag{5.25}$$

Hodnota Δ má stejný význam jako v předchozím případě prohazování indexů a říká nám, kolik se objeví nových spon mezi skupinami, když je $\Delta > 0$ nebo kolik spon mezi skupinami zmizí pro $\Delta < 0$. Pro výpočet hodnoty Δ vycházíme z rovnice (5.25).

$$\frac{(Q' \; - \; Q)}{4} \; = \; \Delta = w \tag{5.26}$$

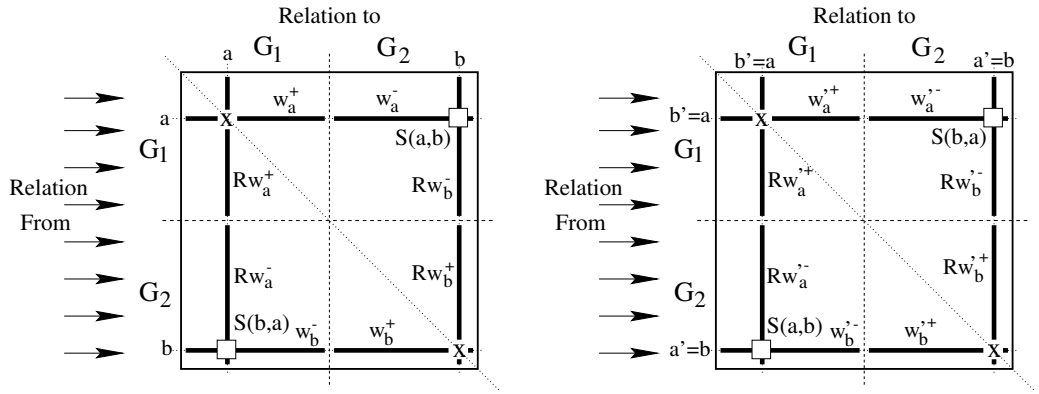
Po přesunu indexu z jedné skupiny do druhé dojde pochopitelně také ke změně kvalit všech ostatních indexů ve vektoru kvalit. Následující rovnice ukazují způsob aktualizace ostatních položek ve vektoru kvalit jednotlivých indexů $\vec{w}$:

$$\vec{w}'_x = \vec{w}_x + 2\mathbf{S}(x,a) \; ; \qquad x \in G_1; x \neq a \; . \tag{5.27}$$

$$\vec{w}'_x = \vec{w}_x - 2\mathbf{S}(x,a) \; ; \qquad x \in G_2; x \neq a \; . \tag{5.28}$$

5.7 Asymetrická (orientovaná) relace mezi indexy

Zkusme uvažovat relaci mezi indexy jako asymetrickou. U asymetrické relace již neplatí rovnost prvků nad a pod diagonálou matice $\mathbf{S}(a,b) \neq \mathbf{S}(b,a)$. Tato situace je zachycena na obrázku obr. 5.10(a). Jak je ukázáno v tomto oddíle je výpočet možný i v případě asymetrické relace za cenu přibližně dvojnásobného počtu výpočetních operací.



(b) Vybrané indexy a a b (c) Zaměněné indexy a a b

Obrázek 5.10: Prohození indexů v tabulce **S** pro asymetrickou relaci.

Relaci uvažujme ve směru zleva nahoru. Řádky matice **S** určují počáteční prvek relace a sloupce konečný prvek relace. Naším cílem je minimalizovat počet relací mezi prvky z množiny G_1 k prvkům množiny G_2.

Části sloupců v matici **S** již nejsou složkami kvalit jednotlivých indexů. Při záměně dvou indexů a a b však dojde i k záměně sloupců v matici **S** a tyto též ovlivní globální kritérium Q. Proto musí být sloupce ve výpočtech uvažovány.

Ke kvalitě indexu w_k je nutno přidat tzv. inverzní kvalitu indexu. Rozdíl mezi kvalitou a inverzní kvalitou indexu $a \in G_1$ ukazuje následující rovnice:

$$\begin{aligned} w_k = w_k^+ - w_k^- &= \sum_{i\in G_1;\, i\neq k} \mathbf{S}(i,k) - \sum_{i\notin G_1;\, i\neq k} \mathbf{S}(i,k)\,; \quad k \in G_1 \\ Rw_k = Rw_k^+ - Rw_k^- &= \sum_{i\in G_1;\, i\neq k} \mathbf{S}(k,i) - \sum_{i\notin G_1;\, i\neq k} \mathbf{S}(k,i)\,; \quad k \in G_1\,. \end{aligned} \tag{5.29}$$

Situace po prohození indexů a a b je ilustrována na obr. 5.10(b). Nové hodnoty kvalit prohozených indexů jsou označeny čárkou viz. rovnice (5.30). U asymetrické relace je ještě potřeba vyhodnotit nové hodnoty inverzních kvalit.

Číselné hodnoty lze určit z následujících rovnic:

$$\begin{aligned} Rw_a'^+ &= Rw_b^- - \mathbf{S}(a,b) \quad ; \qquad Rw_a'^- = Rw_b^+ + \mathbf{S}(b,a)\,; \\ Rw_b'^+ &= Rw_a^- - \mathbf{S}(a,b) \quad ; \qquad Rw_b'^- = Rw_a^+ + \mathbf{S}(a,b)\,. \end{aligned} \tag{5.30}$$

Vyšetřujme změnu globálního optimalizačního kritéria Q. Zaveďme si pomocné kritérium Q^*, které obsahuje ohodnocení celé tabulky sousedností indexů s výjimkou sloupců a řádek, které se vztahují k prohazovaným indexům a a b. Tímto trikem provedeme rozdělení globálního kritéria na dvě části. První část je závislá na prohazovaných indexech a druhá, označená symbolem Q^*, závislá není.

$$Q = Q^* + w_a + w_b + Rw_a + Rw_b \quad \rightarrow \quad Q^* = Q \; - w_a - w_b - Rw_a - Rw_b \,. \tag{5.31}$$

Nyní proveďme prohození dvojice indexů. Nová kritéria si označme čárkou např. Q'. Za povšimnutí stojí, že Q^* a Q'^* jsou stejné, poněvadž změna indexů a a b se projeví pouze na hodnotách w_a^+, w_a^-, w_b^+, w_b^-, Rw_a^+, Rw_a^-, Rw_b^+, Rw_b^-. Situaci po prohození indexů zachycují následující rovnice:

$$\begin{aligned} Q' = Q'^* + (w'_a + w'_b \;\; + \;\; Rw'_a + Rw'_b) \;\rightarrow & \\ Q'^* \;\; = \;\; Q' - w'_a - w'_b - Rw'_a - Rw'_b \,; & \qquad (5.32) \\ Q'^* \;\; = \;\; Q' + w_b + w_a + Rw_b + Rw_a + 4\,(\mathbf{S}(a,b) + \mathbf{S}(b,a)) & \qquad (5.33) \end{aligned}$$

Více než absolutní hodnota globálního kritéria nás samozřejmě zajímá změna k níž dojde po prohození indexů. Pro výpočet změny optimalizačního kritéria stačí vyhodnotit rozdíl rovnic (5.33) a (5.31):

$$\begin{aligned} Q'^* - Q^* \;\; = \;\; & (Q' - w_a - w_b - Rw_a - Rw_b) - \\ & (Q \;\; + w_a + w_b + Rw_a + Rw_b + 4\,(\mathbf{S}(a,b) + \mathbf{S}(b,a)))\,; \quad (5.34) \\ Q \;\; - \;\; Q' \;\; = \;\; & -\,2\,(w_a + w_b + Rw_a + Rw_b) - 4\,(\mathbf{S}(a,b) + \mathbf{S}(b,a))\,. \quad (5.35) \end{aligned}$$

Pro přímý výpočet hodnoty Δ_A je možno vyjít z rovnice (5.35), čímž dostaneme rovnici (5.36). Je-li symetrická relace počítána jako symetrická, pak musí platit: $2\,\Delta_A = \Delta$.

$$\frac{(Q' \;\; - \;\; Q)}{2} \;\; = \;\; \Delta_A = w_a + w_b + Rw_a + Rw_b - 2\,(\mathbf{S}(a,b) + \mathbf{S}(b,a))\,. \tag{5.36}$$

Výsledek výpočtu může čtenáře překvapit. Pro asymetrickou relaci musí platit: $2\,\Delta_A = \Delta$. Matici $\mathbf{S}$ lze převést na matici symetrickou $Sym\_\mathbf{S}(x,y) = \mathbf{S}(x,y) + \mathbf{S}(y,x)$ s dvojnásobným počtem spon. Po symetrizaci platí: $Sym\_w_a = w_a + Rw_a; Sym\_w_b = w_b + Rw_b; Sym\_\mathbf{S}(a,b) = \mathbf{S}(b,a) + \mathbf{S}(a,b)$.

$$\Delta_{Sym} = Sym\_w_a + Sym\_w_b - 2\,Sym\_\mathbf{S}(a,b)\,. \tag{5.37}$$

Protože Δ_{Sym} je rovno Δ_A je počítání s asymetrickou relací naprosto zbytečné!

5.8 Popis přesunů indexů pomocí operátorů

Pro elegantní a přehledný popis celého algoritmu lze s výhodou použít symbolických operátorů[6]. V našem případě operátor symbolicky označuje nějakou operaci s polem indexů např. přesun indexu z jedné skupiny do druhé. Operátor přesunu indexu lze považovat za základní a všechny další operátory vytvořit několikanásobnou aplikací základního operátoru. Takto je možno vytvořit zajímavý systém pro manipulaci s indexy.

5.8.1 Operátor prohození indexů

Máme definován operátor prohození indexů mezi dvěma skupinami. Vícenásobná aplikace tohoto operátoru nám umožňuje přeházet indexy mezi skupinami za současného zachování kardinality obou skupin. Počet aplikací operátoru prohození je možno též využít k měření vzdáleností mezi jednotlivými konfiguracemi indexů.

Uvažujme situaci prohození dvou indexů. Pro prohození indexů je nutno mít dvě disjunktní skupiny G_1 a G_2. Dále je nutné znát pozice prohazovaných indexů $a \in G_1$, $b \in G_2$ a je žádoucí mít k dispozici tabulku sousedností indexů $\mathbf{S}$ pro popis relací mezi indexy.

Výpočet ceny za prohození indexů lze symbolicky zapsat například takto:

$$C = Cost(G_1; G_2; a; b; \mathbf{S}). \tag{5.38}$$

Následuje symbolický zápis operace prohození dvou indexů:

$$\{G_{1new}; G_{2new}; \mathbf{S}_{new}\} = Swap(G_1; G_2; a; b; \mathbf{S}). \tag{5.39}$$

Pro urychlení výpočtu je vhodné do operátorů přidat vektor kvalit jednotlivých indexů $\vec{w}$. Pak dojde k následující modifikaci operátorů:

$$C = Cost(G_1; G_2; a; b; \mathbf{S}; \vec{w}). \tag{5.40}$$

$$\{G_{1new}; G_{2new}; \mathbf{S}_{new}; \vec{w}_{new}\} = Swap(G_1; G_2; a; b; \mathbf{S}; \vec{w}). \tag{5.41}$$

5.8.2 Operátor přesunu indexu

Také operátor přesunu indexu z jedné skupiny do druhé lze popsat symbolickým způsobem. Při přesunu indexu x ze skupiny G_2 do skupiny G_1 dojde k následujícím změnám $G_{1new} = G_1 + \{x\}$; $G_{2new} = G_2 - \{x\}$.

Symbolicky lze popsat operátor přesunu *Move* tímto způsobem:

[6] Operátor představuje obecně funkci pro transformaci dat. Funkce (operátory) mohou být vzájemně kombinovány za účelem vytvoření složitějších operátorů.

$$\{G_{1new}; G_{2new}; \mathbf{S_{new}}\} = Move(G_1; G_2; x; \mathbf{S}). \tag{5.42}$$

Další rovnice ukazuje možnou modifikaci operátoru $Move$, do které byla zahrnuta navíc aktualizace vektoru kvalit indexů:

$$\{G_{1new}; G_{2new}; \mathbf{S_{new}}; \vec{w}_{new}\} = Move(G_1; G_2; x; \mathbf{S}; \vec{w}). \tag{5.43}$$

Tento dodatečný operátor umožňuje volně měnit kardinalitu obou skupin s možností udržování aktuální hodnoty globálního kritéria a vektoru kvalit jednotlivých indexů. Operátor je vhodný pro případy, kde je zapotřebí změnit kardinality skupin vzájemným přesouváním indexů.

5.8.3 Skládání operátorů

Lze dokázat, že dvojím použitím operátoru přesunu (jeden index ze skupiny G_1 je zařazen do G_2 a jiný index z G_2 přejde do G_1) dostaneme operátor prohození indexů. Jednodušší operátor přesunu je obecnější a je možno z něj ostatní složitější operátory pro prohazování indexů sestavit.

$$\begin{aligned} Swap(G_1; G_2; a; b; \mathbf{S}) :- \quad & Move \quad (G_1; G_2; b; \mathbf{S}), \\ & Move \quad (G_2; G_1; a; \mathbf{S}). \end{aligned} \tag{5.44}$$

Naskýtá se otázka, proč tedy nepoužít pouze operátor přesunu, který je jednodušší? V případě optimalizace rozdělení indexů do dvou skupin se striktním požadavkem na zachování kardinality jednotlivých skupin se použití předchozího operátoru prohození jeví jako bezpečnější. Není potřeba hlídat explicitně počet prvků, protože zůstane po aplikaci operátoru prohození nezměněn. Navíc při stále stejné velikosti skupin je možno o něco více optimalizovat algoritmus prohazování indexů než v případě, kdybychom museli uvažovat změnu kardinality jednotlivých skupin. Vzhledem k menšímu počtu výpočetních operací může dojít ke zkrácení doby výpočtu. Další nevýhodou operátoru přesunu je praktická nemožnost jeho použití v nižších bitových rovinách.

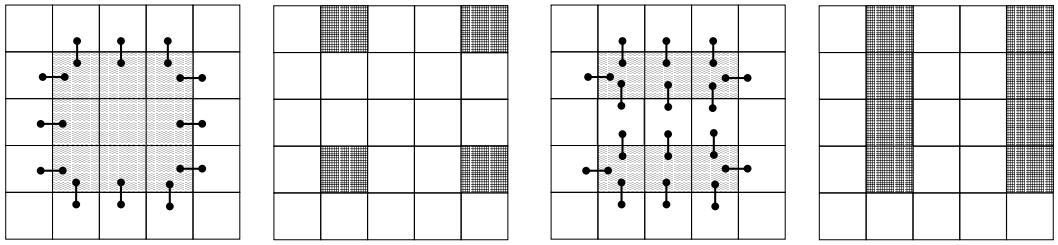
Symbolický zápis ukazuje na přímočarý způsob implementace popisovaného algoritmu do obecného programovacího jazyka. Jednotlivé operátory mohou představovat například procedury.

5.9 Závislost mezi počtem spon a počtem reziduí

V předchozím textu bylo naznačeno, že po fázi předzpracování následuje námi vytvořená komprimační technika založená na binárním prediktoru prof. Schlesingera, jejíž detailní popis je obsažen v kapitole 4. I když tvrdíme, že navrhovanou

fázi předzpracování lze použít pro libovolný komprimační program, původně byla vytvořena s cílem zmenšení počtu reziduí námi navrhovaného komprimačního algoritmu. V této sekci se věnujme způsobu, jakým počet reziduí souvisí s počtem spon v jedné bitové rovině.

5.9.1 Ukázka závislosti na konkrétním příkladu



Obrázek 5.11: Závislost počtu spon a reziduí.

Pro jednoduchost předpokládejme použití původního Schlesingerova kouteč-kového algoritmu. Pro měření kvality rozdělení indexů na dvě oblasti použijme pro ukázku pouze spony ve vodorovném a svislém směru podle obr. 5.3(b). V předchozím textu bylo ukázáno, že jakýkoliv jiný prediktor binárního typu s větší sondou (jedná se o všechny prediktory, které byly popsány v kapitole 4), musí za podmínky optimálního nastavení dávat lepší nebo alespoň stejné výsledky. Pokud bychom opominuli podmínku optimálního nastavení prediktoru, tak o počtu reziduí nelze nic tvrdit.

Na složeném obrázku obr. 5.11 jsou zobrazeny dvě dvojice obrázků data-rezidua. V levé polovině je umístěn jeden objekt a v pravé polovině se nacházejí objekty dva. V datové části jsou vyznačeny spony. Je důležité si povšimnout, že dva objekty mají delší hranici a produkují větší množství spon při srovnání s jediným objektem za podmínky srovnatelného součtu velikostí obou objektů na jedné straně s velikostí jediného objektu na straně druhé.

Podívejme se na předchozí porovnání počtu reziduí ještě trochu jinak. Minimální množství reziduí pro jeden izolovaný objekt, který se nedotýká hranice obrázku je čtyři. Takový objekt musí mít obdélníkový tvar. Pro dva objekty je minimální počet reziduí dvojnásobný. Prakticky může množství spon dvou malých objektů vyrovnat jen členitý větší objekt. Dále závisí počet reziduí na členitosti hranice objektu. Na členitější hranici produkující větší počet reziduí leží větší počet spon. *Proto lze přibližně tvrdit, že spony měří délku hranice mezi černými a bílými oblastmi v binárním obrázku.*

Tvrzení o délce hranice mezi dvěma skupinami platí i pro nižší bitové roviny. Každá bitová rovina z indexové funkce má totiž stejné vlastnosti jako binární obraz.

5.9.2 Vzájemná závislost izolovaných bitových rovin na kompresní poměr

V našem prediktoru nezávisí výsledná délka komprimovaných dat pouze na součtu všech reziduí přes všechny bitové roviny. Závislost mezi počtem reziduí v dané bitové rovině a dosažitelným kompresním poměrem není lineární. Jedna nelinearita je v kompresní metodě již implicitně obsažena. Jedná se o zamítnutí komprese při tzv. záporném kompresním poměru, viz též oddíl 3.1.2. Funkci závislosti komprimačního poměru na počtu reziduí v jedné bitové rovině můžeme při hrubším přiblížení považovat za monotónní. Kompresní poměr při snižování počtu reziduí narůstá rychleji než lineárně. Jako nejvýhodnější se proto jeví mít několik málo bitových rovin s velmi malým počtem reziduí a zbylý šum přesunout do spodních bitových rovin. Z uvedeného důvodu je prováděna nejprve optimalizace v nejvyšší bitové rovině a v nižších bitových rovinách je optimalizováno až to, co k optimalizaci zbyde.

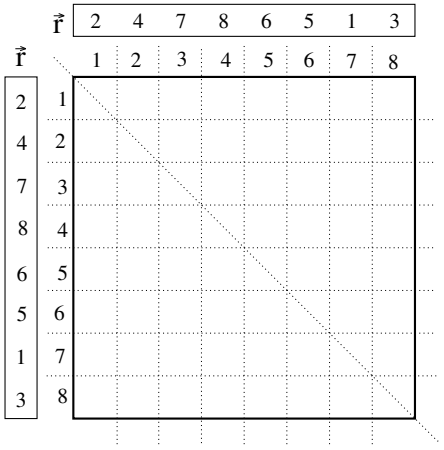
Další skutečností, která zatím nebyla brána v úvahu, je vzájemný vliv jednotlivých bitových rovin na 3-DOF prediktor, který využívá k predikci i data z vyšší bitové roviny. Tuto informaci by bylo vhodné nějakým způsobem zakomponovat do globálního optimalizačního kritéria. Výše uvedený návrh zatím nebyl experimentálně ověřen, ale lze očekávat mírný nárůst kompresního poměru za cenu vyšší složitosti optimalizačního algoritmu.

5.10 Efektivita výpočtu optimalizace

Zabývejme se způsoby zvýšení výpočetní efektivity navrhovaného algoritmu pro přeuspořádání palety. Obecně je možno šikovným návrhem algoritmu řádově snížit výpočetní náročnost na paměť a čas procesoru.

V navrhovaném algoritmu je mnohokrát použita operace prohození dvou indexů. Pro "naivní" provedení operace prohození by bylo možno proházet v tabulce sousedností indexů $\mathbf{S}$ všechny prvky mající nějaký vztah k prohazovaným indexům. Takovýchto prvků je $4n - 4$, kde n je počet všech indexů. Je zřejmé, že "naivní" postup jistě také vede k cíli, ale navržený algoritmus bude velmi pomalý vzhledem k nutnosti neustálé manipulace s velkým množstvím dat.

Ukažme na efektivní metodu řešení nastíněného problému. Nazvěme si původní tabulku sousedností indexů symbolem $\mathbf{S}^{\#}$ a symbol $\mathbf{S}$ ponechme pro aktuální modifikaci tabulky. Samozřejmě, že na počátku optimalizace po provedení analýzy četností musí platit $\mathbf{S} = \mathbf{S}^{\#}$. K datům z matice $\mathbf{S}$ nebudeme nyní přistupovat přímo, ale přes reindexační vektor. Zaveďme si reindexační vektor $\vec{r}$, přes který budou prováděny všechny přístupy k tabulce $\mathbf{S}$. Celá metoda je graficky znázorněna na obr. 5.12. Aktuální položky tabulky lze při použití reindexačního vektoru $\vec{r}$ již velmi snadno zjistit následujícím postupem:



Obrázek 5.12: Použití vektoru pro přeindexování.

$$\mathbf{S}(i,j) = \mathbf{S}^{\#}(\vec{r}(i), \vec{r}(j)). \tag{5.45}$$

Přístup k jednotlivým položkám tabulky **S** se sice stane o něco málo složitějším, ale na druhou stranu dojde k výraznému zjednodušení operace prohození dvou indexů. K prohození dvou indexů postačí pouhé prohození dvou položek v reindexačním vektoru $\vec{r}$.

Po dokončení celé optimalizační fáze je potřeba přerovnat všechna data v indexové funkci tak, jak bylo určeno výpočtem. Vektor $\vec{r}$ se chová jako funkce, která k nové hodnotě indexu určí hodnotu starou. Pro správné proházení indexů je potřeba k vektoru (funkci) $\vec{r}$ najít inverzní funkci. Výpočet inverzního vektoru je poměrně jednoduchý a lze jej udělat za n kroků. Tabulka **S** může být v tomto bodě zapomenuta, poněvadž již není více potřebná.

5.11 Experimenty

Podobně jako v předchozím případě komprimačního algoritmu pro šedotónové obrázky bychom chtěli demonstrovat na konkrétních datech vliv navrhovaného setřídění indexové funkce na účinnost komprese.

Pro kvantitativní měření kompresního poměru je opět použita *účinnost komprese*, kterou lze vypočítat podle rovnice (3.2). Dále je potřeba nějakým způsobem ohodnotit zvýšení kompresního poměru způsobené novým setříděním indexové funkce. Speciálně pro tento účel byla zavedena nová veličina kvantifikující zvýšení kompresního poměru a byla symbolicky nazvána *gain* (podle anglického slova gain=zvýšení, zesílení).

Veličinu *gain* vypočteme z následující rovnice:

$$Gain = \frac{\text{total output bytes (no optimization)} - \text{total output bytes (optimal)}}{\text{total output bytes (optimal)}} \cdot 100\%. \tag{5.46}$$

Gain nepředstavuje zesílení v pravém slova smyslu. Je definován následujícím způsobem. Je-li délka komprimovaných dat před a po optimalizaci stejná, pak $Gain = 0\%$. Pokud dojde k prodloužení délky komprimovaných dat, dosahuje *Gain* záporných hodnot. Při zmenšení délky komprimovaných dat na polovinu platí $Gain = 100\%$.

Jednotlivé proměnné v rovnici mají intuitivní názvy. Poznamenejme, že proměnná[7] *total output bytes (no optimization)* obsahuje celkový počet bajtů zkomprimované původní indexové funkce a v proměnné *total output bytes (optimal)* je uložena délka indexové funkce, která byla nejprve optimalizována a poté zkomprimována.

Pro testování bylo vybráno šest obrázků s různými vlastnostmi. Obrázek Peppers (červené a zelené papriky), viz obr. 5.13, spolu s barevnou Lenou přísluší do standardní testovací sady. Obrázky Garfield (známý kocour z kreslených seriálů) a Lynne (slečna stojící před letadlem) byly nalezeny na internetu. Obrázek Tartan (texturovaná tapeta) byl převzat ze standardní instalace systému Windows. A dále obrázek Descent - obr. 5.16 (sejmutá obrazovka ze hry Descent) byl vytvořen speciálně pro testovací účely.

Použité testovací obrázky by bylo možno rozdělit do čtyř základních kategorií.

- Obrázky vzniklé kvantováním a digitalizací reálné scény: Peppers a Lynne.
- Synteticky vytvořené obrázky různých scén: Descent.
- Ručně kreslené obrázky: Garfield.
- Synteticky vytvořené textury: Tartan.

V tab. 5.4 jsou shrnuty vlastnosti jednotlivých testovacích obrázků a dále jsou zde porovnány dosažené délky zkomprimovaných dat pro různé počáteční odhady rozdělení indexů do dvou skupin. Pro více informací o problematice počátečního odhadu viz oddíl 5.5.2. V prvním sloupci tabulky tab. 5.4 je zapsáno jméno obrázku. Druhý sloupec obsahuje informaci o velikosti a typu obrazových dat. První dvě čísla udávají rozměr obrázku a poslední číslo nese informaci o maximálním počtu indexů v indexové funkci. Čísla jsou zapsána ve formátu řádky × sloupce × počet bitových rovin na pixel. Ve třetím sloupci je obsažena délka dat nezkomprimovaných (jedná se o součet délek celé matice indexů a palety), která jsou uložena

[7] Pro zachování konzistence s anglickými texty článků jsou ponechány původní názvy proměnných beze změny. V české verzi jsou však doplněny překladem vysvětlujícím českému čtenáři jejich význam.

ve formátu BMP (Windows Bitmap). V posledních třech sloupcích se nacházejí velikosti komprimovaných dat pro tři různé počáteční odhady. Pro následnou komprimaci byla použita metoda s označením FH, jejíž popis je součástí kapitoly 4 této práce, viz též [HF97c, HF97a, HF99]. Sloupec s označením pouze FH znamená použití původního setřídění indexů bez jakéhokoli odhadu. Ve sloupci označeném návěštím Y+FH byla data počátečního odhadu setříděna podle intenzity. Intenzita byla vypočítána z palety. Pro poslední sloupec byl proveden počáteční odhad speciálně navržený a popsaný v této práci. Do délky zkomprimovaných dat je též započítána délka uložené palety.

Jméno obr.	Velikost $x \times y \times rovin$	Nekomprimovaný [byte]	FH [byte]	Y + FH [byte]	Opt. + FH [byte]
Descent	320×200×8	65078	24334	27115	21132
Garfield	640×480×4	153718	3323	-	2955
Lena, color	512×512×8	263222	235579	185275	154401
Lynne	320×200×8	65078	59016	43506	38769
Peppers	512×512×8	263222	206349	165315	129513
Tartan	256×256×4	32886	1596	-	1478

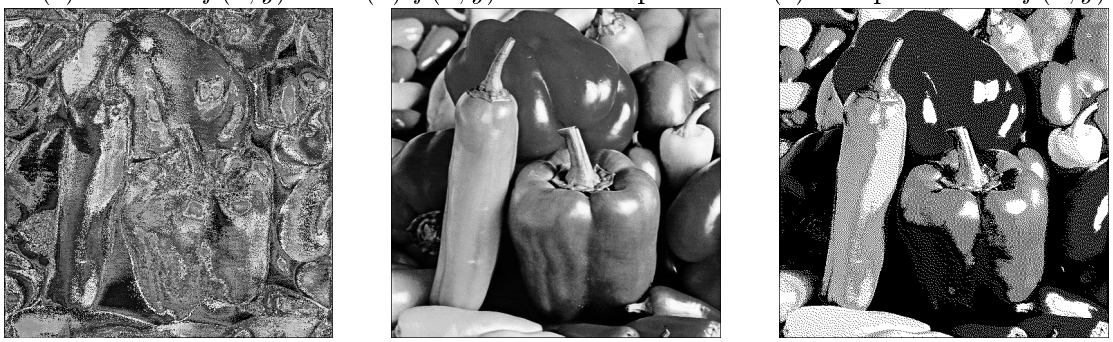
Tabulka 5.4: Vliv počátečního odhadu na účinnost komprese daných testovacích obrázků s paletou.

A nyní vizualizujme dosažené výsledky. V grafické podobě je mnohem snazší postřehnout změnu "hladkosti indexové funkce" v jednotlivých bitových rovinách. Nejprve je ukázán původní barevný obrázek Peppers s červenými a zelenými paprikami - obr. 5.13. Na dalším obrázku obr. 5.15 jsou zobrazena tři různá uspořádání indexové funkce. Protože uspořádání indexové funkce nesmí být navenek viditelné, je kompenzováno stejným přerovnáním palety. Pro vizualizaci indexové funkce je zapotřebí použit jiný způsob zobrazení dat. Ke každému indexu je přiřazen jas odpovídající pouze jeho pozici. Tím dostaneme šedotónový obrázek s počtem odstínů šedi, který je roven počtu indexů. Celou výše popsanou operaci nazýváme *zobrazení jako intenzitní obraz.*

Na obr. 5.14(a) je zobrazena původní indexová funkce jako intenzitní obraz. Po setřídění jednotlivých indexů podle jim příslušející intenzity Y, která byla vypočítána z palety [Fry93] podle vzorce $Y = R + G + B$, a následném ořezání palety vznikne indexová funkce velmi blízká šedotónovému obrázku, viz obr. 5.14(b). Obrázek obr. 5.14(c) zachycuje indexovou funkci po provedení její plné optimalizace navrhovaným algoritmem, která je zobrazena jako intenzitní obraz. Matice obrázků (obr. 5.15) obsahuje všech osm bitových rovin pro všechny tři varianty indexové funkce z obrázku obr. 5.14(a)(b)(c) tedy celkem 24 obrázků. V každém sloupci je umístěna sada bitových rovin pro jedno uspořádání indexů. Pohled na jednotlivé bitové roviny nabízí intuitivní vhled do činnosti algoritmu pro přerovnání indexů.

Obrázek 5.13: Původní pseudobarevný obrázek papriky-Peppers.

(a) Původní $f(x, y)$ (b) $f(x, y)$ Setříděna podle Y (c) Po optimalizaci $f(x, y)$



Obrázek 5.14: Porovnání tří modifikací indexové funkce palety $f(x, y)$ zobrazené jako intenzitní obraz.

(a) Původní $f(x, y)$ (b) $f(x, y)$ setříděna podle Y (c) Optimalizovaná $f(x, y)$


Bitová rovina #8


Bitová rovina #8


Bitová rovina #8


Bitová rovina #7

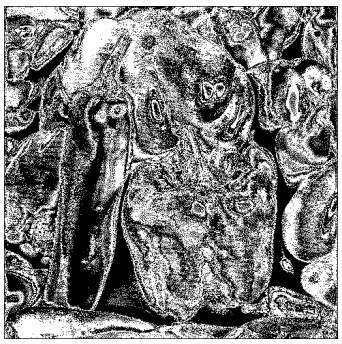

Bitová rovina #7

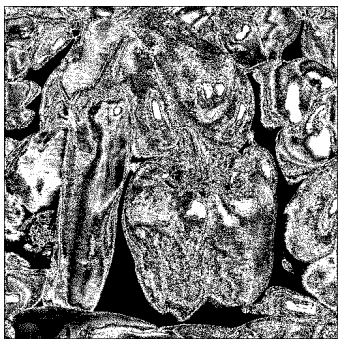

Bitová rovina #7


Bitová rovina #6


Bitová rovina #6


Bitová rovina #6


Bitová rovina #5


Bitová rovina #5


Bitová rovina #5

Obrázek 5.15: Jednotlivé bitové roviny ze tří modifikací indexové funkce palety $f(x, y)$.

Stejné chování algoritmu pro přerovnání indexů v paletě lze demonstrovat též na jiných obrázcích např. Descent obr. 5.16.

Nyní věnujme pozornost nejvyšší bitové rovině #8 (první řádek obr. 5.15). Ve variantě (a) je i v této rovině obsažena vysoká míra neuspořádanosti (lze měřit podle entropie prvního řádu). Na první pohled lze poznat, že varianty (b) a (c) vypadají o hodně lépe. Nyní se podívejme pozorněji na nižší bitové roviny. Již v bitové rovině #6 se u varianty (b) uplatňuje pozorovatelné zvýšení entropie prvního řádu. U varianty (c) vypadají bitové roviny $\#8 \div \#4$ mnohem uspořádaněji. Zde ukázané chování algoritmu pro setřídění palety je možno demonstrovat na většině pseudobarevných obrázků s paletou. Připomeňme, že všechny tři modifikace indexové funkce vytvářejí totožný barevný obrázek.

Jméno obr.	*CE* FH	*Gain* FH	*CE* Calic	*Gain* Calic	*CE* PNG	*Gain* PNG
Descent	67,6%	13,6%	50,8%	13.1%	66,5%	0.51%
Garfield	98,1%	13,4%	*neznám*	*neznám*	97,2%	-2.0%
Lena, color	41,3%	52,2%	33,4%	39,1%	32,4%	0%
Lynne	40,4%	52,2%	30,6%	15,7%	24,5%	0%
Pepper	50,1%	59,3%	41,9%	49,0%	45,3%	0%
Tartan	95,0%	11,0%	*neznám*	*neznám*	92,6%	3.2%

Tabulka 5.5: Vliv modifikace indexové funkce na FH a jiné kompresní metody.

Provedené experimenty ukazují, že navrhované přerovnání palety zvýší účinnost i dalších komprimačních algoritmů. Účinnost komprese CE byla určena jako poměr podle (3.2). Nejvýrazněji by se měl zvýšit kompresní poměr u algoritmů zpracovávajících bitovou mapu po jednotlivých bitových rovinách. Bohužel zatím nemáme dostupnou implementaci takového cizího algoritmu. Srovnávací testy byly provedeny s algoritmy Calic [WM97] a PNG (Portable Network Graphics) [Tea96]. Implementace obou algoritmů byly staženy z internetu. Výsledky provedených testů jsou shrnuty v tab. 5.5. Komprimační algoritmus Calic podporuje komprimaci pouze 256 úrovňových bitmapových obrázků, a proto je obsah některých položek označen slovem *neznám*. Komprimační algoritmus PNG již v sobě obsahuje velmi jednoduchou metodu pro přerovnání indexů. Předpokládám, že tato technika mohla do jisté míry narušit optimální přerovnání indexů, což se projevilo na výrazně nižší hodnotě koeficientu zlepšení kompresního poměru *Gain* určeného z rovnice (5.46).

Nejvyšších hodnot zlepšení kompresního poměru *Gain* bylo dosaženo pro naší metodu *FH* [HF97c, HF97a]. V rámci objektivity je třeba poznamenat, že obě metody byly na sebe optimálně naladěny, viz oddíl 5.4.2 a nové naladění optimalizačního algoritmu pro jiné komprimační techniky již nebylo provedeno.


Původní obrázek


Indexy setříděny podle Y


Optimální kód

Obrázek 5.16: Ukázka 3 modifikací indexové funkce pro pseudobarevný syntetický obrázek Descent.

Analýza časové náročnosti algoritmu

Navrhovaný algoritmus pro přerovnání palety je poměrně rychlý. Na procesoru Intel Pentium 120 MHz trvá zpracování pseudobarevného obrázku 512×512×8bit kolem 2 sekund.

Nyní se podívejme podrobněji na doby trvání jednotlivých částí algoritmu. Do uvedené doby se promítají tři základní složky. Jedná se o:

A. Výpočet matice sousedností indexů.

B. Hlavní výpočet optimalizované indexové funkce.

C. Přemapování dat indexové funkce a palety podle vypočteného vektoru $\vec{r}$.

Doby trvání jednotlivých částí algoritmu jsou pro dva konkrétní obrázky Lynne a Park přehledně znázorněny v tabulce 5.6. Doba nutná pro provedení počátečního odhadu je zanedbatelná, a proto nebyla zahrnuta do tabulky. Kroky (A) a (C) jsou závislé na velikosti obrazových dat. Krok (B) závisí pouze na velikosti palety (počtu barev) a nikoliv na rozměrech obrázku. V některých případech je možno provádět přemapování indexové funkce přímo při zápisu dat na výstupní zařízení a tím výrazně zkrátit třetí fázi zpracování.

Jméno Obr.	A Výpočet **S**	B Optimalizace	C Přemapování	A+B+C Celkem $\sum$
Lynne	0.22 s	0.88 s	0.16 s	1.26 s
Park	0.55 s	1.04 s	0.55 s	2.14 s

Tabulka 5.6: Analýza časů jednotlivých částí algoritmu pro přemapování palety.

Poznamenejme, že navrhovaný algoritmus přerovnání indexů nebyl nijak časově optimalizován a pro časově kritické aplikace by bylo možno jej zapsat efektivněji.

Kapitola 6

Rychlé zpracování n-rozměrných binárních obrázků

6.1 Původ navrhované techniky

Profesor Schlesinger na svých přednáškách hovořil o svých nových technikách rychlého zpracování binárních obrázků. Hlavní myšlenkou je vytvořit takovou transformaci obrazu, která umožní přímé zpracování komprimovaných dat v paměti počítače. Pro zpracování se nejlépe osvědčila koutečková reprezentace. Původní koutečková metoda prof. Schlesingera je popsána v oddíle 3.6.4.

Celý obraz lze transformovat na seznam koutečků. Se seznamem lze provádět většinu běžných transformací tak, jako by se jednalo o nekomprimovaná data. Jedná se o běžné operace AND, NOT, OR, XOR, posun o jeden pixel vlevo, posun o jeden pixel vpravo, výpočet skeletonu a morfologie.

Cílem tohoto pojednání je zobecnit koutečkovou transformaci obrazu pro složitější struktury než dvourozměrné binární obrázky.

6.2 Teoretické pozadí

Koutečková transformace je postavena na binárním prediktoru s pevným modelem dat. Jednoznačnost transformace binárních dat na rezidua a transformace reziduí na binární data byla ukázána v sekci 4.7. V jednorozměrném případě je prediktor dán rovnicí:

$$\hat{x_2} = e(x_1) = x_1 \tag{6.1}$$

Dvourozměrný koutečkový prediktor je zakreslen na obr. 3.8. Rovnice prediktoru má tvar:

$$\hat{x_4} = e(x_1, x_2, x_3) = x_1 \oplus x_2 \oplus x_3 \tag{6.2}$$

Takto je možno postupovat do dalších rozměrů. Vždy se jedná o n rozměrnou krychli o velikosti dvou buňek, jejíž jeden vrchol je predikován podle ostatních vrcholů. Funkce prediktoru je složena z nonekvivalencí.

6.3 Nové návrhy

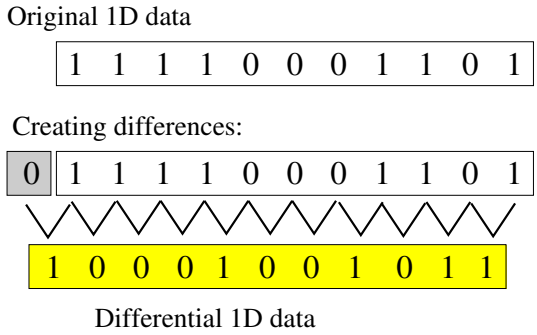
Jako nejnovější výsledek mého výzkumu se mi podařilo rozšířit Schlesingerův přístup pro n-rozměrná data. Téměř všechny závěry platné pro dvourozměrná data lze přenést do více rozměrů. Největší využití by mohlo být např. v oblasti tomografie, kde je prováděno třírozměrné modelování částí těla.

Hlavní a základní nápad na kterém je navržené technika postavena, spočívá v hierarchickém vytváření algoritmů pro jednotlivé dimenze. To znamená, že nejprve je potřeba vytvořit sadu funkcí pro 1D data. Na nich lze vystavět další sadu funkcí pro 2D data. A tak lze postupovat dále.

Jedním z důvodů, proč dosud většina úsilí o vyřešení uvedeného problému více dimenzí ztroskotala, je relativní složitost úlohy. Teprve při hierarchickém přístupu se celé řešení stává jednodušším a průhlednějším.

Tuto problematiku nemám bohužel zpracovánu do takové hloubky jako předchozí pojednání o komprimaci dat. Domnívám se však, že se jedná o zajímavý problém, jehož popis by neměl v mé práci chybět.

6.3.1 Jednorozměrný případ



Storing as a list of absolute values of differences:

X=[0, 4, 7, 9, 10]

Obrázek 6.1: Zpracování jednorozměrných dat.

V předchozím textu již bylo zmíněno, že celá koutečková technika je založena na manipulaci s diferencemi. Tvorba seznamu diferencí v jednorozměrném případě je zobrazena na obr. 6.1. Originální data jsou zapsána do rámečku umístěného

nahoře. Vzhledem ke skutečnosti, že diference je počítána ze dvou hodnot, je přidána zleva k datům fiktivní položka s nulovou hodnotou. Operace diference je v obrázku schématicky znázorněna dvěma čárami ve tvaru písmene 'V'. Diferenci lze počítat jako operaci nonekvivalence XOR. Dále je vhodné poznamenat, že pro úplný popis dat si postačí zapamatovat polohy všech míst, kde byla diference nenulová. U reálných se vyskytuje odlišnost, tedy hodnota 1, mnohem méně často, a proto se vyplatí všechny nenulové diference reprezentovat jejich seznamem. Každé místo s nenulovou diferencí si nazvěme *1D kouteček*.

Na jednotlivé diference je možno pohlížet i jiným způsobem. Každá diference představuje negaci celého proužku dat od místa svého výskytu až do konce.

Inverze 1D struktury

Obrázek 6.2: Výpočet inverze pro 1D data.

S jednorozměrnými daty lze též provádět binární operace. Nejjednodušší operací je inverze. Způsob výpočtu inverze je zobrazen na obr. 6.2. Jsou zde ukázány diference originálních a invertovaných dat. Jediná odlišnost nastává v místě první diference.

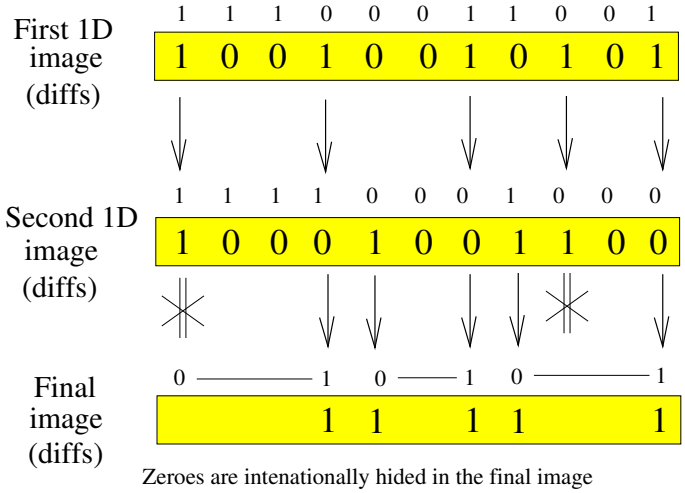
Z odlišnosti vyplývá velmi jednoduchý algoritmus pro výpočet negace z diferencí, které jsou reprezentovány jako seznam. Pokud je diference s číslem 1 přítomna, tak bude vyjmuta ze seznamu. V opačném případě je nutno diferenci s číslem 1 přidat do seznamu. Takový algoritmus může být zapsán např. následujícím způsobem:

$$\begin{aligned} if\ X[1] = 1 \quad then \quad & notX = X[2, ...n] \\ else \quad & notX = [1, X] \end{aligned} \tag{6.3}$$

X je původní seznam diferencí s počtem prvků n. Seznam $notX$ představuje jeho inverzi.

Výpočet XOR pro 1D strukturu

Výpočetně velmi snadným se jeví též výpočet funkce XOR. Postačí totiž jediným průchodem slít 1D koutečky z obou seznamů dohromady. V případě, pokud se vyskytne v obou seznamech na stejné pozici nějaký kouteček, jsou oba koutečky ignorovány a ve výsledném seznamu se nevyskytne na dané pozici žádný kouteček. Ilustrativně je činnost algoritmu vyobrazena na obr. 6.3. Místa, v nichž nedochází ke změně jsou nedůležitá a správná hodnota výsledných pixelů bude určena správně, poněvadž je jednoznačně dána pozicemi diferencí.

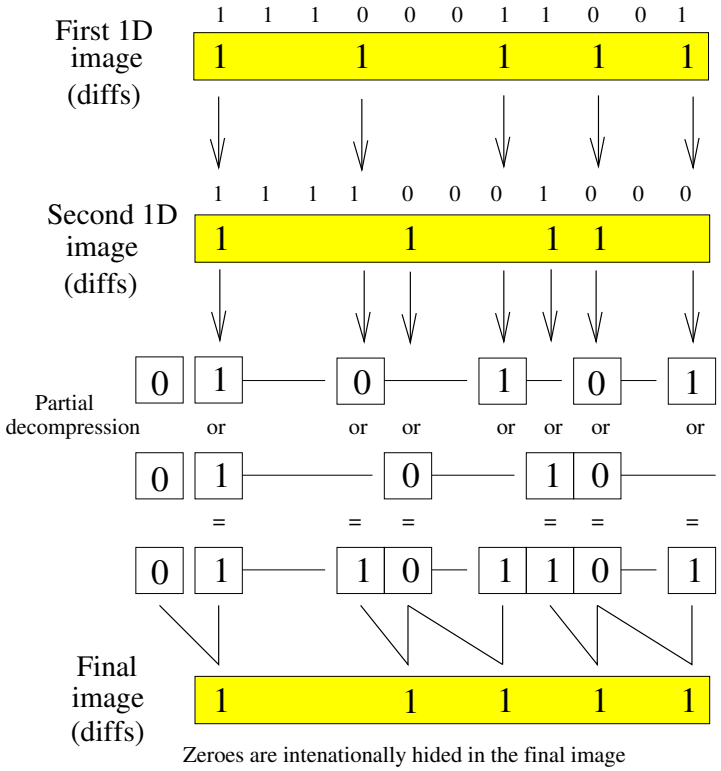


Obrázek 6.3: Výpočet funkce XOR pro 1D data.

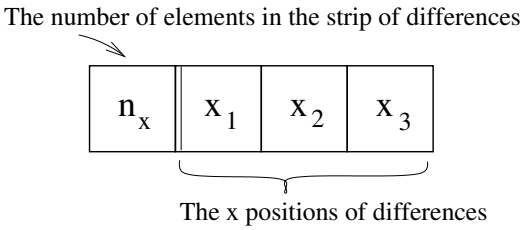
Výpočet funkcí AND a OR pro 1D strukturu

Výpočet funkce OR (popř AND) je již o něco komplikovanější. Platné je pouze předchozí tvrzení o tom, že v místech mezi koutečky je hodnota pixelů stejná, a tudíž se nemá smysl jí zabývat. Při výpočtu funkce OR musí být v každém koutečku rekonstruována původní hodnota jemu odpovídající buňky. Pro sudé koutečky se jedná o nulu a pro liché o jedničku. Hodnota je platná až do další změny, což je vyznačeno vodorovnou čárou na obr. 6.4. Ze dvou hodnot pod sebou lze vypočítat funkci OR a v případě změny ve výstupních datech, uložit kouteček do výstupního seznamu. Celý výpočet je demonstrován na obr. 6.4. Oblasti označené vodorovnou čarou nemusí být algoritmem uvažovány. Podobným způsobem lze vypočítat funkci AND a v podstatě jakoukoliv logickou funkci ze dvou proměnných pro celý seznam.

Poznamenejme, že většinu úkonů souvisejících s výpočtem logických funkcí AND, OR aj. lze vykonat specializovaným konečným automatem.



Obrázek 6.4: Výpočet funkce OR pro 1D data.

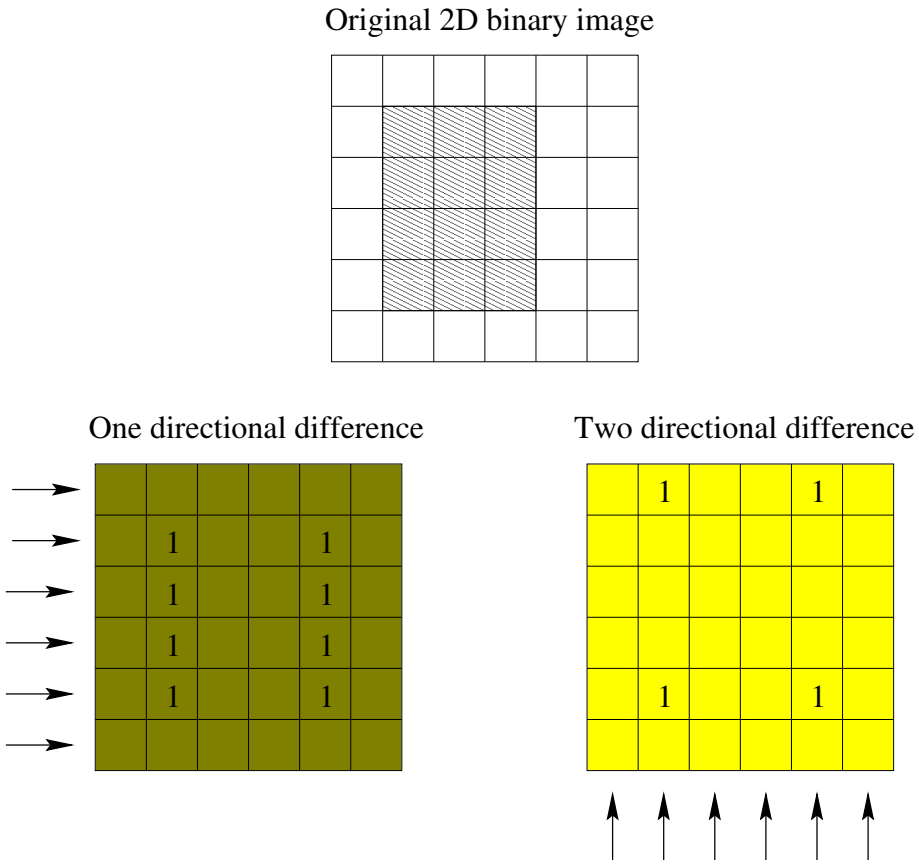


Obrázek 6.5: Návrh paměťové reprezentace uložených 1D diferencí.

Struktura uložení 1D dat v paměti

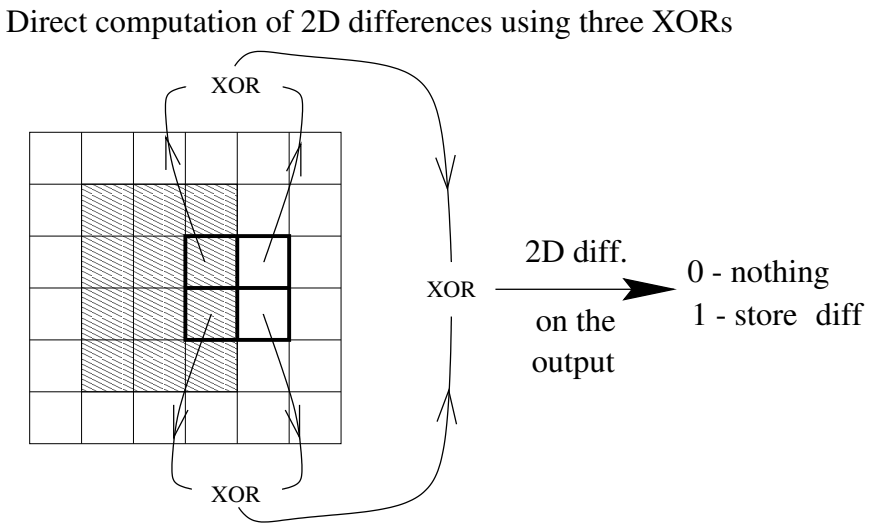
Na obr. 6.5 je ukázána navržená paměťová struktura pro jednorozměrná data. Jedná se o pole obsahující vzestupně uspořádané polohy jednotlivých 1D koutečků. První položka obsahuje počet diferencí.

6.3.2 Dvourozměrná data



Obrázek 6.6: Zpracování dvourozměrných dat.

Výpočet koutečků u dvourozměrných dat je velmi podobný výpočtu u jednorozměrných dat. Protože se jedná o dvourozměrnou strukturu, je nutno počítat diference ve dvou krocích. Nejprve je potřeba vyhodnotit diference v jednom rozměru a následně určit diference předchozích diferencí v rozměru druhém. Na pořadí, ve kterém jsou diference počítány, nezáleží a výsledek je tudíž pokaždé stejný. Celá situace je zobrazena na obr. 6.6. Diference vlastně představují koutečky z výstupu binárního koutečkového prediktoru. Proto lze obě diference vypočítat najednou z malé sondy s rozměry 2×2 pixelů jak je ukázáno na obr. 6.7.



Obrázek 6.7: Rychlý výpočet dvourozměrných diferencí.

Inverze 2D struktury

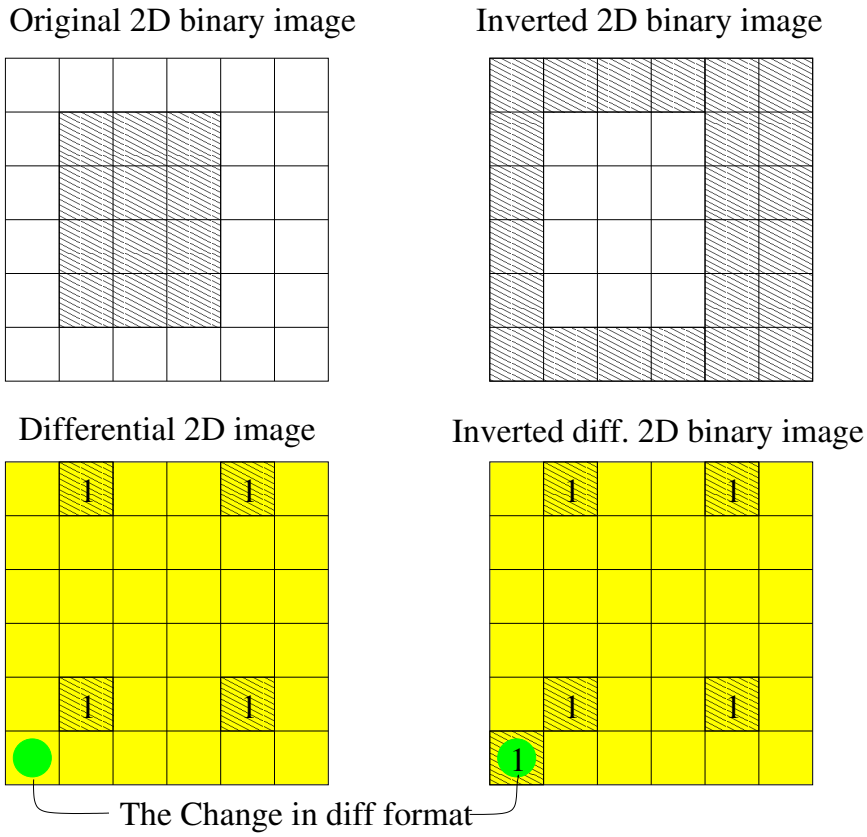
Při inverzi (operace NOT) dvourozměrné datové struktury lze využít stejného triku jako při inverzi struktury jednorozměrné. Celá situace je ukázána na obr. 6.8. Jsou zde graficky znázorněny dva obrázky, přičemž jeden představuje inverzi druhého. Pod nimi jsou zachyceny jejich diference. Jediná odlišnost dvou diferenčních obrázků se vyskytuje v levém dolním rohu. Odlišnosti lze využít pro výpočet inverze, který je dokonce nezávislý na počtu koutečků. Vyskytuje-li se v levém dolním rohu diference, pak je vyřazena ze seznamu 2D diferencí. V opačném případě je potřeba diferenci do seznamu jednoduše přidat. Při použití hierarchické struktury dat lze pracovat pouze s proužkem, který odpovídá nultému řádku.

Výpočet XOR pro 2D strukturu

Výpočet XOR (nonekvivalence) je též jednoduchý. Postačí slít všechny koutečky dohromady a dávat si pozor na koutečky, které se vyskytují v obou datových strukturách na stejném místě. V takovém případě je nutno oba koutečky ignorovat. U výpočtu funkce XOR ve 2D je vhodné demonstrovat hierarchický přístup. Funkci XOR je možno počítat po jednotlivých řádcích (1D strukturách) naprosto nezávisle na sobě.

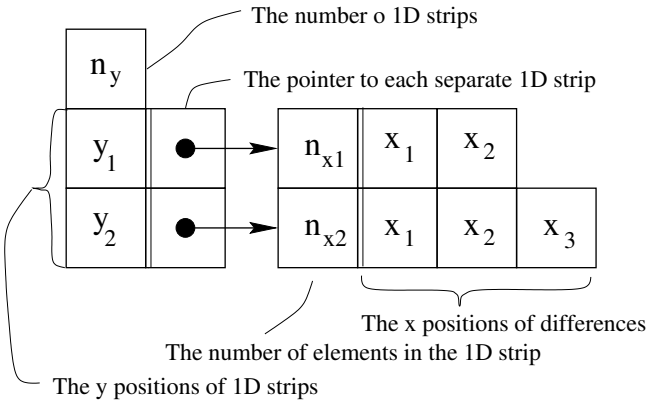
Výpočet funkcí AND a OR pro 2D strukturu

Pro výpočet funkce OR (AND) je již potřeba provádět dílčí dekompresi dvourozměrných struktur na jednorozměrné. Stále je možno ve velké míře využívat procedur z nižší dimenze.



Obrázek 6.8: Výpočet inverze pro 2D data.

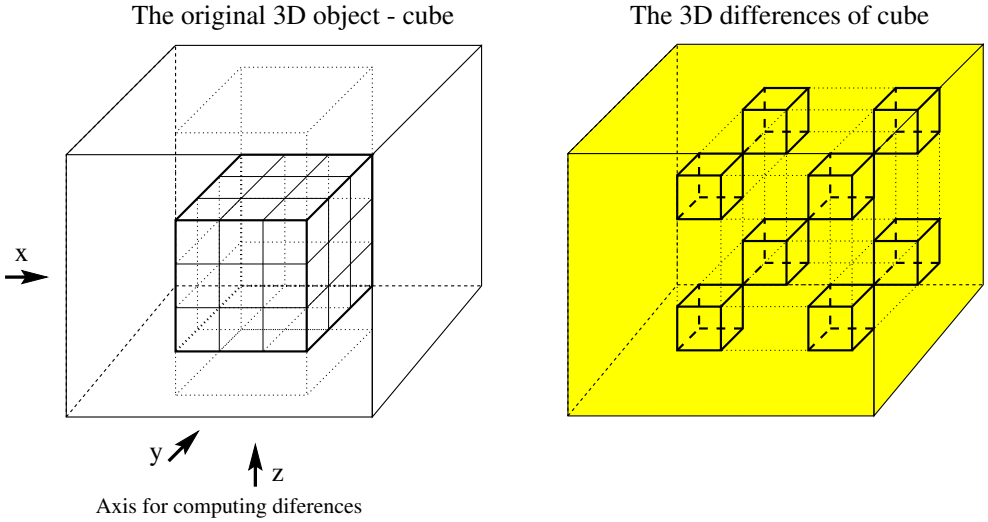
Struktura uložení 2D dat v paměti



Obrázek 6.9: Návrh paměťové reprezentace uložených 2D diferencí.

Na obr. 6.9 je ukázána navrhovaná struktura dvourozměrných dat v paměti počítače. Je zde zřetelné, že dvourozměrná data lze snadno rozložit na data jednorozměrná (viz obr. 6.5) a předávat je příslušným procedurám pro jednorozměrná data. Tento rys se stává mnohem důležitější u vícerozměrných dat.

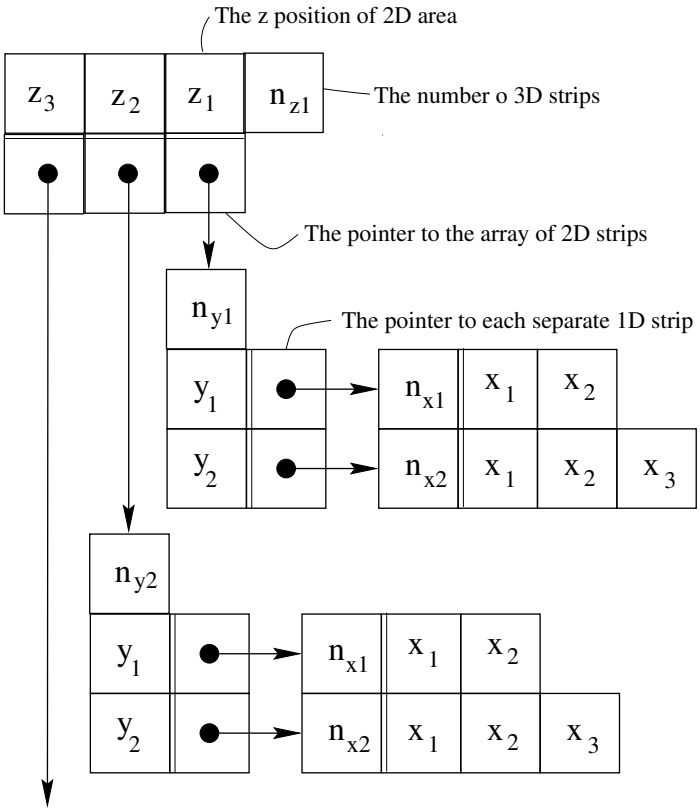
6.3.3 Třírozměrná data



Obrázek 6.10: Zpracování třírozměrných dat.

Všechny úvahy, které byly dosud učiněny, jsou platné i pro třírozměrná data. Pro výpočet 3D koutečků je nutno počítat diference binárních voxelů ve třech roz-

měrech. Situaci směrů diferencí vystihuje obr. 6.10. Na pořadí provádění diferencí nezáleží a vždy je získán stejný výsledek.



Obrázek 6.11: Návrh paměťové reprezentace uložených třírozměrných diferencí.

Na obr. 6.11 ukažme datovou strukturu určenou k uchování 3D koutečků. Pozorný čtenář si povšimne, že se skládá ze seznamu dvourozměrných struktur, ke kterým je navíc přidána pozice v ose z. Ke každé 2D struktuře je navíc přidána pozice v ose z.

6.4 Na co je navrhovaný přístup dobrý?

Již prof. Schlesinger na svých přednáškách hovořil o až 300 násobném zrychlení složitých výpočtů s binárními obrázky. Při tom je samozřejmě menší spotřeba paměti. Hlavním důvodem této skutečnosti je poměrně malý počet změn v běžných obrázcích.

Například při zvyšování rozlišení u scanování binární předlohy, která obsahuje pouze text, od určitého bodu již detailů výrazně nepřibývá. Zato paměťová náročnost roste se čtvercem rozlišení. Právě zde se nejlépe osvědčí práce s daty v komprimovaném tvaru.

U třírozměrných dat by byla situace bez použití navrhované techniky o řád horší. Se zvětšujícím se rozlišením roste paměťová náročnost se třetí mocninou. Obrovská krychle binárních voxelů nenachází zatím velkého uplatnění právě pro svoji paměťovou náročnost.

Vezměme si příklad objemových dat pocházejících dalším zpracováním dat z range-fingeru[1]. Není problém si vytvořit v pracovní paměti krychli o velikosti $65536 \times 65536 \times 65536$ pro interní popis modelu. Je-li prázdná, tak nezabírá téměř žádnou paměť. Nyní lze udělat měření jedné plochy tělesa. Naměřené a vypočtené body lze uložit do paměti ve zkomprimovaném tvaru. Při dalším měření lze naměřená data uložit do jiné struktury a pomocí logické funkce OR složit obě datové struktury dohromady.

Podobné využití by mohlo být u tomografu. Zde je také potřeba snímat, manipulovat a ukládat třírozměrná objemová data.

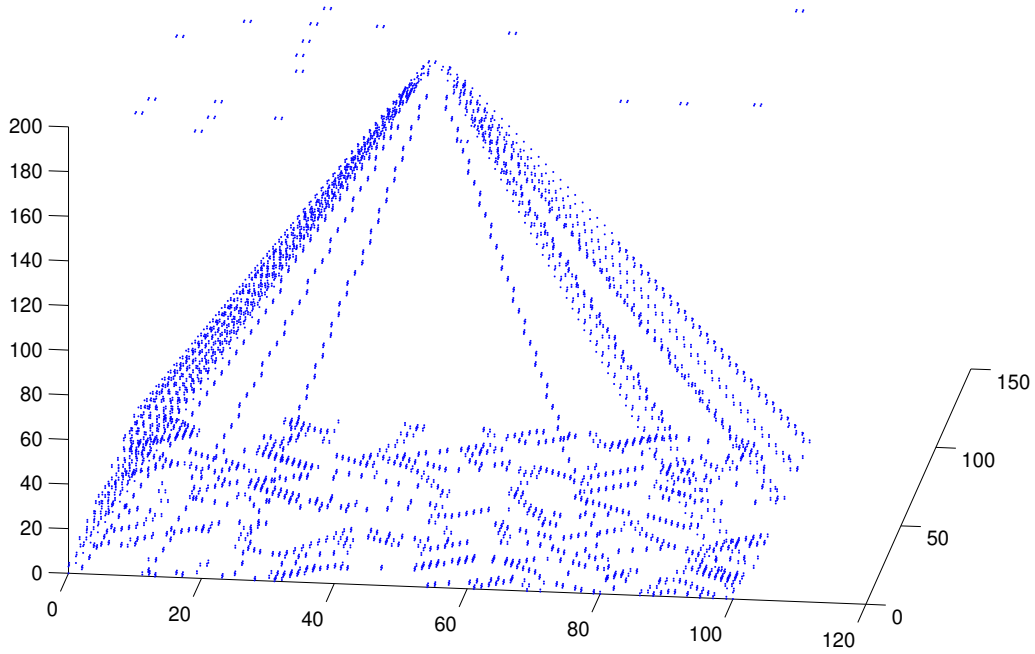
6.5 Experimenty

Tuto část své práce jsem chtěl též experimentálně vyzkoušet a demonstrovat funkčnost předložených tvrzení. Jednodušší koutečková knihovna pro 2D je popsána např. v [BHP+00].

Bohužel jsem narazil na nečekané překážky: velmi málo grafických formátů umí uchovávat 3D bitovou matici voxelů, neměl jsem k dispozici žádnou knihovnu pro načítání voxelových dat a moje knihovna pro 3D koutečky příliš často alokuje dynamickou paměť, což snižuje její výkonnost. Z tohoto důvodu neuvádím srovnávací rychlostní testy.

Přesto mohu na obr. 6.12 demonstrovat funkční verzi knihovny. Obrázek vzniknul fůzí několika set čtyřbokých jehlanů v koutečkové reprezentaci. Na obrázku jsou zachyceny pouze koutečky. Podobné těleso ve voxelové reprezentaci by obsahovalo 3600000 voxelů.

[1]Range-finder je zařízení určené ke snímání hloubkových map, tj. produkuje matici, jejíž prvky představují vzdálenosti od pozorovatele.



Obrázek 6.12: Koutečky geometrického tělesa vzniklého fůzí jehlanů.

Kapitola 7

Závěr

Testovací implementace je dostupná z mojí domovské www stránky:
http://cmp.felk.cvut.cz/~fojtik/.

7.1 Komprimace šedotónových obrázků

Navrhl jsem a realizoval nový komprimační algoritmus. Můj návrh byl inspirován přednáškami prof. Schlesingera. Mým původním záměrem bylo ověřit a prozkoumat možnosti kodéru, který je založen na čistě bitovém způsobu zpracování obrazových dat. Tento záměr se do velké míry zdařil.

Prediktor prof. Schlesingera se podařilo zobecnit a doplnit tak, aby bylo možno úspěšně komprimovat šedotónové obrázky. Zobecnění bylo matematicky popsáno funkcí prediktoru. V práci byl ukázán způsob, jak efektivně vybírat jednotlivé buňky pro prediktor. Pro potřeby komprimace byla vyvinuta metoda efektivního ukládání reziduí do komprimovaných dat.

Navíc byl navržen nový typ prediktoru s adaptivní velikostí sondy. Dále byl nalezen nový způsob výpočtu četnostní analýzy s poloviční náročností na kapacitu paměti.

Výsledky dosahované při kompresi dat jsou podle provedených experimentů poměrně dobré. Algoritmus je použitelný především pro archivaci obrazových dat. Nejlepších kompresních poměrů je dosaženo pro technické kresby a pro ručně kreslené barevné obrázky. Další výhodou popisovaného algoritmu je možnost jeho snadné realizace logickými obvody. S tím souvisí i možnost komprimovat videosignál v reálném čase. Navržený algoritmus není vhodný ke komprimaci obrázků, které vznikly ditheringem nebo které obsahují silný impulsní šum.

Výhody a nevýhody komprimace binárním prediktorem

Tato část mé práce byla publikována v následujících sbornících: [HF97b, HF97c, HF97a, Foj97] a v též v článku v časopise [HF99]. Stručné shrnutí hlavních vý-

hod a nevýhod navrhovaného algoritmu pro komprimaci šedotónových obrázků je uvedeno v následujícím výčtu.

Výhody:

A. Komprimační metoda je plně automatická. Uživatel nemusí nastavovat žádné parametry.

B. Vzhledem k použití pouze velmi jednoduchých operací SHL, XOR, ADD a výběru z tabulky může (závisí na implementaci) být komprimační algoritmus velmi rychlý.

C. Vzhledem k použití bitových operací mohou být podstatné části dokonce napsány ve strojovém kódu nebo realizovány přímo hardwarově.

D. Metoda dává uspokojivý kompresní poměr srovnatelný s dnešními nejlepšími bezeztrátovými komprimačními technikami.

E. Algoritmus neomezuje kompresní poměr: $\lim_{x\to\infty;y\to\infty;E\to 0} CE = 100\%$ Kompresní poměr CE byl definován rovnicí 3.2; x a y jsou rozměry monotónního obrázku s malou nezvětšující se ikonou uprostřed. Tato vlastnost nastává u horších kompresních algoritmů. Popisovaná vlastnost se nejlépe projeví při komprimaci zvětšující se černé plochy s bílým bodem.

F. Algoritmus lze snadno paralelizovat a tím dále urychlit.

G. Prediktor je poměrně snadno rozšiřitelný.

H. Při zvyšování rozlišení obrazu nedochází ke zhoršování kompresního poměru. Tato nepříjemná vlastnost se vyskytuje u některých typů pyramidní komprese třeba MLP, viz [Fry93].

Nevýhody navrhované metody jsou:

A. Navržená kompresní technika vyžaduje minimálně dva průchody obrazovými daty. Při použití Huffmanova kódu pro kódování reziduí je potřeba ještě dalšího třetího průchodu. Všechny průchody jsou plně sekvenční.

B. Algoritmus je optimalizován pouze na určitou třídu grafických dat. Jiné algoritmy mohou dosáhnout pro jiné třídy dat vyšší kompresní poměr.

C. Obrazová data jsou zpracovávána po jednotlivých bitových rovinách. Tato vlastnost může být v některých implementacích na překážku. Skládání bitových rovin dohromady totiž vyžaduje dostatečně velkou paměť pro celý obraz, která nemusí být na přijímací straně dostupná. Na druhé straně je možno paralelně komprimovat všech osm bitových rovin nezávisle na sobě.

D. Dekomprimační algoritmus je velmi citlivý na chyby v datech. Ale tato vlastnost je typická i pro jiné komprimační algoritmy.

7.2 Komprimace barevných paletových obrázků

Pro zvýšení rozsahu použitelnosti navrženého komprimačního algoritmu a pro rozšíření třídy komprimovaných obrázků jsem navrhl cestu, jak umožnit též komprimaci obrázků s paletou. Algoritmus předzpracuje data z paletových obrázků tak, že dojde k výraznému zvýšení kompresního poměru u předchozího komprimačního algoritmu. Celý algoritmus je unikátní zejména tím, že optimalizuje po bitových rovinách.

Výhody a nevýhody přerovnání palety

Stručné shrnutí hlavních výhod a nevýhod algoritmu, který přemapovává indexovou funkci (zaměňuje indexy) za účelem zvýšení její komprimovatelnosti.

Výhody:

A. Při kompresi přerovnané indexové funkce dochází ke zvýšení kompresního poměru.

B. Algoritmus přerovnání indexů lze použít i před jiné kompresní algoritmy.

C. Po přerovnání téměř nikdy nedochází ke zhoršení kompresního poměru indexové funkce.

D. Způsob přerovnání indexů lze modifikovat změnou relace sousednosti.

E. Algoritmus je poměrně rychlý vzhledem k použití efektivních operací s binárními čísly.

F. Algoritmus je možno z časových důvodů kdykoliv zastavit a vzít si nejlepší dosud nalezené řešení.

G. Celý postup lze zobecnit a nalézt i jiné použití. Například při klasifikaci prvků pospojovaných vazbami do dvou hlavních skupin a případně i do podskupin.

Nevýhody:

A. Na výstupu z algoritmu není zachováno původní uspořádání indexů v indexové funkci.

B. Algoritmus je sice ve většině situací rychlý, ale nelze přesně určit, kdy skončí.

C. Navrhované přeuspořádání není vhodné pro kompresní algoritmy mající již v sobě zabudován nějaký jednoduchý algoritmus k přerovnání indexů, např PNG. Oba algoritmy pak působí proti sobě.

D. Algoritmus ke své činnosti potřebuje určitou paměť a předzpracování nějakou byť i krátkou dobu trvá. Proto je potřeba zvážit jeho použití pro aplikace běžící v reálném čase.

E. Kritérium zatím neohodnocuje konfigurace v bitových rovinách ležících nad aktuální bitovou rovinou.

F. I přes použité heuristiky projevuje algoritmus v některých případech sklony k uváznutí v lokálním extrému a nenalezne globální optimum.

7.3 Dvou a vícerozměrné metody pro rychlé zpracování binárních dat

V této části práce se mi podařilo ukázat, že původně dvourozměrné reprezentace prof. Schlesingera lze poměrně snadno rozšířit do více rozměrů. Podařilo se mi dokázat a ověřit, že u vícerozměrných reprezentací lze dělat všechny binární operace s komprimovanými daty.

Podařilo se mi uvést do provozu funkční implementaci experimentální knihovny pro 3D koutečky a ověřit správnost předchozích teoretických předpokladů.

7.4 Náměty pro další výzkum

Při popisu algoritmů byla v textu označena slabá místa a bylo podrobně popsáno jakým způsobem lze příslušné algoritmy zlepšit. Od mých dalších návrhů nelze očekávat revoluční zvýšení kompresního poměru, ale mohly by způsobit alespoň jeho mírné zvýšení. Touto poznámkou odkazuji na patřičná místa v textu. Z časových důvodů nebyly některé popsané náměty implementovány. Výsledkem mých posledních experimentů je, že při kombinaci prediktoru s technikou JBIG [ISO92] by mohlo být dosaženo asi o 5% lepšího kompresního poměru.

Při práci na kompresi dat jsem jako vedlejší produkt definoval 3 rozměrné reprezentace koutečkového typu s možností rozšíření do n rozměrů. Pokud mi bude dopřáno trochu času, tak bych rád dovedl tuto problematiku do zdárného konce.

Kapitola 8

Resume

8.1 Compression of grey images

The new method for lossless image compression of grey-level images is proposed in the first part of my doctoral thesis. The image is treated as stacked bit planes. Its compressed version is represented as residuals of a non-linear local predictor spanning from the representative point in the current bit plane and a few neighbouring ones. Predictor configurations are grouped into couples that differ in one bit in the representative point only. The occurrence frequency of predictor configurations is checked in the input image. The predictor adapts automatically to the image, it is able to estimate the influence of cells in the neighbourhood and thus copes even with complicated structure or fine texture.

Residuals between original and predicted image are those that correspond to the less frequent predictor configurations. Effectively coded residuals constitute the output image. To our knowledge, the proposed compression method supersedes methods of others in performance.

8.2 Compression of palette images

Second part of my doctoral thesis relates to lossless compression of pseudocolor images (images with a palette). The proposed method is a preprocessing step preceeding actual compression. Indices in the palette are semioptimally permuted during preprocessing. For actual image compression, our own nonlinear predictor based method is used [HF97a] but the proposed invisible palette modification is relevant to most of other compression techniques too. Experiments with numerous images show that indices reordering in the palette yields data savings from 10 to 50 % for typical images.

We suggest a preprocessing phase that (a) analyses statistics of the adjacency relations of index values, (b) performs optimization, and (c) permutes indices to palette to achieve more smooth image. The smoother image causes that the

lossless image compression methods yield less output data. The task to optimally permute indices is a NP complete combinatorial optimization. Instead of checking all possibilities, we propose a reasonable initial guess and a fast suboptimal hill climbing optimization.

8.3 Significant Speed up of Image Processing Based on n-Dimensional Differential Representation

The last part of my master thesis proposes set of novel methods for **fast manipulation** with n dimensional binary data. The main trick is not to process image as raster of pixels but its **differences**. It is possible to apply following operations on compressed image: AND, OR, XOR, NOT, shift left, shift right. The complexity does not depend on the size of the image but on the number of nonzero differences only. The speedup can be significant in many cases.

This work is based on research of Prof. Schlesinger from Ukraine. I extend his approach to n dimensions. New differential image tool is **strictly hierarchical**. This feature eases maintaining the software. New procedures for the dimension n can be build on procedures from lower dimensions only. Three dimensional tool may be useful e.g. in processing tomographic data.

Příloha A

Komentované popisy důležitých částí komprimačního algoritmu

Po přečtení teorie obsažené v předchozí části této práce by mohlo být obtížné a dokonce zbytečné pokoušet se o další implementaci komprimačního algoritmu znovu úplně od začátku. Proto jsem považoval za důležité podrobně popsat nejzajímavější části algoritmu. Samozřejmě uznávám, že nejsou napsány tím nejoptimálnějším možným způsobem a uvítám, pokud někdo implementuje mnou popsaný algoritmus lépe. Ale i pro takového člověka by mohla moje implementace představovat důležité vodítko.

Pro jednoduchost byl zvolen jako programovací jazyk PASCAL. Jazyk C++ je jednoznačně lepší pro tvorbu větších projektů. Tato volba byla provedena z důvodu, že se mi jeví zdrojový kód v PASCALU o něco přehlednější. Přepsání algoritmu z Pascalu např. do jazyka C je velmi přímočaré (což o opačném převodu již zdaleka říci nelze).

Jak již bylo řečeno jsou z důvodů vyšší přehlednosti a zaměření se na podstatné části popsány pouze fragmenty kódu. Z popsaných fragmentů by již při troše šikovnosti neměl být problém sestavit funkční komprimační algoritmus.

A.1 Schlesingerův analyzátor četností

Účelem analyzátoru četností je vypočítat počty výskytů jednotlivých konfigurací sondy. O teorii analýzy četností je pojednáno např. v sekci 3.6.

Celá analýza četností je realizována procedurou `SchCount(var obr:Picture; var Smask:Tsmask):Longint;`. Tabulka četností je vypočítávána pro sondu podle obr. 3.4.

Vstupem do funkce `SchCount` je samozřejmě binární obrázek předávaný ve struktuře `obr`. Detaily o obsahu objektu `obr` nejsou důležité a mohou se lišit v různých implementacích. Objekt musí nabízet proceduru (popřípadě metodu) `Pixel`, která vrací hodnotu pixelu (buňky) na pozici x, y. Dále by měl umožnit

přístup k ukazatelům na binární data jednotlivých řádek. V naší implementaci je přístup proveden přes pole ukazatelů `data`.

Výstupem je tabulka četností umístěná do struktury `Smask` již v zahuštěném tvaru. To znamená, že každému sloupečku v původní tabulce četností již odpovídá pouze jediný bit. Dalším výstupem je hodnota funkce, která nám dává informaci o počtu všech reziduí v obrazu `obr`, který by byl redukován navrhovaným prediktorem.

Popis důležitých proměnných v proceduře `SchCount`:

Proměnné `x` a `y` ukazují na aktuální pozici v rastru pixelů.

Aktuální konfigurace sondy je uložena v proměnné `i`. Každý bit proměnné `i` odpovídá jedné buňce sondy. Pro každou další pozici je proměnná `i` aktualizována.

Celá statistika o konfiguracích sond je ukládána do pole `a`. Na konci výpočetní fáze jsou pro jednotlivé páry sond všechny četnější konfigurace nalezeny a informace o pozici četnější konfigurace v páru je uložena do proměnné `Smask`, která obsahuje úplný popis nastavení binárního prediktoru.

```
Function SchCount(var obr:Picture;var Smask:Tsmask):Longint;
var x,y:Word;
    mask,ii,i:byte;
    a:array[0..15] of longint;
    residuals:longint;
    p1,p2:^Byte;

    pi,e:real;
    n:longint;
begin
if ((obr.Planes<>1)or(not(Obr.Valid))) then exit;

fillchar(a,sizeof(a),0);
for y:=0 to Obr.y-2 do                {hlavní řádková smyčka}
     begin
     p1:=pointer(Obr.data^[y]);    {načti ukazatel na první řádek}
     p2:=pointer(Obr.data^[y+1]);  {načti ukazatel na druhý řádek}
     mask:=64;                     {nastav masku dat}
      {proměnná 'i' obsahuje otisk sondy v~aktuální pozici}
     i:=2*Pixel(obr,0,y)+8*Pixel(obr,0,y+1); {nastav 'i' na -1 řádek}
     for x:=0 to Obr.x-2 do        {hlavní sloupcová smyčka}
         begin
         i:= ((i shr 1)and 5);     {aktualizuj otisk v~'i'}
         if (p1^ and mask)<>0 then i:=i or 2;
         if (p2^ and mask)<>0 then i:=i or 8;
         mask:=mask shr 1;         {aktualizuj datovou masku pro příští sloupec}
         if mask=0 then            {Přecházíme hranici jednoho bajtu?}
                  begin
                  mask:=128;       {aktualizuj binární masku pro tento případ}
                  inc(p1);         {aktualizuj také oba datové ukazatele}
```

```
                inc(p2);
                end;
          inc(a[i]);                {inkrementuj počet konfigurací}
          end;                      { shodných s~aktuální konfigurací sondy}
      end;

Smask:=[];              {nyní urči popis prediktoru}
residuals:=0;           {a počet méně četných konfigurací, tedy kvalitu prediktoru}
for i:=0 to 7 do        {každý pár sond má své vlastní místo (máme 7 párů)}
    if a[i+8]>a[i] then
            begin               {označ více četnou konfiguraci}
            Smask:=Smask + [i];  {a podle méně četné}
            inc(residuals,a[i]); {konfigurace vypočti}
            end
    else inc(residuals,a[i+8]);  {počet reziduí prediktoru}
SchCount:=residuals;    {vrať počet reziduí pro optimální prediktor podle 'Smask'}
end;
```

A.2 Schlesingerův prediktor tvořící rezidua

Procedura `SchComp` provádí vlastní redukci binárního obrázku nebo jedné bitové roviny pocházející z víceúrovňového obrázku na řídkou matici reziduí. Na proceduře je zajímavé, že komprimace je prováděna do stejné datové struktury a původní data jsou postupně nahrazována rezidui. Pro splnění tohoto úkolu musí být postupováno pozpátku (oproti směru při dekomprimaci).

Procedura má dva vstupy. Prvním z nich je struktura `Obr`, ve které jsou uložena data z aktuální matice a jako druhý vstup je vyžadován popis prediktoru ve struktuře `Smask`. Ten lze získat buď jako výsledek analýzy četností realizované např. procedurou `SchCount` nebo lze zvolit jakékoliv jiné nastavení (samozřejmě s horšími výsledky měřenými počtem reziduí).

Způsob přepisování originálních dat rezidui si zaslouží trochu pozornosti. Z následující analýzy lze zjistit, že existují čtyři základní možnosti:

1. Na aktuální pozici se ukládá reziduum (označme 1).
   (a) Aktuální buňka má hodnotu 0.
   (b) Aktuální buňka má hodnotu 1.
2. V aktuální pozici reziduum býti nemá (označme 0).
   (a) Aktuální buňka má hodnotu 0.
   (b) Aktuální buňka má hodnotu 1.

Z uvedeného výčtu je patrné, že ve skutečnosti postačí přepisovat data v originální bitové rovině pouze v polovině případů. Pro zbylé dva případy se hodnota rezidua kryje s originální hodnotou v bitové mapě. Uvedená informace nám

umožní optimalizovat algoritmus a zmenšit počet přístupů k datům asi o 1/2. Pro lepší pochopení je zbytečná část kódu ponechána v proceduře jako komentář.

Celá procedura pro komprimaci se příliš neliší od funkce pro výpočet analýzy četností `SchCount` popsané v A.1. V proměnné `i` je obsažen popis aktuální sondy. Ukazatele `p1` a `p2` obsahují adresu aktuální pozice v obou řádcích zahrnutých sondou. Obsah proměnné `Rmask` koresponduje s aktuální pozicí sondy uvnitř jednoho bitu. Proměnné `x` a `y` označují aktuální pozici sondy v obrázku.

```
Procedure SchComp(var obr:Picture;Smask:Tsmask);
var x,y:Word;
    Rmask,i,ii:byte;
    p1,p2:^Byte;

begin
if ((obr.Planes<>1)or(not(Obr.Valid))) then exit; {Je obrázek v~pořádku?}

for y:=Obr.y-2 downto 0 do {hlavní řádková smyčka}
    begin
    p1:=pointer(Obr.data^[y]);        {načti ukazatel na první řádek}
    p2:=pointer(Obr.data^[y+1]);      {načti ukazatel na druhý řádek}
    inc(p1,(Obr.x-2) shr 3);          {a přesuň oba ukazatele na konce}
    inc(p2,(Obr.x-2) shr 3);          {obou řádků}
    Rmask:=128 shr ((Obr.x-2) and 7); {nastav bitovou datovou masku}
                  {proměnná 'i' obsahuje otisk sondy v~aktuální pozici}
    i:=Pixel(obr,Obr.x-1,y)+4*Pixel(obr,Obr.x-1,y+1); {obr.x-1 sloupec}
    for x:=Obr.x-2 downto 0 do        {hlavní sloupcová smyčka}
          begin
          i:= ((i shl 1)and $A);     {občerstvi otisk sondy v~'i'}
          if (p1^ and rmask)<>0 then i:=i or 1;
          if (p2^ and rmask)<>0 then i:=i or 4;
          rmask:=rmask shl 1;        {obnov datovou masku pro příští sloupec}
          if rmask=0 then            {Přecházíme hranici celého bajtu?}
              begin
              rmask:=1;          {aktualizuj binární masku pro tento případ}
              dec(p1);           {aktualizuj také oba datové ukazatele}
              dec(p2);
              end;

          if ((i and 7) in Smask) then
              begin {Nastala situace, kdy je třeba změnit hodnotu aktuální buňky.}
              if (i>=8) then SetPixel(obr,x+1,y+1,0)  {aktuální=1; smaž pixel}
                        else SetPixel(obr,x+1,y+1,1); {aktuální=0; nastav reziduum}
              end

{ Upozornění: Nastala situace, kdy již aktuální buňky obsahuje žádanou hodnotu, a proto
  není co měnit! Proto jsou následující příkazy zakomentovány.}
    {     else
              begin                  již nastaveno!
              if (i>=8) then SetPixel(obr,x+1,y+1,1)  aktuální je již 1
                        else SetPixel(obr,x+1,y+1,0); aktuální je již 0
```

```
              end;}
         end;
    end;
end;                                    {konec procedury SchComp}
```

A.3 Zpětná transformace z reziduí na původní data

Procedura `DeSchless` konvertuje reziduální bitovou mapu zpět na původní bitovou mapu. Realizuje inverzní operaci k proceduře `SchComp`.

Také procedura `DeSchless` vyžaduje předání popisu prediktoru v argumentu `Smask`, kterým byla data redukována. Vnitřní struktura procedury je velmi podobná předchozím dvěma procedurám, a proto není potřebné její popis již dále rozebírat.

Poznamenejme, že první řádek a první sloupec nesmí být v zakódovaném tvaru před voláním procedury `DeSchless`. Obrazová data jsou postupně rekonstruována a ukládána na místa původních reziduí. Postup obrazovými daty při rekonstrukci je opačný než postup při tvorbě reziduí.

```
Procedure DeSchless(var obr:Picture; Smask:Tsmask);
var x,y:Word;
    ii,i,Rmask:byte;
    p1,p2:^byte;
begin
if ((obr.Planes<>1)or(not(Obr.Valid))) then exit;

for y:=0 to Obr.y-2 do                  {hlavní řádková smyčka}
     begin
     p1:=pointer(Obr.data^[y]);         {načti ukazatel na první řádek}
     p2:=pointer(Obr.data^[y+1]);       {načti ukazatel na druhý řádek}
                          {proměnná 'i' obsahuje otisk sondy v~aktuální pozici}
     i:=2*Pixel(obr,0,y)+8*Pixel(obr,0,y+1);
     Rmask:=64;                         {nastav bitovou datovou masku dat}

     for x:=0 to Obr.x-2 do             {hlavní smyčka pro sloupce}
          begin
          i:= ((i shr 1)and $55);       {občerstvi otisk sondy v~'i'}
          if (p1^ and Rmask)<>0 then i:=i or 2;
          if (p2^ and Rmask)<>0 then i:=i or 8;
          Rmask:=Rmask shr 1;           {obnov datovou masku pro příští sloupec}
          if Rmask=0 then               {Přecházíme hranici celého bajtu?}
               begin
               Rmask:=128;              {Aktualizuj pro tento případ binární masku}
               inc(p1);                 {a také oba ukazatele}
               inc(p2);
               end;
```

```
  if ((i and 7) in Smask) then
                begin                    {rezidua(minoritní)=0; prázdné(majoritní)=1}
                if (i>=8) then
                    begin            {Reziduum nalezeno}
                    SetPixel(obr,x+1,y+1,0); {méně častá konfigurace zde má 0}
                    i:=i and 7;
                    end
                else
                    begin
                    SetPixel(obr,x+1,y+1,1); {četnější konfigurace zde má 1}
                    i:=i or 8;
                    end;
                end;
{    else                minoritní=1; majoritní=0}
                         {ale toto je již nastaveno-ponechá se beze změny}
          end;
     end;
end;
```

Příloha B

Slovník použitých termínů

Vzhledem k nejasnosti některých termínů jsem se rozhodl blíže vysvětlit méně časté termíny použité v této práci.

abeceda Množina všech použitelných symbolů v rámci jedné zprávy.

adaptivní kódování Typ prediktivního kódování, u něhož se prediktor plynule mění v závislosti na přicházejících datech.

b (postfix) Označení čísla, které je zapsáno ve dvojkové soustavě např. $1100b = 12$.

bitová hloubka Počet uspořádaných bitů potřebných pro popis jednoho pixelu.

bitová mapa Datová struktura, která uchovává rastrový binární obraz. Data jsou uspořádána do dvou rozměrů.

bitová rovina Bitová mapa, která vznikla z n-tého bitu každého pixelu.

buňka V kontextu buňka označuje elementárnější datovou strukturu než pixel tzn. pixel se může skládat z více buněk. Zavedení buňky umožňuje též lepší přenesení úvah do vícerozměrného prostoru bez zavedení nechtěných předpokladů o jeho dimenzi. Pixel $\sim$ 2D; voxel $\sim$ 3D atd. U binárních obrázků je buňka totožná s pixelem.

CMY Barevný model sloužící k popisu barvy nějaké oblasti v tzv. subtraktivním tvaru. Barva je vyjádřena pomocí trojice hodnot C=modrozelená (cyan), M=magenta (fialová) a Y=žlutá (yellow).

CMYK Barevný model sloužící k popisu barvy nějaké oblasti v tzv. subtraktivním tvaru. Barva je vyjádřena pomocí čtveřice hodnot C=modrozelená (cyan), M=magenta (fialová), Y=žlutá (yellow) a K=černá (black). Model je používaný pro tisk v tiskárnách, poněvadž smícháním složek CMY nelze prakticky docílit černé barvy.

data Blíže nespecifikovaná část nebo části zprávy obsahující nějakou informaci. Data lze při kódování rozložit na elementární buňky.

DOF (z anglických slov Degree Of Freedom) Jedná se o zkratku sloužící k označení počtu rozměrů v datech.

h (postfix) Označení čísla, které je zapsáno v šestnáctkové (hexadecimální) soustavě např. $0A5h = 165$. Pokud hexadecimální číslo začíná některým ze symbolů A, B, C, D, E, F píše se před tímto symbolem znak 0.

matice indexů Analogie obrazového rastru u pseudobarevných obrázků. Na místě uspořádaných pixelů obsahujících červenou zelenou a modrou složku jsou však umístěny indexy do palety.

Metrika je funkce definovaná na kartézském součinu dvou prostorů $x \times x$. Její funkční hodnotou je skalární veličina. Skalární veličina je v textu nazývána vzdálenost.

nestacionární zdroj Běžný zdroj dat generuje zprávy s různou četností, která je neměnná. U nestacionárního zdroje se četnost jednotlivých zpráv pomalu mění s časem.

operátor Operátor představuje obecně funkci pro transformaci dat. Funkce (operátory) mohou být vzájemně kombinovány za účelem vytvoření složitějších operátorů.

pevný model dat Typ prediktivního kódování, u něhož se prediktor nemění v závislosti na datech.

pixel Vychází z anglických slov picture element což odpovídá českému ekvivalentu obrazový bod. Obrazový bod popisuje barvu či stav čtvercové (obdélníkové nebo i jiné) oblasti o určité velikosti. Pro plný popis obrazového bodu může být v některých typech obrazových dat potřeba většího množství čísel (např. RGB barva je složena z červené, zelené a modré).

plné barvy Způsob ukládání barevných obrazových dat, při němž jsou uloženy jednotlivé barevné složky RGB odděleně.

plovoucí okno Plovoucí okno je datová struktura, která uchovává informace o předchozích datech do určité hloubky. Struktura je využívána plně adaptivním algoritmem k jeho nastavování.

proud Vstupní řetězec n stejných po sobě jdoucích znaků k u kódování RLC. Číslo k je nazýváno *hodnota proudu*.

pseudobarvy Vzhledem k velké paměťové náročnosti kódování barevných grafických dat v plných barvách je prováděn výběr několika základních barev z celého spektra. Vybrané barvy jsou ukládány jako indexy do palety. Obrázek obsahující paletu je často nazýván jako pseudobarevný.

obrazový rastr Matice pixelů, které jsou uspořádány do dvourozměrné obdélníkové struktury.

redundance Nadbytečná informace, které je výsledkem činnosti zdroje dat, který generuje nějaké zprávy častěji než jiné.

reziduální bitová mapa Datová struktura, která uchovává rastrový binární rozdílový obraz. Obraz vznikne jako rozdíl mezi původní bitovou mapou a predikovanou bitovou mapou. Data jsou uspořádána do dvou rozměrů.

reziduální bitová rovina Bitová mapa, která vznikla z n-tého bitu každého pixelu a následně byla převedena na reziduální bitovou mapu.

reziduum Místo, ve kterém nedošlo ke shodě predikované a skutečné hodnoty. Někdy též rozdíl mezi predikovanou a skutečnou hodnotou.

RGB Barevný model sloužící k popisu barvy nějaké oblasti v tzv. aditivním tvaru. Barva je vyjádřena pomocí trojice hodnot R=červené (red), G=zelené (green) a B=modré (blue).

semiadaptivní Typ prediktivního kódování, u něhož se prediktor pevně nastaví po jednom průchodu daty. Poté se stane popis prediktoru součástí zakódované zprávy.

range-finger Zařízení určené ke snímání třírozměrných obrazů.

RLE Název pochází ze slov Run Length Encoded a jedná se o datové struktury složené ze dvou čísel: hodnota a počet opakování.

roura (anglicky pipe) Označuje v systémech typu Unix datovou strukturu, do které se na jedné straně posílají data a z druhé se vybírají. Z hlediska programu tato struktura vypadá jako soubor. Na struktuře roury je v Unixových systémech založena synchronizace více procesů na data.

Schlesingerův prediktor Prediktor vyvinutý prof. Schlesingerem, který predikuje data v binárních dvouúrovňových obrázcích na základě analýzy četností výskytů malého okénka (sondy).

steganografie Steganografie je nauka zabývající se ukrýváním dat do jiných dat. Do digitálních obrazových a též zvukových dat je možno ukrýt např. textové zprávy.

voxel Vychází z anglických slov volume element. Termín by bylo možno přeložit jako třírozměrný (3D) obrazový bod. 3D obrazový bod popisuje barvu nebo jiný atribut kvádrové (popřípadě jiné) oblasti o určité velikosti. Pro plný popis 3D obrazového bodu může být v některých typech obrazových dat potřeba většího množství čísel (např. RGB červené, zelené a modré).

vycpávka Úmyslně vložené prázdné místo v datech. Nejčastěji se používá pro zarovnání, které vyhovuje nějaké architektuře počítače nebo SW vybavení. Nejčastěji je vycpávka přidána za účelem zarovnání na sudé bajty, čtveřic bajtů, násobky 16, 256 atd. Je vhodná pro použití v dočasných strukturách, ale u trvale uložených dat zbytečně zvyšuje jejich objem.

Rejstřík

Literatura

[Adá89] Jiří Adámek. *Kódování*. SNTL, Praha, 1989.

[Ani89] Jain Anil, K. *Fundamentals of Digital Image Processing*. Prentice Hall, Englewood Cliffs, NJ, 1989.

[AT94] R.B. Arps and T.K. Truong. Comparision of international standards for lossless still image compression. *Proceedings of the IEEE*, 82(6):889–899, June 1994.

[AT96] R.B. Arps and T.K. Truong. A new fast and effective image codec based on set partioning in hierarchical trees. *IEEE Transactions on Circuits and Systems for Video Technology*, 6, June 1996.

[BHP+00] Luboš Bureš, Karel Hanton, Jan Paleček, Tomáš Telenský, Václav Hlaváč, and Michail I. Schlesinger. Corners toolbox allowing processing binary images in a compressed form. In Svoboda Tomáš, editor, *Czech Pattern Recognition Workshop*, volume I, pages 81–87, Center for Machine Perception, Department of Cybernetics, Czech Technical University, Karlovo náměstí 13, Prague, Czech, February 2000. CMP.

[CAJ96] A.K. Chaudhary, J. Augustine, and J. Jacob. Lossless compression of images using logic minimization. In P. Delogne, editor, *Proceedings of the International Conference on Image Processing*, volume II, pages 77–80, Lausanne, Switzerland, September 1996. IEEE Signal Processing Society, Louvain-La-Neuve, Belgium.

[Car97] Bruno Carpentieri. A new lossless image compression algorithm based on aritmetic coding. In Bimbo Alberto, Del, editor, *Image Analysis and Processing*, volume II, pages 54–61, Florence, Italy, September 1997.

[CL97] Kuo-Liang Chung and Yih-Kai Lin. A novel memory-efficient huffman decoding algorithm and its implementation. *Signal Processing: Image Communication*, (62):207–213, June 1997.

[Dv94] James D.Murray and William vanRyper. *Encyclopedia of Graphics File Formats.* O'Reilly & Associates, Inc., Sebastopol, California, USA, 1994.

[Foj97] Jaroslav Fojtík. Adaptive non-linear predictor for lossless image compression. Technical Report K335-1997-CMP-140, ČVUT FEL, FEL ČVUT Karlovo Náměstí 13, October 1997.

[Fry93] M. Frydrych. Image compression. Master's thesis, Charles University, Faculty of Mathematics and Physics, Prague, Czech Republic, 1993.

[HF97a] V. Hlaváč and J. Fojtík. Adaptive non-linear predictor for lossless image compression. In G. Sommer, K. Daniilidis, and J. Pauli, editors, *Proceedings of the conference Computer Analysis of Images and Patterns'97, Kiel, Germany*, pages 279–288. Springer-Verlag, LNCS 1296, September 1997.

[HF97b] V. Hlaváč and J. Fojtík. Adaptive non-linear predictor for lossless image compression. In Tomáš Pajdla CTU Prague, editor, *Czech Pattern Recognition Workshop*, pages 111–120, Milovy, Czech Republic, February 17-20 1997. CMP.

[HF97c] V. Hlaváč and J. Fojtík. Predictor based on frequency analysis of the local configurations used for lossless image compression. In *Proceedings of the 1st IAPR TC1 workshop on Statistical Techniques in Pattern Recognition, Prague, Czech Republic, June 9-11, 1997*, pages 73–78, Prague, Czech Republic, June 1997. Institute of Information Theory and Automation, Czech Academy of Sciences.

[HF99] Václav Hlaváč and Jaroslav Fojtík. Adaptive predictor for lossless image compression. *Computing*, 62(4):339 – 354, June 22 1999.

[HS94] Andrew C. Hadenfeldt and Khaid Sayood. Compression of color-mapped images. *IEEE Transactions on Geoscience and Remote Sensing*, 32(3):534–541, May 1994.

[HS92] R. M. Haralick and L. G. Shapiro. *Computer and Robot Vision, Volume I.* Addison Wesley, Reading, Ma, 1992.

[HŠ92] V. Hlaváč and M. Šonka. *Počítačové vidění.* Grada Praha, 1992.

[HV93] Paul G. Howard and Jeffrey Scott Vitter. Fast and efficient lossless image compression. In *Proceedings of the International Conference/NASA/CESDIS Data Compression Conference*, pages 351–360, Snowbird, Utah, April 1 1993.

[ISO93] A fast moving international standard for still image compression. *ISO bulletin: JPEG alias ISO/IEC 10918-1*, December 1993.

[ISO94] ISO/IEC 10918-1: Information technology-digital compression and coding of continuous-tone still images: Requirements and guidelines, 1994.

[ISO92] ISO/IEC 11544: Coded representation of picture and audio information - progressive bi-level image compression, April 1992.

[JCP96] H.Y. Jung, T.Y. Choi, and R. Prost. Rounding transform for lossless image coding. In P. Delogne, editor, *Proceedings of the International Conference on Image Processing*, volume II, pages 65–68, Lausanne, Switzerland, September 1996. IEEE Signal Processing Society, Louvain-La-Neuve, Belgium.

[KD97] Omid Kia and David Doermann. Integrated segmentation and clustering for enhanced compression of document images. In Alberto Del Bimbo, editor, *International Conference on Document Analysis and Recognition*, volume 1, pages 406–411, Ulm, Germany, August 18-20 1997.

[Kol97] Petr Kolář. Nový grafický formát PNG (in czech). *Softwarové noviny*, (2):75–79, February 1997.

[Kop95] Manfred Kopp. Lossless wavelet based image compression with adaptive 2D decomposition. Technical Report TR-182-2-95-11, TUV, Technical University of Vienna, Institute of Computer Graphics, Karlsplatz 13/186-a, A-1040 Wien, Austria, February 1995.

[Mel96] Bořivoj Melichar. *Textové informační systémy (in czech) (Text information systems)*. Czech Technical University, Faculty of Electrical Engineering, Prague, Czech Republic, 1996.

[MV96] Nasir D. Memon and Ayalur Venkateswaran. On ordering color maps for lossless predictive coding. *IEEE Transactions on Image Processing*, 5(11):1522–1527, November 1996.

[NJ98] F. Johnson Neil and Sushil Jajodia. Exploring steganography: Seeing unseen. *Computer*, (98.02):26–34, February 1998.

[Pav98] Janík Pavel. Co je to bz2? *Linuxové noviny*, (1):5–6, January 7 1998.

[PM96] P. Piscaglia and B. Macq. Multiresolution lossless compression scheme. In P. Delogne, editor, *Proceedings of the International Conference on Image Processing*, volume II, pages 69–72, Lausanne, Swi-

tzerland, September 1996. IEEE Signal Processing Society, Louvain-La-Neuve, Belgium.

[RA96] Krishna Ratakonda and Narendra Ahuja. Segmentation based reversible image compression. In P. Delogne, editor, *Proceedings of the International Conference on Image Processing*, volume II, pages 81–84, Lausanne, Switzerland, September 1996. IEEE Signal Processing Society, Louvain-La-Neuve, Belgium.

[Sch89] M.I. Schlesinger. *Matematiceskie sredstva obrabotki izobrazenij, in Russian, (Mathematic tools for image processing).* Naukova Dumka, Kiev, Ukraine, 1989.

[Ska93] Václav Skala. *Světlo barvy a barevné systémy v počítačové grafice.* Academia, Praha, 1993.

[Spe69] R. David Spencer. Bit plane encoding of continuous tone pictures. In Fox Jerome, editor, *Proceedings of the symposium on Computer processing in communications*, volume XIX, pages 101–120, Brooklyn, N. Y., April 1969. Polytechnic press of the Polytechnic Institute of Brooklyn.

[Tea96] Botuel Thomas et al. PNG (portable network graphics) specification version 1.0. Available on http://www.w3.org/Graphics/PNG/, 1996.

[TVP97] Pekka J Toivanen, Ari M Vepsäläinen, and Jussi P.S. Parkkinen. Image compression using the distance transform on curved space (dtocs) and Delaunay triangulation. In Michael Frydrych, Jussi Parkkinen, and Ari Visa, editors, *The 10th Scandinavian Conference on Image Analysis*, pages 229–236, Lappeenranta, Finland, June 9–11 1997.

[VŠ93] Mařík Vladimír and Olga Štěpánková. *Umělá Inteligence*, volume 1. Academia, Praha, 1993.

[WM97] X. Wu and Nasir D. Memon. Context-based adaptive lossless image coding. *IEEE Transactions on Communications*, 45(4):437–444, 1997.

[Xia97] Wu Xiaolin. Lossless compression of continuous-tone images via context selection, quantization and modelling. *IEEE Transactions on Image Processig*, 6(5):656–664, February 1997.

[ZL93] André Zaccarin and Bede Liu. A novel approach for coding color quantized images. *IEEE Transactions on Image Processing*, 2(4):442–453, October 1993.